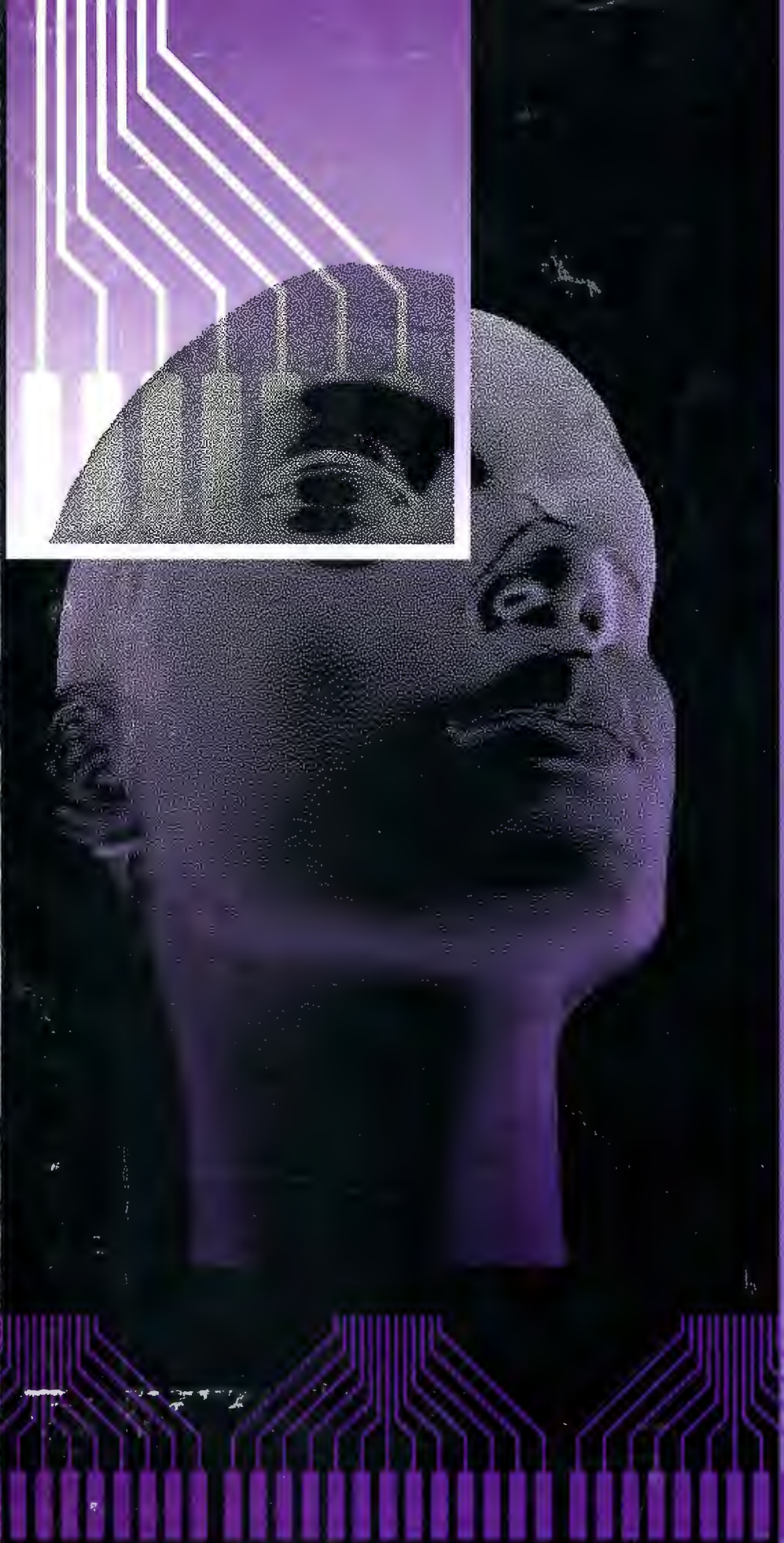


AI 94

ARTIFICIAL INTELLIGENCE

MAY 16 - 20
Banff Alberta

Proceedings

Tenth Canadian Conference on Artificial Intelligence

Actes

X[e] conférence canadienne sur l'intelligence artificielle

Sponsored by: Canadian Society for Computational Studies of Intelligence
Commanditaire : Société canadienne pour l'étude de l'intelligence par ordinateur

Edited by/Éditrice
Renée Elio

Proceedings of the Tenth Biennial Conference of the Canadian Society for Computational Studies of Intelligence

Actes de la Xe Conférence biennale de la Société canadienne pour l'étude de l'intelligence par ordinateur

edited by/sous la direction de Renée Elio

Banff Park Lodge
Banff, Alberta
16-20 May/mai 1994

Sponsored by/ Parrainée par
Canadian Society for Computational Studies of Intelligence

Supported by/Avec l'appui financier de
Natural Sciences and Engineering Research Council of Canada
Alberta Research Council
University of Alberta

In Cooperation with /et la collaboration de
University of Alberta

ISBN 0-9694596-3-7

Edited by/Sous la direction de Renée Elio

Printed by/ Imprimerie Quality Color Press, Inc. Edmonton, Alberta

Within Canada, copies of these proceedings may be obtained as follows. Send orders, together with payment of: $40 CDN each (CSCSI members), $45 CSN each (CSCSI non-members) (Add $5 CDN for postage) to:

Au Canada, prière d'adresser les commandes accompagnées du règlement en dollars canadiens (prix unitaire : 40 $ pour les membres de la SCEIO et 45 $ pour les non-membres + 5 $ de frais d'envoi) à:

CIPS
243 College Street (5th Floor)
Toronto, Ontario M5T 2Y1

Outside of Canada, please contact /
À l'extérieur du Canada, communiquer avec :

Morgan Kaufman Publishers, Inc.
Order Fulfillment Center
P.O. Box 50490
Palo Alto, California 94303
USA

CSCSI 1994 Program Committee
Comité de programme — SCEIO 1994

Program Chair / Présidente du Comité de programme

Renée Elio
Department of Computing Science
University of Alberta

Program Committee / Comité de programme

Referees / Lecteurs spécialisés

Message from the Program Chair

This volume contains the proceedings of the AI-94, the Tenth Biennial Conference of the Canadian Society for the Computational Studies of Intelligence (CSCSI/SCEIO). Since 1976, this conference has been a forum for the best recent work by both Canadian and international researchers in theoretical and applied artificial intelligence.

This year, 86 papers were submitted. The program committee found the quality of the submitted papers to be very high. Although the initial program schedule allowed for only 30 acceptances, there many more submissions that the reviewers considered to be excellent work. Therefore, the presentation times were adjusted as much as possible within the constraints of the conference schedule, allowing us to accept a total of 37 papers. Even so, several other outstanding submissions could not be included.

The AI-94 conference program is due to the efforts of many individuals. Wayne Davis and Tony Marsland served as general co-chairs for the combined AI/GI/VI conference. I am very grateful to the program committee and the additional referees for their diligent and knowledgeable reviewing. I am also indebted to Janice Glasgow and Peter Patel-Schneider for their insights and advice throughout the conference planning process. On behalf of the entire program committee, I thank our distinguished invited speakers for contributing to this conference, the workshop organizers for their efforts and enthusiasm in creating additional forums for scholarly discussions, and of course, the authors for their excellent research. Finally, I thank Janet Service, who served as my administrative assistant. Her organizational skills, efficiency, and dedication were invaluable assets to this process.

Renée Elio
Program Chair AI-94

Message de la présidente de programme

Voici les Actes de AI-94, la X^{e} Conférence biennale de la Société canadienne pour l'étude de l'intelligence par ordinateur (SCEIO/CSCSI). Depuis 1976, cette conférence sert à diffuser les meilleurs travaux récents des spécialistes canadiens et étrangers de l'intelligence artificielle théorique et appliquée.

Cette année, 86 articles ont été soumis. Le comité de programme a jugé qu'ils étaient de très haute qualité. Bien qu'initialement les organisateurs n'aient prévu de retenir qu'une trentaine de communications, les lecteurs ont été très favorablement impressionnés par un nombre bien supérieur de travaux. Nous regrettons de n'avoir pu faire place à tous, mais nous sommes toutefois parvenus à inscrire 37 articles au programme.

De nombreuses personnes ont travaillé à la réalisation du programme de la conférence. Wayne Davis et Tony Marsland ont siégé à titre de présidents généraux pour la conférence conjointe AI/GI/VI. Je suis très reconnaissante au Comité de programme et aux autres spécialistes consultés du travail diligent qu'ils ont accompli et de leur expertise. Je tiens également à exprimer ma gratitude à Janice Glasgow et à Peter Patel-Schneider qui ont prodigué leurs judicieux commentaires et conseils tout au long de la planification. Au nom du comité, je remercie les conférenciers éminents qui ont participé à cette conférence, les organisateurs d'atelier qui, à force d'efforts et d'enthousiasme, ont réussi à offrir des séances de discussion supplémentaires et, bien sûr, les auteurs qui ont fourni une excellente recherche. Finalement, merci à Janet Service, notre assistante administrative. Ses qualités d'organisatrice, son efficacité et son dévouement ont été inestimables.

Renée Elio
Présidente du programme AI-94

CSCSI 1994 Organizing Committee
Comité organisateur de la SCEIO 1994

General Co-Chairs / Présidents généraux AI/GI/VI '94

Wayne Davis
University of Alberta

Tony Marsland
University of Alberta

AI-94 Program Chair / Présidente du Comité de programme

Renée Elio
University of Alberta

Local Arrangements Chair / Organisation locale

Jan Mulder
Alberta Research Council

Treasurer / Trésorier

Peter van Beek
University of Alberta

Invited Speakers / Conférenciers invités

Bonnie Webber, University of Pennsylvania
Animated Human Agents as a Testbed for Language Understanding and Planning

Hector Levesque, University of Toronto
Knowledge, Action and Ability in the Situation Calculus

Stephen Judd, Siemens Corporate Research
Constraint Satisfaction and Neural Net Learning

CSCSI Executive Committee 1992-1994

Comité exécutif de la SCEIO 1992-1994

President/Présidente

Janice Glasgow
Queen's University

Past President/Président sortant

Ian Witten
University of Waikato

Vice-President/Vice-président

Peter Patel-Schneider
AT&T Bell Laboratories

Secretary/Secrétaire

Stan Matwin
University of Ottawa

Treasurer/Trésorier

Eric Neufeld
University of Saskatchewan

Editor/Éditeur

Roy Masrani
Alberta Research Council

Best Paper Award

The CSCSI best paper award is sponsored by the Editorial Board of *Artificial Intelligence*. It is given for the paper or papers that report important research results in a clear, well-written manner. The sponsorship by the board provides both an honorarium and a rapid review process in the journal for an extended version of the conference paper(s).

Potential award-winning papers were identified by reviewers as part of the reviewing process. These papers were then sent to all members of the program committee, who read the set of nominated papers and indicated which paper they felt was most deserving of this award.

The CSCSI Program committee is pleased to award the 1994 Best Paper Award to Craig Knoblock and Qiang Yang, for "Evaluating the tradeoffs in partial-order planning algorithms."

Prix de la meilleure communication

Décerné par la SCEIO, le Prix de la meilleure communication est parrainé par le Conseil de rédaction de *Artificial Intelligence*. Il couronne l'article ou les articles qui font état de travaux de recherche importants, dans un style limpide et élégant. L'appréciation du Conseil se traduit par des honoraires et une rapide évaluation visant la publication éventuelle d'une version plus élaborée de la communication originale.

Les articles retenus font l'objet d'une première évaluation et sont ensuite communiqués à tous les membres du Comité de programme qui désignent alors celui qui leur paraît le plus méritoire.

Le Comité de programme de la SCEIO est heureux de décerner le Prix de la meilleure communication 1994 à Craig Knoblock et Qiang Yang, pour "Evaluating the tradeoffs in partial-order planning algorithms."

CSCSI Distinguished Service Award

The executive of the Canadian Society for Computational Studies of Intelligence (CSCSI/SCEIO) is pleased to announce that Alan Mackworth of the University of British Columbia is the recipient of the 1994 CSCSI Distinguished Service Award. This award is presented biennially to an individual who has made outstanding contributions to the Canadian AI community in one or more of the following areas: community service, research, training of students, and research/industry interaction.

Alan has a long list of distinguished contributions to AI in Canada and internationally. He has done fundamental research in constraint satisfaction, vision and integrated robotic systems. He was and is a leader in the development of the University of British Columbia AI group, one of the strongest AI groups worldwide. He currently serves as Director of the UBC Laboratory for Computational Intelligence. At the national level, he was one of the founders and intellectual leaders for the Institute for Robotics and Intelligent Systems (IRIS), funded by the government of Canada through its Networks of Centres of Excellence programme.

In 1976, Alan was the program chair of the first CSCSI/SCEIO national conference at UBC. This event started a tradition of strongly technical AI conferences in Canada. He subsequently served as CSCSI/SCEIO president for 1976-78. Alan was also a driving force in bringing the 1981 International Joint Conference on AI (IJCAI) to Vancouver. This event placed Canadian AI on the international map, where it has played a prominent role ever since. He was the general chair for the 1985 IJCAI held in Los Angeles. Between 1983 and 1991, he was a member of the Board of Trustees of IJCAI, chairing it from 1983 to 1985.

Alan has received many other awards and honors, including the 1991 ITAC/NSERC award for Academic Excellence, given each year to two Canadian academics for outstanding contributions to Canadian Information Technology. He holds a fellowship with the Canadian Institute for Advance Research (CIAR) and is also a fellow of AAAI.

Beyond these many distinctions, Alan has always been a leader among his peers and a role model for younger researchers and graduate students, always serious and methodical in his work, always open and forthcoming in his discussions, always worth listening to in his presentations.

Alan is the second recipient of the Distinguished Service Award. The inaugural award was presented to John Mylopoulos in 1992.

Prix du mérite de la SCEIO

Le conseil exécutif de la Société canadienne pour l'étude de l'intelligence par ordinateur (SCEIO/CSCSI) est heureux d'annoncer que Alan Mackworth, de la University of British Columbia, est le lauréat du Prix du mérite 1994. Ce prix est décerné tous les deux ans à la personne qui s'est distinguée par le caractère exceptionnel de ses contributions à la communauté canadienne de l'IA dans un ou plusieurs des secteurs suivants : service communautaire, recherche, formation des élèves, et interaction recherche/industrie.

Les contributions d'Alan à l'IA sont nombreuses, tant au Canada qu'à l'étranger. Ses travaux de recherche fondamentale portent sur la satisfaction de contraintes, la vision robotique et les systèmes robotiques intégrés. Il a joué un rôle crucial dans la création du groupe d'IA dans son université, une des équipes les plus solides du monde. Il est présentement directeur du laboratoire d'intelligence informatique de UBC. À l'échelle nationale, il est un des fondateurs et des maîtres de l'Institut de robotique et de systèmes intelligents (IRIS) que le gouvernement du Canada a créé par l'intermédiaire du Programme des réseaux de centres d'excellence.

En 1976, Alan était le président du programme de la première conférence nationale de la SCEIO qui a eu lieu à UBC. Cet événement marque le coup d'envoi d'une série de conférences prestigieuses d'IA au Canada. En 1976-78, il a été président de la Société canadienne pour l'étude de l'intelligence par ordinateur (SCEIO/CSCSI). C'est en grande partie grâce à son initiative que l'International Joint Conference on AI (IJCAI) s'est tenue à Vancouver en 1981. Cette manifestation a contribué à placer le Canada sur la scène internationale, où notre pays continue à jouer un rôle déterminant. M. Mackworth a été président général de la IJCAI qui a eu lieu à Los Angeles en 1985. De 1983 à 1991, il a été membre du Conseil d'administration des IJCAI, et son président de 1983 à 1985.

Alan a reçu de nombreuses récompenses prestigieuses, y compris le Prix d'excellence décerné par le CCTI/CRSNG, qui couronne chaque année deux chercheurs canadiens exceptionnels. Il est membre de l'Institut canadien des recherches avancées (ICRC) et de l'AAAI.

Au-delà de tous ces honneurs, Alan a toujours été un meneur parmi ses pairs et un modèle pour les jeunes chercheurs et élèves diplômés. Sérieux et méthodique dans son travail, il reste ouvert à la discussion et mérite toujours d'être entendu.

Alan est le second lauréat du Prix du mérite de la SCEIO. Le premier prix a été attribué à John Mylopoulos en 1992.

Table of Contents / Table des Matieres

Language

Learning I

Learning II

Connectionist Learning

Problem Solving

Reasoning and Knowledge Representation I

Reasoning & Knowledge Representation II

Planning & Search I

Planning & Search II

Detecting Digressions Using a Model for Interactive Generation

Susan M. Haller
Department of Computer Science
State University of New York at Buffalo
Buffalo, New York 14260
haller@cs.buffalo.edu

Abstract

The Interactive Discourse Planner (IDP) plans text to describe and/or justify a domain plan. IDP uses the user's questions to decide how to extend its discourse plan in a way that both satisfies the user and achieves its own discourse goal. As part of this process, IDP can detect three types of user-initiated digressions. As a testbed for my model, IDP plans text to discuss driving routes.

1 Introduction

Systems that plan text and accept feedback are called *interactive generators*. Situations that call for interactive generation arise when the system is acting as an expert advisor or a tutor which must be able to explain and justify what it says. Instead of analyzing the user's questions to infer his plans and goals, these systems plan discourse using the user's questions as feedback to try to achieve their own discourse goals. One capability that an interactive generator must possess is the ability to recognize when the user's feedback initiates a digression from the discussion purpose.

The Interactive Discourse Planner (IDP) plans text to describe and/or justify a domain plan. IDP uses the user's questions to decide how to extend its discourse plan in a way that both satisfies the user's informational needs and achieves the system's discourse goal. As part of this process, IDP can detect three types of user-initiated digressions. As a testbed for my model, IDP plans text to discuss driving routes.

In the next section, I describe the interactive mode that IDP operates in, and the assumptions that were made to develop the IDP model. Next, I describe the processing model. I follow this with a discussion of the two kinds of text plan operators that IDP uses. Next, there is a brief description of the analyzer algorithm. Finally, I discuss how the IDP model is used to detect user-imposed digressions.

2 The Interactive Mode

In highly interactive settings, people often are called on to analyze and respond to vaguely articulated questions like *Why?* and *What?*, and ill-formed queries like *Why take Maple?* and *Why not Sheridan Drive?*. In a discussion that the system controls, we assume that these kinds of questions are used by the user to tell the system what part of the discourse plan has failed, and how it can be replanned to succeed. For plan discussions like giving route advice, IDP does not have a preset agenda. It plans text to describe or justify a selected driving route only in reaction to user feedback, and only until the user indicates that he is satisfied.

IDP works in an interactive mode in which the system is the primary speaker, the user is the primary listener, and the system is the uncontested expert. The formulation of text plans and processing procedures for this mode relies on two simplifying assumptions that explanation systems typically make. I refer to these as the *explanation assumptions*:

1. The explanation facility's knowledge is correct.
2. The user automatically believes what the explanation facility informs him of.

The first assumption excuses IDP from correcting its information based on interactions with users. Therefore, IDP is not concerned with reasoning about its own beliefs in a context where a user presents contradictory ones. The second assumption rules out argumentation and reasoning about the user's beliefs.

IDP tries to achieve one of two discourse goals (DGs) that have to do with the user's attitude or ability with respect to a domain plan. They are

1. To have the user be able to execute a domain plan
2. To have the user adopt a recommended domain plan

The first DG requires describing a domain plan to the user, and the second DG requires justifying a recommended domain plan. WHen the user indicates that he is satisfied, he signals IDP that its DG has been achieved. In the IDP model, the system's *intentions* are synonymous with the system's unachieved DGs.

Another simplifying assumption is the *discourse knowledge assumption*. Several researchers have argued that communication does not occur unless the listener recognizes the speaker's underlying intention [Grice, 1969; Searle, 1969; Cohen and Perrault, 1979; Allen and Perrault, 1980]. We assume that when IDP speaks,[1] the user knows the system's intention and the

[1] IDP does not have a speech synthesis component. It

discourse plan that IDP has used to try to achieve it. We assume that the user's feedback addresses the system's intent by indicating how the system should extend the discourse plan to achieve its DG.

3 Processing

3.1 Knowledge Requirements

IDP interprets the user's feedback using three types of knowledge:

1. the discourse plan that the system has used so far
2. possible expansions of the discourse plan
3. the user's domain knowledge

Following Carberry, when the system uses a discourse plan, the user has expectations for what will follow [Carberry, 1989]. Motivated by Grice's Maxim of Relation [Grice, 1975], IDP analyzes feedback using its executed discourse plan as the discourse context. IDP analyzes user feedback to determine how to expand the discourse plan further using the possibilities for expanding its discourse plan and the user's domain knowledge.

To analyze questions like *Why?*, IDP uses the most recently expanded portion of its discourse plan (called the *active path*), possibilities for expanding the discourse plan along the active path, and a set of propositions that comprise reasoning associated with the last proposition that IDP expressed. Of these propositions, the ones that the user does not know are called the *localized unknowns*. Applying the Gricean maxim of Quantity, questions like "Why?" should be coherent within this context.

3.2 The Approach to Processing

Processing proceeds by trying to find a way of expanding the discourse plan so that the user comes to know a proposition from the localized unknowns. IDP searches the active path for possible expansions of its discourse plan that do this. The following interaction demonstrates the results:

```
  U: Should I take Maple or Sheridan to go
     to the Eastern-Hills Mall?
IDP: you should take Maple.
  U: Why?
IDP: taking Maple avoids heavy traffic.
  U: Why?
IDP: since taking Maple there are fewer
     businesses than taking Sheridan,
```

Consistent with the Gricean maxim of Quantity, the more the user says, the more he feels he needs to say for the system to identify the continuation that is sought. IDP uses any additional information that the user supplies to try to recognize the discourse plan expansion that he seeks, even if it is not an immediate continuation of what was said last. The following demonstration illustrates this type of processing:

```
  U: Should I take Maple or Sheridan to go
     to the Eastern-Hills Mall?
IDP: take Maple.
  U: Why not go on Sheridan?
IDP: you could take Sheridan however,
     taking Maple avoids heavy traffic.
```

prints text out.

```
ANT-CSQNT({GOAL-ACT(?g),
           ACT-PLAN(?g, ?p),
           SECONDARY-GOAL-ACT(?g2)
           ACT-PLAN(?g2,?p)},

          ACT-PLAN(motivate(user, DO(user,?p)),
                   snsequence(advise(user, DO(user,?p)),
                              circumstantiate(ACT-PLAN(?g2,?p)),
                              say(ACT-PLAN(?g2,?p)),
                              restate(ACT-PLAN(?g2,?p)))))
```

Figure 1: A TP-operator for Motivate

In this interaction, the user's feedback indicates that he would like IDP's response to include information about the feasibility of an alternative route. IDP uses the mentioned action *go* ing *on Sheridan* to identify a discourse entity and a discourse plan than continues to address the system's intentions while providing the implicitly requested information.

4 The Text Plans

4.1 Text Plan Operators

IDP's text plan operators (TP-operators) are based on Rhetorical Structure Theory (RST) [Mann and Thompson, 1987] and are written using the SNePS Actor planning formalism [Shapiro *et al.*, 1989]. In RST, each essential text message (called the *nucleus*) can be augmented with additional information (called the *satellite*) through a *rhetorical relation*. In the planning formalism, plan operators are written as rules that state what consequents can be deduced from a set of antecedents.

Figure 1 shows a *text-plan operator* (TP-operator) for the *motivate* act. [2] In the formalism, an *act* decomposes into one or more structures of other acts called *plans*. IDP instantiates plans, preconditions, and effects for a given act by satisfying a rule's antecedents. These are the *constraints* on the plan, and the process of constraint satisfaction selects new content for the text. For TP-operators that are based on rhetorical relations, this new content is a satellite proposition that is appropriate for the relation and a given nuclear proposition. The TP-operator in Figure 1 states that if there is a domain goal-act ?g that is enacted by a plan ?p, and a secondary goal-act ?g2 that is also enacted by plan ?p, then a plan for the act of motivating the user to do ?p is a sequence of four other acts.

4.2 The Kinds of Text Plans

IDP uses a two-way classification of *text plans* (TPs) that separates those that directly address the system's DGs from ones that merely augment information that is to be, or that has been, presented. The two kinds of TPs are *discourse text plans* (DTPs) and *content-selection text plans* (CTPs). The overarching plan is always a DTP. This is consistent with Moore and Pollack's contention that a speaker always structures information in a dis-

[2] Arguments enclosed in braces,{...}, are unordered set arguments.

course with a high-level intention in mind [Moore and Pollack, 1992].

The division is based on a two-way division of the rhetorical relations that Mann and Thompson describe. A speaker relates two text spans with a *presentational relation* to increase an inclination in the hearer. In contrast, a speaker relates two text spans with a *subject-matter* relation to inform the hearer of the rhetorical relation itself. In the IDP model, DTPs are used to attempt and reattempt the achievement of the system's DGs. Since these goals have to do with affecting the user's attitudes and abilities towards domain plans, DTPs are based on speech acts and presentational rhetorical relations. The DTPs describe how to try to achieve discourse goals by selecting some minimal text content. IDP can augment this content by conveying related propositions with subject-matter rhetorical relations. CTPs are used to plan additional text with subject-matter rhetorical relations.

Figure 2(a) shows a DTP for *motivate* that IDP deduces from the TP-operator given in Figure 1. The motivate act takes the user and a nuclear clause (the user taking the Maple Road route) as its arguments. This DTP includes references to two additional CTPs that are potential growth points: *circumstantiate* and *restate*. A plan for circumstantiate is given in Figure 2(b). Since CTPs are not executed to affect the user in any way other than to provide information, the user is not an argument to acts for CTPs. IDP can only deduce a CTP for an act when there is an active *content-goal* (CG) that the plan satisfies. A constraint on all CTP-operators requires there to be an active CG to let the user know the proposition that will be the satellite in a subject-matter rhetorical relation.

As shown in Figure 2(b) as the second step in executing the circumstantiate CTP, the CG is retracted. Because the ACT-PLAN proposition is deducible only when the CG exists, the SNePS Belief Revision component (SNeBR) [Martins and Shapiro, 1988] retracts the ACT-PLAN proposition from the knowledge base as part of the execution of the CTP. This precludes the same CTP from being used twice if the system reattempts a DTP, or uses another DTP for which the same CTP is appropriate.

4.3 The Text Plan

Figure 3 shows IDP's TP for the first demonstration run in this paper. The TP has been formulated to achieve the DG of having the user adopt the plan to take Maple Road. The high-level plan is a DTP which can decompose into other DTPs and CTPs. The TP always bottoms out in the primitive act, *say*. The argument to *say* is a text message which includes a proposition as the content to be expressed. In the TP, the checks ($\surd$) mark the active path. Note that the plan for motivate has not been executed in the order indicated by the sequencing act *snsequence* (see Figure 2(a)). In particular, the second act expands to an optional CTP, which IDP does not use until it responds to *Why?* a second time.

5 The Analyzer

IDP's analyzer uses the existing TP, the active path, and the set of localized unknowns to analyze user questions and expand the TP. IDP tries to expand the TP in two ways to let the user know a localized unknown:

1. Starting with the last DTP on the active path, go through all the DTPs on the active path to try to find another way to replan a DTP on the active path.

2. Examine the last DTP on the active path to see if one of its unused CTPs can be expanded.

In the first phase, IDP analyzes *Why*-questions in relation to its own intent as represented by the DTPs along the active path. The analyzer starts with the most recently executed DTP (the last DTP on the active path), and the localized unknowns associated with it. It tries to find another way to expand a DTP along the active path that lets the user know a localized unknown. The analyzer backs up the active path testing each DTP in turn. If this fails, in the second phase, the analyzer considers augmenting the existing DTP at the informational level. This level is reflected in the CTPs. The analyzer examines the most focussed DTP on the active path to see if it can be expanded with a CTP to let the user know a localized unknown.

If IDP fails to identify a DTP expansion in the first two phases, it tries to identify a third type of expansion:

3. Examine the CTPs associated with the TP.

If one of these CTPs can be expanded to inform the user of a localized unknown, IDP executes it. However, in doing so, IDP recognizes this kind of expansion as a digression.

6 Detecting User Digressions

Digressions are one of the three types of discourse interruptions identified by Grosz and Sidner [Grosz and Sidner, 1986]. In the IDP model, a user-imposed digression occurs when his feedback no longer addresses the system's intentions. IDP demonstrates that user-imposed digressions can be detected from a model designed for a single mode of interaction by using its TP and its operator classification. IDP identifies the user's question as digressive if it is answerable by expanding a CTP *that is associated with* a TP instead of expanding its TP directly. There are three kinds of user-initiated digressions that IDP can detect in this way.

6.1 Direct Questions About Used CTPs

The first and simplest kind of digression is when the user asks a direct question about a proposition that was expressed in a CTP used to expand a DTP. The following interaction demonstrates this kind of digression and how IDP handles it:

```
(a) ACT-PLAN(motivate(user,
                      DO(user, *maple-plan)),
             snsequence(advise(user, DO(user, *maple-plan)),
                        circumstantiate(ACT-PLAN(avoid(heavy-traffic),
                                                 *maple-plan)),
                                        say(ACT-PLAN(avoid(heavy-traffic),
                                            *maple-plan)),
                                        restate(ACT-PLAN(avoid(heavy-traffic),
                                                *maple-plan)),

(b) ACT-PLAN(circumstantiate(ACT-PLAN(avoid(heavy-traffic),
                                      *maple-plan)),
             snsequence(say(OBJECT-PROPERTY(*maple-plan,
                                            fewer-businesses)),
                        forget(CONTENT-GOAL(KNOW(user,
                                                 OBJECT-PROPERTY(*maple-plan,
                                                                 fewer-businesses))))))
```

Figure 2: (a) A DTP for Motivate (b) A CTP for Circumstantiate

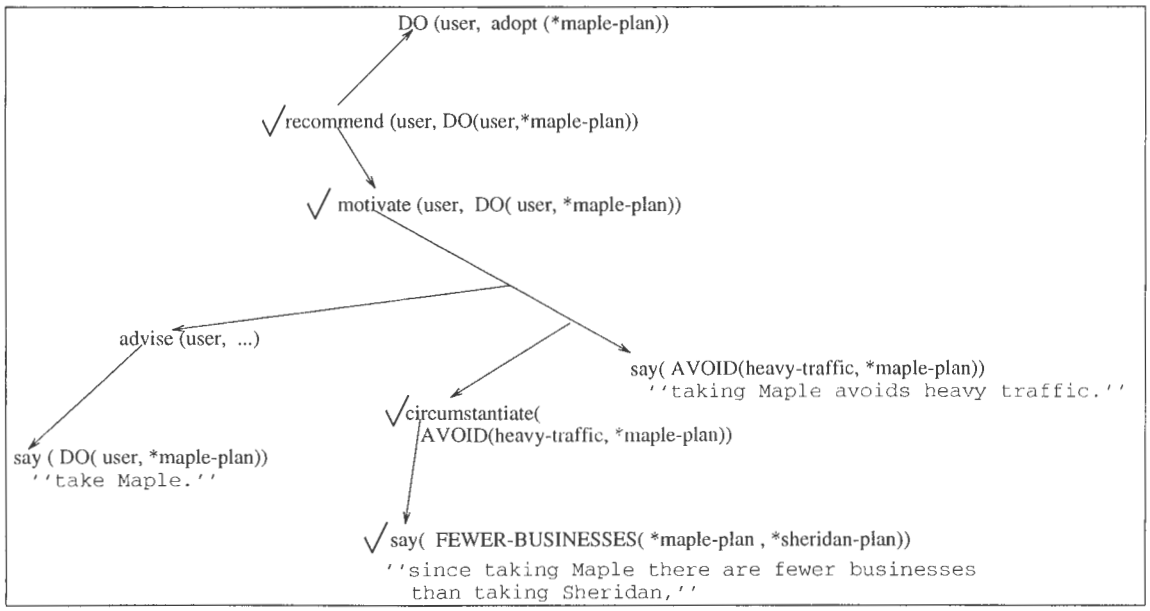


Figure 3: The TP for the First IDP Demonstration

```
U:   Should I take Maple or Sheridan to go
     to the Eastern Hills Mall?                    (1)
IDP: if now there was light traffic,               (2)
     you could take Sheridan however,              (3)
     now there is heavy traffic.                   (4)
     you should take Maple.                        (5)
     since taking Maple there are fewer
     businesses than taking Sheridan,              (6)
     taking Maple avoids heavy traffic.            (7)
U:   Why is there heavy traffic now? *             (8)
IDP: since now is rush hour.                       (9)
     as I was saying taking Maple avoids
     heavy traffic.                                (10)
```

As part of IDP's advice (line (4)), it has expressed a proposition that lets the user know a circumstances behind its recommendation. IDP has told the user that `now there is heavy traffic.` IDP identifies the user's question at line (8) (marked with an asterisk) as a digression.

Figure 4 gives IDP's TP when the user asks the question at line (8). DTPs like the overall plan to recommend the Maple Road route, are in times-roman font, the CTPs that can be used to augment the DTPs are in *italics*. All TPs bottom out in one system primitive act, say, which is shown in **boldface**. When IDP executes a say-act, it passes the proposition which is its single argument to a generation grammar. The active path (indicated by $\surd$ marks) marks the portion of the DTP that has been most recently expanded and executed.

IDP recognizes that the user's question digresses from the discourse purpose when it must expand a previously used CTP, *circumstantiate(do(user, *maple-plan))*, with another CTP (Figure 5). The dashed line indicates where the expansion has been made. This leads to the system's response at line (9). Since this expansion does not extend any of the DTPs along the active path, IDP recognizes that the user's question and its own response, digresses.

6.2 Recovering from the Digression

We do not have a theory of managing initiative. Therefore, the IDP model does not address the question of how to decide if a digression should be allowed, and for how long. When IDP detects a user-imposed digression, it answers the question, and then it immediately shifts the discussion back to its TP. In this demonstration, its intention as expressed in its DTP is to have the user adopt the plan of taking the Maple Road route.

Sidner notes that discourse markers are used by speakers to tell listeners that the next utterance conveys a new intention [Sidner, 1985]. The new intention could also be a return to an old one. To signal a return from a digression, Grosz and Sidner note that speakers use discourse markers like "anyways..." or "as I was saying...". As demonstrated by line (10) of IDP's response, it can use its own TP as content in order to do this. This is possible because IDP's TP is represented uniformly with the domain plans that are under discussion. In particular, IDP can instantiate the *restate* content-selection operator only when there has been a digression and the TP indicates that what IDP is about to say was stated previously. Figure 6 shows how IDP returns to the active path by expanding the last DTP on it (motivate) with the *restate* CTP.

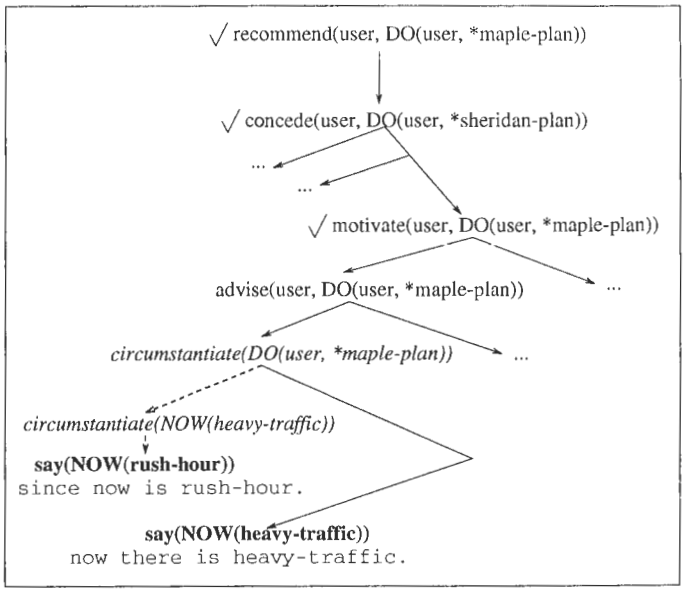


Figure 5: The System's Plan for the Response

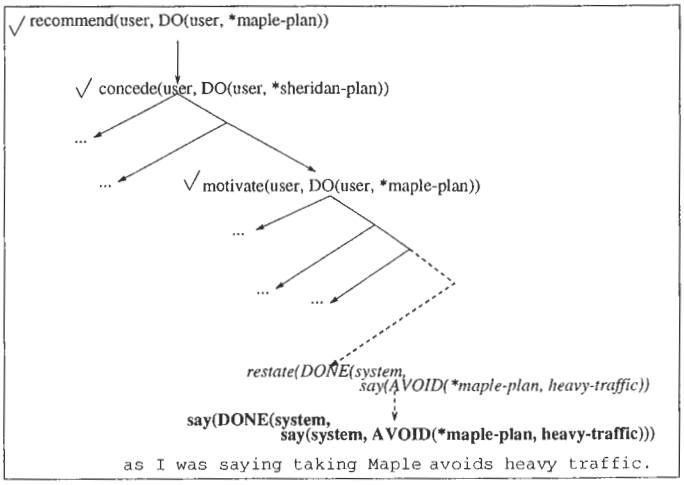


Figure 6: The System's Plan for Returning from the Digression

6.3 User Questions About Implicit Content

A second kind of digression occurs when the user refers to a proposition that was never expressed, but that is inferable from the content presented. This leads to the following interaction:

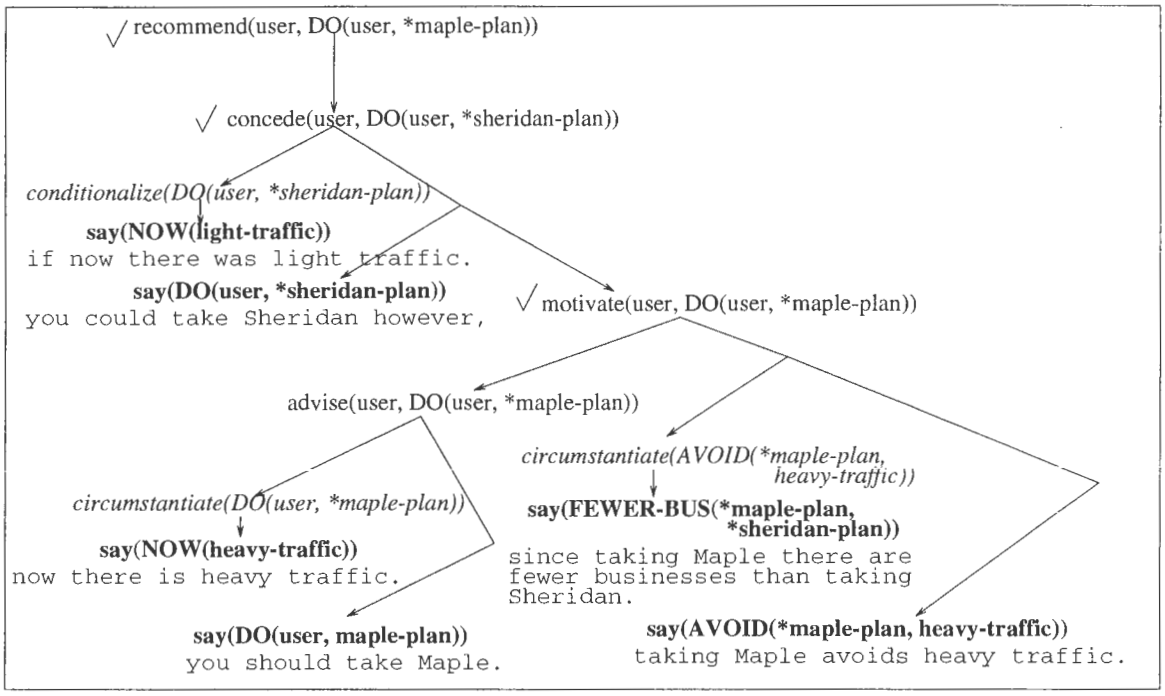


Figure 4: The System's Discourse Plan

U: Should I take Maple or Sheridan to go to the Eastern Hills Mall? (1)
IDP: take Maple. (2)
U: Why take Maple? (3)
IDP: taking Maple avoids heavy traffic. (4)
U: Why? (5)
IDP: taking Maple there are fewer businesses than taking Sheridan. (6)
U: Why is there heavy-traffic now? * (7)
IDP: since now is rush hour, as I was saying taking Maple avoids heavy traffic. (8) (9)

In this demonstration, IDP uses its DTP to detect that the user's third question, unlike the user's first two questions, initiates a digression. This is notwithstanding the fact that IDP never expressed the proposition that the user questions. The user's question at line (7) relies on an inference that he made from the proposition that IDP expressed at line (4).

The TP at the time the user asks the question is given in Figure 7. In this case, IDP cannot associate its planned response:

circumstantiate(now(heavy-traffic))

with any portion of the existing TP. It can however, check that the proposition that the user expressed at line (5) is inferable from a proposition that the system expressed at line (4). Specifically, IDP finds that the proposition that the user has inferred and questioned is a member of the set of localized unknowns for the proposition that IDP expressed with the DTP motivate at line (4). IDP concludes that the user's question is coherent, although it is digressive.

Note that digressive questions can be distinquished from questions that are incoherent in the context of the system's TP:

U: Should I take Maple or Sheridan to go to the Eastern Hills Mall?
IDP: take Maple.
U: Why is there heavy traffic now?
IDP: Huh?

IDP answered this question in our previous example. However, in this case IDP cannot find any association between the question's answer and the existing TP.

6.4 Garden-path Digressions

A third type of digression occurs when the user's line of questioning repeatedly refers to the last proposition that was expressed. This eventually leads to an explanation that does not address the IDP's original intent. The interaction below demonstrates how IDP manages digressions of this kind:

U: Should I take Maple or Sheridan to go to the Eastern Hills Mall? (1)
IDP: take Maple. (2)
U: Why? (3)
IDP: taking Maple avoids heavy traffic. (4)
U: Why? (5)
IDP: taking Maple there are fewer businesses than taking Sheridan. (6)
U: Why? * (7)
IDP: since taking Maple is a newer route than taking Sheridan. anyways taking Maple avoids heavy traffic. (8) (9)

The DTP in Figure 7 is the system's TP when the user asks *Why?* a third time. To answer this question, IDP expands its TP as shown in Figure 8. The way that IDP detects that the third why-question is digressive is similar to the first case. IDP determines that the only way the question's answer can be associated with the DTP

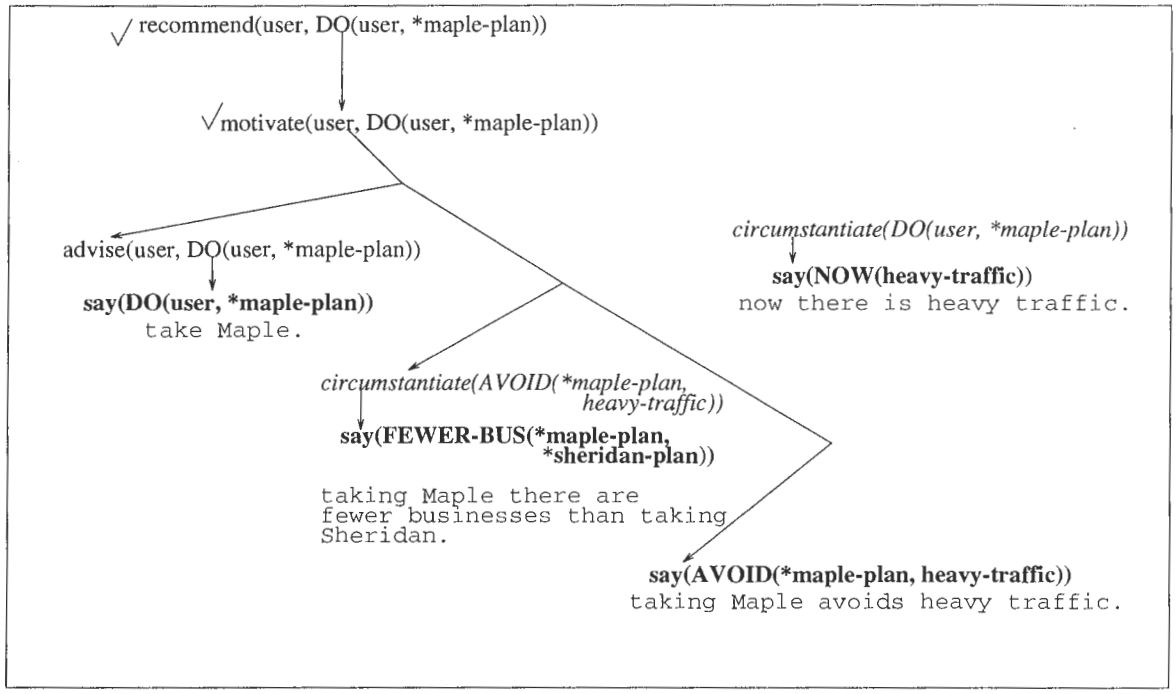


Figure 7: The System's TP

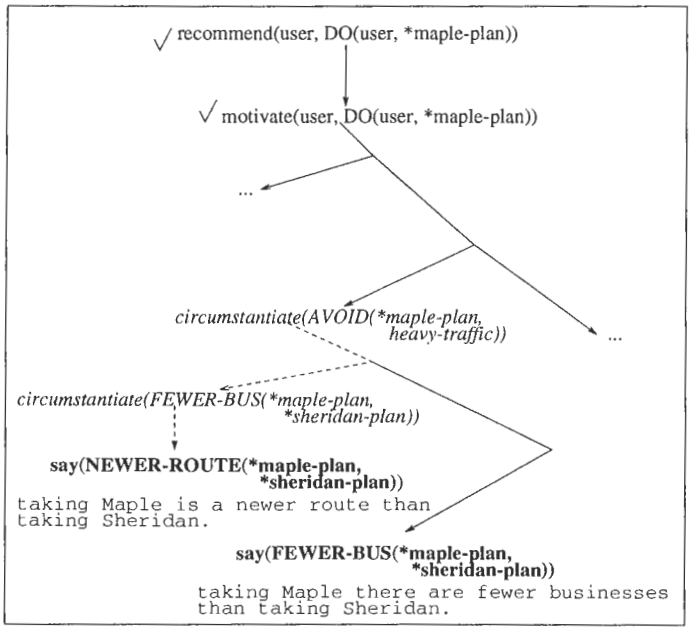


Figure 8: The System's Response

is as an expansion of the CTP *circumstantiate(fewer-businesses(maple-plan, sheridan-plan)*.

7 Related Work

Like the EES Text Planner [Moore and Swartout, 1991] IDP plans only as needed in reaction to feedback. However, the EES Text Planner does not build a single text plan [Moore and Swartout, 1991]. The system analyzes questions like *Why?* using a stack of text plans from previous exchanges and a development history provided by the expert reasoning system. The system applies heuristics to these sources of information to generate a set of likely interpretations and select one of them.

In the IDP model, the system's TP is treated as a richer resource of information. Therefore, IDP builds a single TP. In this respect, IDP is similar to the Explanatory Discourse Generator (EDGE) [Cawsey, 1990]. EDGE formulates and represents a single text plan, and uses it as the discourse context to analyze feedback. However, EDGE plans tutorials. Therefore, unlike IDP, EDGE formulates its text plan in advance and executes it incrementally.

8 The System Implementation

There is a one-way flow of control through three of IDP's e five components: the parser, the analyzer, and the planner-actor. During a complete user-system exchange, this flow starts with, and cycles back to, the parsing component. After selecting and structuring content, the planner-actor sends a text message to a fourth component, the generation grammar written in the Generalized Augmented Transition Network (GATN) formalism [Shapiro, 1982]. The same formalism has been used for the parser. Finally, these four components all have access to a fifth component, a knowledge base, which they consult to find or deduce information. [Haller, 1994] provides details.

In the knowledge base, the several sources of information that the system needs to analyze and plan the discourse are all represented uniformly using the Semantic Network Processing System (SNePS) [Shapiro and Rapaport, 1987; Shapiro, 1991]. This includes knowledge

of the text plan operators, the domain plans, entities in the domain, the user model, the discourse plan executed so far, and rules for reasoning about all of the above.

The planning-acting component is based on the SNePS Actor [Shapiro *et al.*, 1989]. Based on the TOUR model [Kuipers, 1978], the various driving routes that IDP can discuss are represented as preconstructed plans that are composed of two types of acts: *going* and *turning*. The domain plans are represented at various levels of detail and, as conceptual entities, can have properties. Whenever the system reasons about the domain, the reasoning that leads to deductions is recorded in the knowledge base along with the deductions themselves and is available as content for explanations.

9 Current Status and Future Work

IDP currently uses ten TP-operators to formulate TPs for justifying domain plan advice. I am extending the system to engage in domain plan descriptions and system-imposed topic shifts. Since my model represents the TP uniformly with the domain plans that are under discussion, another objective is to extend IDP so that it can engage in full-blown, meta-level discussions of its own TP.

References

[Allen and Perrault, 1980] J. Allen and C. R. Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15, 1980.

[Carberry, 1989] S. Carberry. A pragmatics-based approach to ellipsis resolution. *Computational Linguistics*, 15(4), 1989.

[Cawsey, 1990] A. Cawsey. Generating explanatory discouse. In R. Dale, C. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation*. Academic Press, 1990.

[Cohen and Perrault, 1979] P. R. Cohen and C. R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3, 1979.

[Grice, 1969] H. P. Grice. Utterer's meaning and intentions. *Philisophical Review*, 78, 1969.

[Grice, 1975] H. P. Grice. Logic and conversation. In P. Cole and J. L. Morgan, editors, *Syntax and Semantics 3: Speech Acts*. Academic Press, New York, 1975.

[Grosz and Sidner, 1986] B. J. Grosz and C. L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linquistics*, 12, 1986.

[Haller, 1994] S. M. Haller. A model for cooperative interactive plan explanation. In *Proceedings of the The Tenth IEEE Conference on Artificial Intelligence for Applications*, San Antonio, Texas, 1994.

[Kuipers, 1978] B. Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2, 1978.

[Mann and Thompson, 1987] W. C. Mann and S. A. Thompson. Rhetorical structure theory: A theory of text organization. Technical report, Information Sciences Institute, 1987.

[Martins and Shapiro, 1988] J. P. Martins and S. C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35(1), 1988.

[Moore and Pollack, 1992] J. D. Moore and M. E. Pollack. A problem for RST: The need for multi-level discourse analysis. *Computational Linguistics*, 18(4), 1992. discussion.

[Moore and Swartout, 1991] J. Moore and W. Swartout. A reactive approach to explanation: Taking the user's feedback into account. In C. Paris, W. Swartout, and W. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academic Publishers, 1991.

[Searle, 1969] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.

[Shapiro and Rapaport, 1987] S. C. Shapiro and W. J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*. Springer–Verlag, New York, 1987.

[Shapiro *et al.*, 1989] S. C. Shapiro, D. Kumar, and S. Ali. A propositional network approach to plans and plan recognition. In *Proceedings of the 1988 Workshop on Plan Recognition*, Los Altos, CA, 1989. Morgan Kaufmann.

[Shapiro, 1982] S. C. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *American Association of Computational Linguistics*, 8, 1982.

[Shapiro, 1991] S. C. Shapiro. Case studies of SNePS. *SIGART Bulletin*, 2(3), 1991.

[Sidner, 1985] C. Sidner. Plan parsing for intended response recognition in discourse. *Computational Intelligence*, 1, 1985.

From Text to Horn Clauses: Combining Linguistic Analysis and Machine Learning

Sylvain Delisle *, Ken Barker **, Jean-François Delannoy **, Stan Matwin **, Stan Szpakowicz **

* Département de mathématiques et d'informatique
Université du Québec à Trois-Rivières
Trois-Rivières, Québec, Canada G9A 5H7
Sylvain_Delisle@uqtr.uquebec.ca

** Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5
{kbarker, delannoy, stan, szpak}@csi.uottawa.ca

Abstract

The paper describes a system that extracts knowledge from technical English texts. Our basic assumption is that in technical texts syntax is a reliable indication of meaning. Consequently, semantic interpretation of the text starts from surface syntax. The linguistic component of the system uses a broad-coverage, domain-independent parser of English, as well as a user-assisted semantic interpreter that memorizes its experience. The resulting semantic structures are translated into Horn clauses, a representation suitable for Explanation-based Learning (EBL). An EBL engine performs symbol-level learning on representations of both the domain theory and the example provided by the linguistic part of the system. Our approach has been applied to the Canadian Individual Income Tax Guide and examples from it are used in the presentation.

1. Introduction

The goal of our project is knowledge extraction from texts. We are building a system that accumulates knowledge using the smallest possible domain-specific kernel prepared in advance. In the case of texts that we characterize as technical, our approach performs this extraction with *no* advanced, precoded knowledge, assuming the assistance of a cooperative user. The system applies natural language processing and machine learning techniques to process English technical text. The result is a Horn clause rule base representing knowledge about semantic relations among concepts presented by the text. We think that this can be done from scratch, provided the user trains the system in the initial phase of knowledge extraction. User intervention should then decrease with time as the system effectively learns from previous interactions.

This work is supported by a strategic grant from the Natural Sciences and Engineering Research Council of Canada. Thanks to Terry Copeck for commenting on a very late draft of the paper.

Because much knowledge is communicated via textbooks, manuals, handbooks etc., a system like ours will be an extremely useful alternative to "traditional" knowledge acquisition tools. In domains where an expository text is available, its user-assisted processing with our system will provide a first, perhaps unpolished version of the knowledge base. This version may then be debugged and improved by the user.

We note the main features of our approach, along with the ensuing conditions of its applicability.

- Detailed surface-syntactic analysis of a fragment of a technical text[1] precedes semi-automatic analysis of clause-level relations, Case relations and relations inside noun phrases (work in progress). The resulting semantic structure is transformed semi-automatically into Horn clauses. Our research hypothesis is that, in technical texts, syntax gives a reliable indication of meaning: literal interpretation based on surface syntax is usually appropriate [Kieras, 1985] and a high degree of compositionality is possible. A standard, linguistic-theory-neutral grammar of English based on Quirk *et al.* [1985] underlies the parser. Details of parsing and Case-based semantic analysis are presented in Delisle [1994].

- The system needs to be trained by the user. It remembers and generalizes the user's additions and changes to its semantic pattern dictionaries. Saturation with domain-dependent *linguistic* knowledge is gradually achieved as the user moves forward in the given text; the intervention is eventually reduced to simple approval. We are now working on the details of a measure of user interaction to be used as the yardstick of our system's performance. It will consider the amount of interaction (for example, the number of updates of the pattern dictionaries) and the ease

[1] It is widely accepted that technical texts are somehow easier to process, despite the absence of any commonly acceptable definition of what constitutes a technical text. We have assembled a checklist of linguistic properties that make a text technical; a working paper is forthcoming.

of interaction (for example, using Angluin's [1988] oracle types).

- We assume rich syntactic knowledge and detailed semantics of function words. We also rely on publicly available domain-independent lexical knowledge. The Collins dictionary [Karp *et al.*, 1992] is used by the parser for part-of-speech information. WordNet [Miller, 1990] will be used for disambiguation and for semantic clustering [Feng *et al.*, 1994]. No domain-specific knowledge need be assumed: the system can be run with its semantic pattern dictionaries initially empty. Early experiments confirm that Case analysis of a text segment starting with empty dictionaries can produce an acceptably high percentage of the system's hypotheses merely confirmed by the user [Delisle, 1994].

- The system acquires knowledge incrementally during linear processing of the input text. Although the system can return to previously processed fragments just as human readers do, essentially it leads the user forward and learns useful facts even after a single pass over a fragment. The system performs learning at the symbol level in order to acquire accurate and sufficient knowledge that will be used to represent the meaning of the text.

2. Organization of the System

The organization of the system is summarized in Figure 1 (ovals represent modules, light rectangles denote data passed between modules, heavy rectangles are permanent repositories). The MaLTe[2] system receives its linguistic data from the TANKA[3] system. DIPETT[4] is TANKA's noncommittal surface-syntactic parser (Jacobs and Rau [1993] briefly discuss the role of inexact parsing in text processing). A parse tree of the current sentence produced by DIPETT may be reorganized by the phrase reattachment module. The structurally correct parse tree is processed by HAIKU, a three-step interactive semantic analysis module. HAIKU suggests semantic relations among clauses in the sentence (for example, causality, enablement, precedence), then Case patterns in clauses and finally relations inside noun phrases; the user confirms or overrides those suggestions. Relations not encountered earlier are added to HAIKU's semantic dictionaries (not shown in Figure 1).

The result, a composite graph containing syntactic and semantic information about the sentence, is called a *protonetwork* ("early representation of the network"). Within TANKA, it is passed on to the Network Fragment Builder that turns it into a fragmentary conceptual network. This fragmentary network will then be merged with a growing conceptual network representation [Yang and Szpakowicz, 1991] of the part of the source text processed so far.

The protonetwork is simultaneously passed on to the first module of MaLTe, which translates it into a set of Horn clauses. Translations of the narrative part of a text and its examples are distinguished. The Machine Learning module, which includes an EBL[5] engine, organizes these Horn clauses into a domain theory.

There are two main tasks in this operation. First, the domain theory is accumulated and organized by a hierarchization of Horn clauses into a stratified rule set in which levels of rules are clearly delineated. This is achieved by transformations of sets of clauses (for example, absorption) used in Inductive Logic Programming to reorganize clause sets into logic programs. The second task, essential to the MaLTe approach, applies EBL to the clausal representation of the examples. The clausal representation of the explanatory text plays the role of domain theory. This gives a compiled, generalized and operational rendering of the examples which includes the knowledge necessary to explain them.

Horn clauses representing the results of EBL are turned into a (simplified) protonetwork and fed back into the Network Fragment Builder. When the domain theory is sufficiently rich, it may be transmitted to a performance task external to the TANKA/MaLTe system. One example of such a performance task is a rule-based program producing income tax returns. The skeletal rule base containing the knowledge part of this program would be acquired by TANKA/MaLTe directly from the Income Tax Guide.

The system is being implemented in Quintus Prolog on Sun SparcStations. The parser and the Case analyzer are fully implemented. Prototypes of the machine learning mechanisms, the clause-level relationship analyzer and the protonetwork to Horn clause translator are close to completion. The reattachment module, the Network Fragment Builder and the noun-modifier relationship analyzer have been designed.

3. Syntactic and Semantic Analysis

3.1. The Parser

The parser accepts most sentences found in a technical text. Such a broad-coverage parser ensures that acquisition of knowledge from text is reasonably complete. Without a rich semantic model, syntax is the only support for meaning. DIPETT [Delisle and Szpakowicz, 1991; Delisle, 1994] handles, fully or partially, about 90% of the sentences in sample *unedited* texts.

---

2 Machine Learning from Text

3 Text Analysis for Knowledge Acquisition

4 Domain-Independent Parser of English Technical Texts

5 Explanation-Based Learning

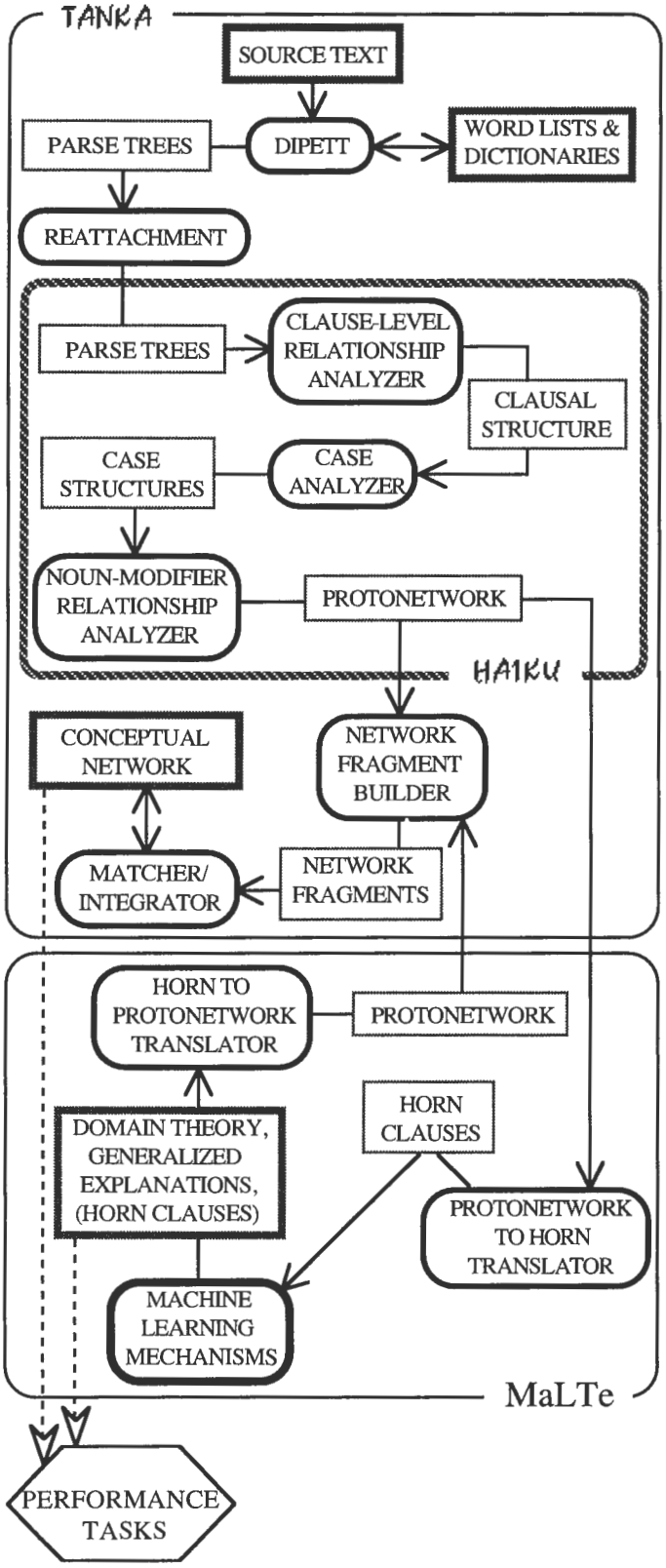


Figure 1. The TANKA and MaLTe Systems

Linguistic theories of syntax such as GPSG, HPSG and LFG use feature structures to encode linguistic objects: these structures' form and content depend on the underlying theory. DIPETT is not committed to any particular linguistic theory. Its DCG grammar formalism is theory-neutral (as are other formalisms, for example, PATR-II). Most of the grammar's rules are based on Quirk *et al.* [1985]. DIPETT may be considered as a functional grammar, that is, one in which syntactic analysis is based on syntactic roles instead of only word order.

In addition to standard parsing functions, the parser contains several subsidiary non-standard components: a simple tagger, a dynamic dictionary expansion facility, a memorizing device (that is, a well-formed substring table or passive chart) and an error explanation mechanism.

3.2. The Case Analyzer

We have defined a Case system which is used by HAIKU. Six Cases appear in this paper's examples: *Agent* (AGT), *Accompaniment* (ACMP), *Beneficiary* (BENF), *Location to* (LTO), *Object* (OBJ), *Time at* (TAT). The complete list of 28 Cases, as well as the motivation and the discussion of other published Case lists are presented in Barker *et al.* [1993].

The Case Analyzer (CA) takes a parse tree produced by DIPETT and semi-automatically extracts the Case pattern that best represents its meaning. Cases are signaled by Case markers, which are realized in two ways: *lexically*, for example, apreposition that introduces a prepositional phrase, and *positionally*, as subject (`psubj`), direct object (`pobj`), indirect object (`piobj`).

CA accumulates Case patterns in its semantic dictionaries and refers to them when processing new sentences. A sentence that has little in common with previously encountered patterns may introduce new elements of knowledge. These are integrated into the incrementally growing dictionaries. For a sentence similar to previously analyzed sentences, CA suggests a semantic interpretation for the user to confirm or reject. CA is instrumental to our intent of turning knowledge-based text understanding into knowledge acquisition. Knowledge acquired during this operation constitutes an important part of the conceptual model of a text's domain.

The main data structures are stored in the Meaning Dictionary, the Case-Marker Pattern Dictionary and the Case Pattern Dictionary. All dictionaries may be empty when the system is first used on a new text. CA does not need any seed knowledge to work properly, though it does require significant involvement of the user in the initial phase. The amount of work decreases as CA processes more sentences and acquires more patterns.

The following definitions describe CA terms used in the rest of this paper.

A **Case-Marker Pattern (CMP)** is an ordered list of Case markers, representing the markers appearing in a

clause. In CA, a clause has only one CMP, that is, only one syntactic analysis from which a unique CMP is derived.

A **Case Pattern (CP)** is an ordered list of Case abbreviations, representing the Cases appearing in a clause. In CA, a clause normally has only one CP, although it may have more if the clause is semantically ambiguous.

The **Meaning Dictionary** has entries for individual words. A verb entry contains a list of CMPs found with this verb in the text, and a list of Cases associated with each marker in these CMPs. The Meaning Dictionary also contains entries for prepositional and adverbial Case markers. Such an entry contains a fixed list of Cases the marker can realize; details are in Barker *et al.* [1993].

The **Case-Marker Pattern Dictionary** has entries for CMPs. An entry contains a list of CPs already associated with this CMP. Each CP is illustrated by an example sentence. This dictionary may be initialized with entries for a number of common CMPs. In CA, a CP may be associated with one or more examples since many syntactically different clauses can have the same Case pattern.

The **Case Pattern Dictionary** has CP entries, each containing a list of verbs associated with this CP in the text.

All three dictionaries are continuously updated. As mentioned above, a parsable clause should normally be associated with a unique CP. To accomplish this, CA first searches its dictionaries for the *target CP* that matches the CMP of the input sentence most closely, if not perfectly [Delisle *et al.*, 1993a]. Next, an example sentence to illustrate the target CP is fetched from the dictionary. If the CP and example sentence are not acceptable, the system asks for the user's help.

Note that order does not matter in patterns: it is only their semantic interpretation that counts. Thus, `subj-obj-at-by` is equivalent to `subj-obj-by-at`, and, similarly, AGT-OBJ-LAT-TAT is equivalent to AGT-OBJ-TAT-LAT.

Sentences from the following paragraph will be used as examples to illustrate CA as well as other processes described in the rest of this paper.

> "Jim is a member of the Canadian Armed Forces and was posted to Lahr in 1989. Jim's wife moved with him to Lahr. He broke all residential ties with Canada. Jim is a resident of Canada because he is serving abroad in the armed forces."

The fourth sentence above ("Jim is a resident of Canada because he is serving abroad in the armed forces.") contains two clauses, each of which will be analyzed separately by CA. The main verb of the first clause is the stative verb "be". Although stative verbs introduce facts about objects, activities and their properties, they do not have Cases. Consequently, clauses with stative main verbs are not treated by CA but are passed on for semantic processing by subsequent modules—for a description of the treatment of stative verbs, see Delisle *et al.* [1993b]. The second clause is Case Analyzed. The CMP `psubj-adv-in` is associated with the clause's main verb "serve". CA first checks the Meaning Dictionary to determine if the verb "serve" has appeared previously in the text and what CMPs and CPs have been associated with it. It also checks the CMP Dictionary to see if the CMP `psubj-adv-in` has occurred previously and what CPs have been associated with it. Based on this historical data as well as information about which Cases the individual Case Markers mark, CA suggests a CP to the user. The user may accept the suggested CP or reject it and supply a new CP for this clause. The three dictionaries are then updated to reflect this Case assignment. The output for the sentence (after successful Case assignment) would consist of the following Case structures:

```
case_structure(*statement1*, be, psubj-pobj-of,
 nil, 'Jim', resident, 'Canada')
case_structure(*statement2*, serve, psubj-adv-in,
 agt-lat-benf, 'Jim', abroad, 'the armed forces')
```

3.3. The Clause-Level Relationship Analyzer

Case Analysis deals with semantic interpretation of the relationships between a verb and its arguments within a clause. Semantic information is also conveyed by relationships between clauses. In particular, the causal links which are vital to the construction of rules from text are commonly found at the inter-clausal level. We are completing an extension of the semantic analyzer onto Clause-Level Relationships (CLRs). The design of the CLR Analyzer (CLRA) closely mirrors that of the Case Analyzer. First, a list of semantic relationships was constructed based on an exhaustive study of the lexical items signaling them. This set was then checked for completeness against a number of works in traditional and computational linguistics. The current list of CLRs is: *Causation*, *Enablement*, *Entailment*, *Prevention*, *Detraction*, *Conjunction*, *Disjunction*, *Location*, *Temporal Precedence*, *Temporal Co-occurrence*—details have been presented in Delisle *et al.* [1993b].

During interactive semantic analysis, CLRA is activated when the current input sentence contains syntactically connected clauses. The connective (a conjunction) is matched against a list of potential CLR markers and the CLRs they typically mark. One or more CLRs from this list are suggested to the user who is given the option of accepting a suggested CLR or assigning a new one.

Consider again the sentence "Jim is a resident of Canada because he is serving abroad in the armed forces". CLRA recognizes two clauses in the input connected by the conjunction "because". It finds in its dictionary of CLR markers that "because" often marks Causation, Enablement and Entailment. The user can choose one of these or enter a new CLR. If Entailment is chosen, the analysis will be stored as:

> "Jim is serving abroad in the armed forces"
> <entails>
> "Jim is a resident of Canada"

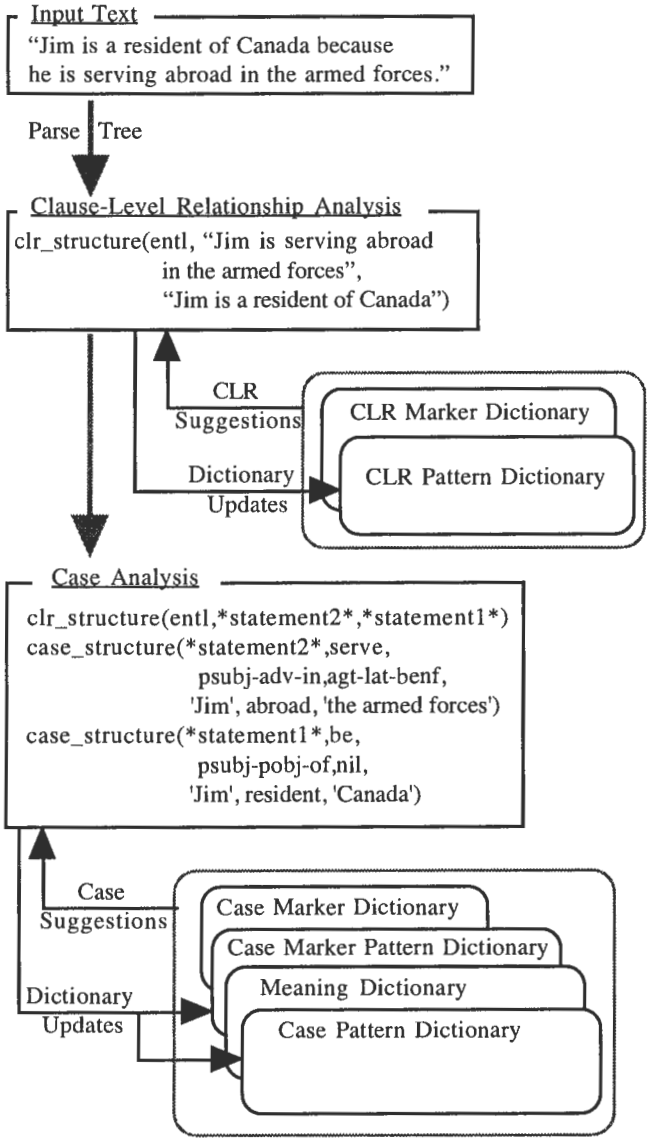


Figure 2. A Condensed Illustration of HAIKU Semantic Processing

The whole process of semantic analysis is summarized in Figure 2.

We have recently incorporated additional linguistic clues in the semantic analysis of CLRs. The syntactic makeup of clauses themselves helps identify the semantic relationships between them. For example, the modality of their main verbs is useful in distinguishing between different causal relationships, as the following example sentences suggest: "If you claim expenses then you *must* have paid money for services" (*Entailment*); "If you paid money for services then you *may* claim expenses" (*Enablement*).

4. The Protonetwork-to-Horn Processor

4.1. Construction of Horn Clauses

In contrast to several other approaches (see section 5), the conversion chain followed in MaLTe involves deriving a linguistically justified semantic encoding, which is then translated into first-order logic. For a moderate additional computational cost, the resulting principled process is also general and portable. Generality relies on the separation of processing tasks from user interaction. The source of domain knowledge is the user, who makes domain-dependent choices in disambiguation and Case or CLR assignment, while the processes of parsing and semantic processing are inherently domain-independent.

Translation from semantics to logic is done after all phrase reattachment and pronoun resolution have been completed. The input is the protonetwork material, which includes the CLR structure of a sentence, the detailed Case structure of its clauses and the internal description of the semantics of noun phrases.

The aspect of a clause (stative versus nonstative) is a key factor. Stative clauses in a technical context are taken to be definitional (for example, "An eligible child could be your child, your spouse's child, etc."), or they can describe an attribute of an instance ("Jim is a member of the Canadian armed forces."). HAIKU assigns no Cases to stative clauses. We translate them with a predicate based on the attribute. Nonstative clauses ("Jim moved to Lahr") are assigned Cases whose labels are attached to the verb to form a specific predicate name (such as `serve_agt_lat_benf`).

The algorithm follows a declarative description of the tree structure produced by HAIKU. The structural organization corresponds roughly to the input grammar of DIPETT, except that the structure to analyze is not a token sequence, but Prolog terms. These terms correspond to the sentence and its clauses; their nodes should be seen as mere functors, not predicates, since they can have variable "arity" according to the presence or absence of optional syntactic constituents.

At sentence level, the clause-level relationship is the most important criterion to determine how to piece together the elements of the Horn clause: with *Causation*, *Enablement* and *Entailment* (the latter especially important in a tax guide), the module asserts a rule. When clauses are simply conjoined, it asserts two independent facts. The user is asked for confirmation in all situations, which include more ambiguous relations like temporal precedence (which may or may not denote causality).

Thus, the fourth sentence:

> "Jim is a resident of Canada because he is serving abroad in the armed forces"

which is represented by :

```
clr_structure(entl, *statement2*, *statement1*)
case_structure(*statement1*, be, psubj-pobj-of,
 nil, 'Jim', resident, 'Canada')
case_structure(*statement2*, serve, psubj-adv-in,
 agt-lat-benf, 'Jim', abroad, 'the armed forces')
```

will be transformed into:

```
is_resident_of(jim, canada) :-
   serve_agt_lat_benf(jim, abroad, armed_forces).
```

If a clause has a negative polarity ("X is not eligible...") then we have to assert a rule involving explicit negation. The most frequent situation, and the simplest, is to simply assert a fact because there is no entailment or opposition relationship between the clause and other clauses, as in the other sentences of the example:

```
is_member(jim, canadian_armed_forces).
post_obj_lto_tat(jim, lahr, 1989).
move_agt_acmp_lto(wife, jim, lahr).
break_agt_obj_benf(jim, residential_ties, canada).
```

Note that some inference or interaction may be needed to relate different encoding of the same concepts, as canadian_armed_forces and armed_forces in the example.

4.2. Learning

The logical translation of both the narrative and the example sections of a text is fed to the learning module which performs Explanation-based Learning (EBL).

EBL is a learning method that generalizes a concept or procedure description from a single example. Rather than discriminating and generalizing from features of a large number of examples, as in the standard inductive approach in the spirit of ID3 [Quinlan, 1986], EBL uses an explanation of just one training instance as the basic learning tool. Explanation is usually a deduction that justifies (for example, through a Prolog-style proof) the statement "this specific instance is an instance of the concept we are learning". The explanation is used for two purposes. Firstly, it identifies the relevant features of the example, which are sufficient conditions for describing the concept [Minton *et al.*, 1990]. Secondly, generalization in EBL is performed by regressing the concept definition through the explanatory structure (for example, an AND tree). Consequently, the generalization process often turns constants of the example into terms, rather than just variables [Mitchell *et al.*, 1986]. Those terms bring into the explanation certain relevant parts of domain knowledge. In order to produce an explanation, an EBL learner must have a domain theory. If the theory is represented in the Horn clause format, it can be easily used to produce an explanation of the example. The concept can be treated as a top-level Prolog goal, the example—as a conjunction of Prolog facts, and the domain theory acts as a Prolog program. If the goal, properly instantiated with the constants of the example, can be proven by a Prolog-like interpreter, EBL succeeds and the goal tree is treated as an explanation. In the system described in this paper, the narrative text is converted—in several steps—into a domain theory, and the examples in the text are used as the training instances.

As described elsewhere [Delannoy *et al.*, 1993], we rely on transformations such as abstraction and absorption from Inductive Logic Programming to organize clauses into a meaningful, hierarchical knowledge base. The EBL process takes as input the Horn clause base produced by the Protonetwork-to-Horn (PtH) module, as well as the clausal representation of the examples from the text, and performs EBL on them. We shall illustrate this process below. Suppose that the domain theory (acquired in a manner described above) contains the following rules:

```
claim_child_care_expenses(P, C, E) :-
   person_deduct_expenses(P),
   eligible_child(C),
   deduct_amount_expenses(E).
person_deduct_expenses(P) :-
   is_resident_of(P, canada), eligible(P).
...
```

Let further facts from the example be produced by the PtH module as shown above. The EBL process continues, producing first the proof tree as in Figure 3 (only a fragment is shown).

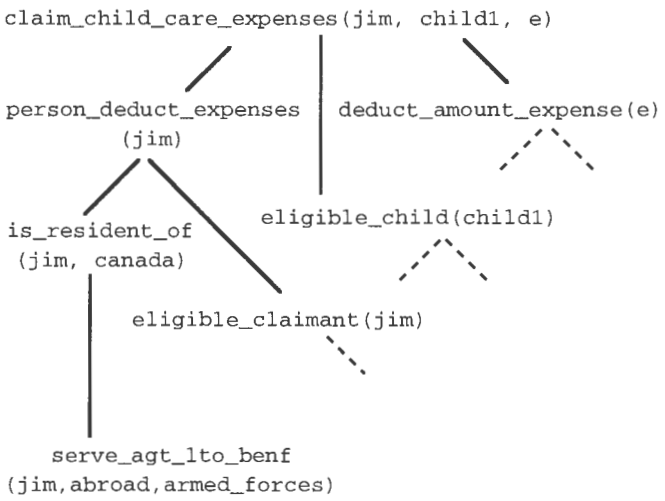


Figure 3. Fragment of the Proof Tree Produced by the EBL Process

EBL extracts from the whole collection of facts available in a given example exactly those that are necessary to prove that the example satisfies the concept definition (here, that it is an instance of the concept `claim_child_care_expenses`). EBL also puts together (compiles) all the knowledge necessary to show the membership of the example in the concept. Moreover, there is generalization in the second phase of EBL that consists in regression of the domain theory rules through the proof tree. Generalization will produce a useful ("operational", in the EBL terminology) generalization, that is:

```
claim_child_care_expenses(P, C, E) :-
   serve_agt_lat_benf(P, abroad, armed_forces),
   ...
```

Such a rule is then added to the domain theory. Consistent with the EBL paradigm, symbol-level learning has been achieved, and it has made the theory more useful than its general rendering (it applies now *directly* to all armed forces personnel stationed overseas), and at the same time more general than the specific example provided in the text. Unlike in simple inductive systems, the generalization obtained is fully justified by the existing domain theory.

5. Related Work

Work on classification tasks, useful in knowledge base (KB) construction, uses text processing as a means of extracting classification knowledge from various textual material. Silvestro [1988] and Gomez [1989] are two examples of that approach. Moulin and Rousseau [1992] describe a system which looks for predetermined, fixed patterns (for example, 'if', 'because', 'when') in the input sentences and decompose the original sentences into representations that stand for production rules or KB elements. Ciravegna *et al.* [1992] present the SINTESI system. It extracts knowledge from short (4 or 5 sentences) descriptive diagnostic reports written in Italian, in order to summarize their technical content and support the constructing of a KB on car faults. All objects that may be of interest in a text are described in a KB that is available *a priori* and there is no incremental KB augmentation.

Kim and Moldovan [1993] describe PALKA, a semi-automatic KA system designed to facilitate the construction of a large KB of semantic patterns accumulated from corpora. PALKA requires much *a priori* knowledge: a general and domain-specific concept hierarchy and frame definitions telling the system what to look for in a text (along with the relevant keywords). Liu and Soo [1993] have implemented a system that attempts to assign thematic roles to sentence elements using minimal *a priori* knowledge. The system uses syntactic clues to propose an initial set of potential thematic roles. This set is then pruned by applying heuristics and consulting the user.

Given the knowledge-intensive character of Natural Language Processing, surprisingly little work has been done in applying machine learning techniques in the context of NLP systems. Hauptmann [1993] describes how a relatively unsophisticated, rote-learning mechanism helps in the acquisition of a mapping from syntax to the meaning of sentences in a given domain. Zelle and Mooney [1993] and Aliprandi [1993] show how different learning techniques, inductive logic programming and standard induction, apply in the resolution of the propositional phrase attachment problem. A separate research community [Powers, 1989] focuses on the difficult questions of applying machine learning in the attempt to understand the cognitive aspects of language learning. Cohen [1990] shows how crucially learning from texts relies on a flexible definition of operationality. His work concentrates on the learning aspects without attempting to develop an integrated NLP-ML system.

6. Conclusion

We propose a combination of partially automated text processing and explanation-based learning for knowledge extraction from unedited technical texts. Early experiments show that this alliance yields more knowledge than standard language processing methods alone; at the same time, novel learning problems and opportunities originate from the fact that the domain theory is semi-automatically constructed directly from the text. Such mutual enrichment of natural language processing and machine learning lies in a largely uncharted territory. Challenging questions arise from the design of the first version of the system. The transformation of text fragments into a domain theory (expressed in first order logic) adequate for explanation-based learning requires intensive user participation, if the knowledge acquisition exercise is to be meaningful. However, learning (inductive generalization of user's interventions) is expected to decrease the amount of user interaction. The characterization of this amount as a function of the properties of the text and of the gradual saturation of the knowledge base through experience is an open research problem. Reliance on surface syntax as the carrier of meaning is vindicated by the ability of the linguistic subsystem to work up from an empty domain-specific knowledge base. Research problems in learning include explanation-based learning in an unavoidably incomplete domain theory, dynamic modification of the operationality criterion according to the changing performance task, and extension of learning from a first order logic rule base onto sorted logic.

References

[Aliprandi and Saviozzi, 1993] G. Aliprandi and G. Saviozzi, "A Supervised Learning Method to Solve PP-Attachment Ambiguities in Natural Language", *Proceedings Machine Learning and Text Analysis Workshop, ECML-93*, Vienna 1993, 45-52.

[Angluin, 1988] D. Angluin, "Queries and Concept Learning", *Machine Learning*, **2**(4): 319-342, 1988.

[Barker *et al.*, 1993] K. Barker, T. Copeck, S. Delisle and S. Szpakowicz, "An Empirically Grounded Case System", submitted to the *International Journal of Lexicography*, 36 pages.

[Ciravegna *et al.*, 1992] F. Ciravegna, P. Campia and A. Colognese, "Knowledge Extraction from Texts by SINTESI", *Proceedings 15th International Conference on Computational Linguistics—COLING-92*, Nantes 1992, 1244-1248.

[Cohen, 1990] W. W. Cohen, "Learning from Textbook Knowledge: A Case Study", *Proceedings AAAI-90*, 743-748, 1990.

[Delannoy *et al.*, 1993] J.-F. Delannoy, C. Feng, S. Matwin and S. Szpakowicz, "Knowledge Extraction from Text: Machine Learning for Text-to-Rule Translation", *Proceedings Machine Learning and Text Analysis Workshop, ECML-93,* Vienna 1993, 1-7.

[Delisle and Szpakowicz, 1991] S. Delisle and S. Szpakowicz, "A Broad-Coverage Parser for Knowledge Acquisition from Technical Texts", *Proceedings 5th International Conference on Symbolic and Logical Computing — ICEBOL5* (Madison, S.D., USA), April 1991, 169-183.

[Delisle, 1994] S. Delisle, "Text Processing without A-Priori Domain Knowledge: Semi-Automatic Linguistic Analysis for Incremental Knowledge Acquisition", Ph.D. thesis, Department of Computer Science—Ottawa-Carleton Institute for Computer Science, TR-94-02, University of Ottawa, 1994.

[Delisle *et al.*, 1993a] S. Delisle, T. Copeck, S. Szpakowicz and K. Barker, "Pattern Matching for Case Analysis: A Computational Definition of Closeness". O. Abou-Rabia, C. K. Chang and W. W. Koczkodaj (eds.) *Proceedings ICCI-93*, 310-315.

[Delisle *et al.*, 1993b] S. Delisle, K. Barker, T. Copeck and S. Szpakowicz, "Interactive Semantic Analysis of Technical Texts: Case Pattern Acquisition", submitted to *Computational Intelligence*, 67 pages, 1993.

[Feng *et al.*, 1994] C.Feng, T. Copeck, S. Szpakowicz and S. Matwin, "Semantic Clustering. Acquisition of Partial Ontologies from Public Domain Lexical Sources". *Proceedings AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff 1994, 1-15.

[Gomez, 1989] F. Gomez, "Knowledge Acquisition from Natural Language for Expert Systems Based on Classification Problem-Solving Methods", *Proceedings 4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff 1989, 15.1-15.18.

[Hauptmann, 1993] A. G. Hauptmann, "Meaning from Structure in Natural Language Interfaces", Ph.D. Thesis, Computer Science Department, Carnegie-Mellon University, 1993.

[Jacobs and Rau, 1993] P. S. Jacobs and L. F. Rau, "Innovations in Text Interpretation", *AI Journal* 63(1-2) (Special Issue on Natural Language Processing), October 1993, 143-191.

[Karp *et al.*, 1992] D. Karp, Y. Schabes, M. Zaidel and D. Egedi, "A Freely Available Wide Coverage Morphological Analyzer for English", *Proceedings 15th International Conference on Computational Linguistics—COLING-92*, Nantes 1992, 950-955.

[Kieras, 1985] D. E. Kieras, "Thematic Processes in the Comprehension of Technical Prose", in B. K. Britton and J. B. Black (eds.), *Understanding Expository Text (A Theoretical and Practical Handbook for Analyzing Explanatory Text)*, LEA, 89-105, 1985.

[Kim and Moldovan, 1993] J.-T. Kim and D. I. Moldovan, "Acquisition of Semantic Patterns for Information Extraction from Corpora", *Proceedings 9th IEEE Conference on AI Applications*, 171-176, 1993.

[Liu and Soo, 1993] R.-L. Liu and V.-W. Soo, "An Empirical Study on Thematic Knowledge Acquisition based on Syntactic Clues and Heuristics", *Proceedings 31st Annual Meeting of the ACL*, Columbus, Ohio, 1993, 243-250.

[Miller, 1990] G. A. Miller, (ed.), "WordNet: An On-Line Lexical Database", *International Journal of Lexicography*, **3**(4), 1990.

[Minton et al., 1990] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni and Y. Gil, "Explanation-Based Learning: A Problem Solving Perspective", Machine Learning (Paradigms and Methods), J. Carbonell (ed.), MIT Press, 63-118, 1990.

[Mitchell et al., 1986] T. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-based Generalization: A Unifying View", *Machine Learning*, **1**(1): 47-80, 1986.

[Moulin and Rousseau, 1992] B. Moulin and D. Rousseau, "Automated Knowledge Acquisition from Regulatory Texts", *IEEE Expert*, October 1992, 27-35.

[Powers and Turk, 1989] D. M. Powers and C. R. Turk, *Machine Learning of Natural Language*, Springer-Verlag, 1989.

[Quinlan, 1986] J. R. Quinlan, "Induction of decision trees", *Machine Learning* 1: 81-106, 1986.

[Quirk *et al.*, 1985] R. Quirk, S. Greenbaum, G. Leech and J. Svartvik, *A Comprehensive Grammar of the English Language*, Longman, 1985.

[Silvestro, 1988] K. Silvestro, "Using Explanations for Knowledge-Based Acquisition", *International Journal of Man-Machine Studies*, **29**: 159-169, 1988.

[Yang and Szpakowicz, 1991] L. Yang L. and S. Szpakowicz, "Inheritance in Conceptual Networks". Karen S. Harber (ed.) *Proceedings Sixth International Symposium on Methodologies for Intelligent Systems (Poster Session)*. Charlotte, NC, 1991, 191-202.

[Zelle and Mooney, 1993] J. M. Zelle and R. Mooney, "ILP Techniques for Learning Semantic Grammars", *Proceedings ILP Workshop, IJCAI-93*, Chambery (France), 83-92, 1993.

A REASONED INTERLINGUA FOR KNOWLEDGE-BASED MACHINE TRANSLATION

John R. R. Leavitt, Deryle W. Lonsdale, and Alexander M. Franz
Center for Machine Translation, Carnegie Mellon University,
Pittsburgh, Pa., USA, 15213.
jrrl@cs.cmu.edu, lonz@cs.cmu.edu, amf@cs.cmu.edu

Abstract

Research in machine translation (MT) has resulted in a number of Machine Translation systems that are based on the interlingua approach. This paper reports the results of designing an interlingua for a large-scale, practical MT system designed to translate technical information from English into a number of target languages. After an analysis of the main features of the translation problem faced by this system we describe the principles underlying our design decisions. We address issues such as the design and development methodology, the grain size of the representation, and our efforts to endow the interlingua representation with the ability to degrade gracefully. We conclude that large-scale MT of technical documentation can be achieved with an interlingua-based architecture if the pivotal point of the system, the interlingua, has been designed in a systems-oriented manner.

Keywords: Machine Translation, Applications, Knowledge Representation

1 Introduction

An interlingual architecture for machine translation (MT) has a number of advantages over other possible architectures, such as the "transfer" model. In an interlingua-based architecture, source text analysis and target text generation are divided into two separate components. An intermediate knowledge representation level, called the *interlingua*, mediates between the analysis and generation components. That is, the analysis component creates interlingua representations for the source text, and the generation component starts from the interlingua representation and creates target text from it. This allows the various knowledge sources (including knowledge sources for different languages) to be developed in parallel, and it creates an independence between the different components that greatly supports the development of more than one language pair.

On the other hand, an interlingua-based architecture creates a number of new difficulties, because the interlingua becomes the central pivot point for the entire translation process. This paper describes the design and implementation of the interlingua for the KANT project [Nyberg and Mitamura, 1992], a large-scale Machine Translation project for translating technical texts into multiple target languages.

2 Problem Definition

This section describes the problem of high-quality automatic translation of technical information.

2.1 Knowledge Sources

Machine translation requires the use of various types of knowledge (hence the term "Knowledge-Based Machine Translation" [Nirenburg et al., 1992]). For each language, this includes spelling, contraction, and formatting rules; morphological rules; lexical knowledge, including syntactic features, semantic concepts, and collocational and terminological information; knowledge about the grammatical structure; and semantic rules. In addition, a certain amount of knowledge about the domain is required.

2.2 Multiple Targets

The KANT project deals with machine translation as a tool for global information dissemination. For example, one KANT application under development translates technical service information for Caterpillar, Inc. products (heavy machinery) from English into the languages of the major export markets. Another example of a possible domain is user information for consumer electronics, such as television sets. Since there is only one source language, the analysis component can lean towards slight language dependence, but it is necessary to handle generation in multiple languages.

2.3 Technical Sublanguage Translation

KANT is a *sublanguage* translation system. That is, it is not designed to translate all of the English language, but rather a well-defined subset. An application sublanguage is constrained both by the domain from which the source texts are drawn (e.g. service information for heavy machinery), and by general restrictions that form a "Constrained Technical English." Since these restrictions define lexical, syntactic, semantic, and conceptual inventory that is in a fundamental sense closed (while of course remaining open for extensions within the framework), it is possible to achieve complete coverage for the source sublanguage during system development.

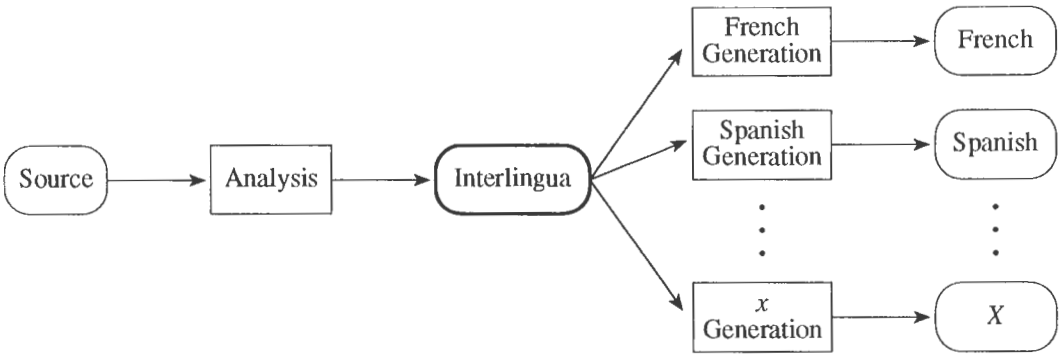


Figure 1: The Interlingual MT Architecture

Furthermore, Standard Generalized Markup Language (SGML) text markup codes are used in the input. Each application uses a tailored Document Type Definition (DTD) that includes tags for the logical and semantic structure of the domain. These tags are used directly during analysis, help to structure the source text, and are explicitly represented in the interlingua.

Compared to literary prose or poetry, technical information is conceptually, semantically, and pragmatically rather uncomplicated. This allows for another important feature: KANT translates on a sentence-per-sentence basis, and does not attempt to compute complicated pragmatic and discourse meanings. This means that the information that has to be represented in the interlingua is restricted to a feasible amount.

2.4 Modular Design

Multilingual MT is a complex problem. The different knowledge sources for the various languages need to be developed by separate language experts, and domain knowledge has to be encoded into the domain model by an expert in the domain. This calls for a modular architecture that separates knowledge sources from processing engines, shields different languages from each other to avoid language-pair-specific development, and that provides module interfaces that are habitable for language experts who are not necessarily skilled programmers.

2.5 The Interlingua Approach

The solution to these challenges is to divide the problem into source language analysis, and target language generation (see Figure 1). The interface between these two components is an intermediate language called the *interlingua*. It is a language-independent, unambiguous representation of the meaning of the input text that has to fulfill a simple functional condition: the interlingua representation must be sufficient for accurate translation in a technical domain.

3 Design Principles

This section describes the principles underlying the design of the KANT interlingua. Section 4 shows how these principles were put into practice.

3.1 Reasoned Approach

Some MT projects adopt a certain theory or methodology at the onset and adhere to it throughout development. In our approach, the first step is to analyze the problem, and then to "reason out" a design that results in a practical, working system.

3.2 Staged Approach

Since the interlingua plays a central role in the system, we chose to develop it by a method of iterative refinement. This allows development of the separate components to proceed smoothly in parallel not only with each other, but with the development of the interlingua itself.

3.3 Balanced Approach

[Tsujii, 1988] discusses three general ways to approach the task of choosing and implementing an interlingua:

- top-down: by considering the domain and enumerating *a priori* the concepts, processes, and relationships required for its treatment;
- bottom-up: by considering the (disambiguated) lexical content expressed by text discussing the domain, and then defining sets, hierarchies, and other relevant relationships; and
- decompositional: by (re-)expressing all relevant aspects of the domain with respect to a highly restricted set of semantic primitives.

We believe that the best design method is a combination of the top-down and bottom-up approaches. Previous work, and linguistic theory, guides an initial top-down structuring of the domain of the interlingua. Then, corpus-based incremental bottom-up work extends the interlingua towards complete coverage of the domain.

3.4 Comprehensive Approach

We believe that the interlingua must represent information from all necessary levels of linguistic analysis: Lexical, syntactic, and pragmatic. The interlingua is designed to represent all such necessary features, but no more.

4 The KANT Interlingua

The KANT interlingua is a recursive representation scheme using nested frames to correspond to information contents of elements of source sentences. Each interlingua frame contains a conceptual head and a series of feature-value pairs and semantic slots which in turn contain additional interlingua frames. Concepts can correspond to source language expressions (e.g. the action `*a-bond`), semantic units from the domain (e.g. `*c-decimal-number`), or structural elements such as tagged SGML constituents. A sample interlingua structure from the domain of heavy equipment manufacturing is shown in Figure 2.

Space prohibits a complete explanation of all the parts of the KANT interlingua here, but it is described extensively in [Leavitt et al., 1993]. This document includes not only detailed discussions of the semantic roles, concepts, and features used in interlingua, but also extensive examples of their usage, which also serve as a test suite for the implementation.

"The gasket must be bonded to the valve cover between the bolt holes (14 places) with <product> permabond 102 <\product>." $\Longrightarrow$

```
(*a-bond
   (punctuation period)
   (mood declarative)
   (obligation medium)
   (tense present)
   (topic-role theme)
   (theme (*o-gasket
             (number singular)
             (reference definite)))
   (attach_to (*o-valve-cover
                 (number singular)
                 (reference definite)))
   (located_between
      (*o-bolt-hole
         (number plural)
         (reference definite)
         (parenthetical
            (*o-place
               (number plural)
               (generic +)
               (quantity
                  (*c-decimal-number
                     (integer "14")
                     (number-type cardinal)
                     (number-form numeric)))))))
   (means_with (*s-product
                  (value "Permabond 102"))))
```

Figure 2: Example KANT Interlingua Structure

4.1 Conceptual Grain Size

Past MT efforts have addressed the problem of specificity in interlingual representations in several different ways. Many involve the creation of a highly structured language-neutral representation which will not be subject to the nuances, ambiguities, and other vagaries of natural linguistic usage. In addressing the granularity issue for the representation, we chose to combine the best features from different possible design paradigms.

With respect to the choice of primitives, we proceeded in both bottom-up and top-down fashions. On the one hand, we established a circumscription of the domain addressed, discourse styles, and typical document structure. At the same time, though, we carried out extensive bottom-up identification of the domain through extraction of knowledge by automated corpus analysis techniques.

From a linguistic perspective, we also proceeded in both directions. Whereas others have sought to incorporate into an interlingua framework the expressive machinery of formalisms from logic, programming languages, or descriptive linguistic theories, direct interpretation cannot always guarantee an appropriate granularity for a unified and comprehensive treatment of linguistic phenomena. We have chosen to avoid the small grain size of interlingual approaches like UNITRAN, which is based on a theory of lexical semantic description. While useful in other contexts, and in fact are not incompatible with a KBMT approach, it seemed overly specific for our needs.

To illustrate this overspecificity, we include examples from two other interlinguas. Figure 3 shows some excerpts from the TAMERLAN representation for an advertising text for a doughnut store[Nirenburg and Defrise, 1994]. The representation covers more than individual sentences, and it is structured as a sequence of frames that are related to each other via pointers. As can be seen from the example, TAMERLAN represents a far greater level of detail, and includes pragmatic and discourse information.

Dorr [1993] uses a different, but still overly rich, type of interlingua. An example of this structure is shown in Figure 4.

We have found that rather that attempting to attain the most diminutive grain size, it is quite useful to perform lexical chunking, combining such highly lexicalized items as fixed phrases, technical nomenclature, and company-specific terminology.

Our grain-size was also set based on an examination of the communication content of the documents we address. Realizing that the complexity of natural language phenomena tends to favor the creation of complicated interlingua structures, we sought to avoid too complex a set of relations, features, and concepts for our constrained domain. For example, some systems seek to establish a comprehensive modeling from the perspective of an intelligent agent interacting with a complex environment, and following goal-directed behavior involving interpretation of discourse and situational contexts (cf. [Defrise, 1993]). Certain types of text may well require such considerations; for ours, they tend to introduce avoidable overhead and complexity.

Clearly our goal was to design a minimalist representation, considering both the breadth and depth of the domain addressed, and avoiding the opposing pitfalls of over-complexity and under-specificity.

"Drop by your old favorite Dunkin' Donuts shop . . ." $\Longrightarrow$

```
(make_frame text_1
  (clauses
    (value clause_1 clause_2 clause_3 ...))
  (relations
    (value relation_1 relation_2 ...))
  (attitudes
    (value attitude_1 attitude_2 ...))
  (producer-intentions
    (value producer-intention_1)))

(make_frame clause_1
  (head (value %visit_1))
  (aspect
    (duration prolonged)
    (phase begin)
    (iteration 1))
  (time (value time_2)))

(make_frame %visit_1
  (is-token-of (value *visit))
  (agent (value *consumer*))
  (destination (value %shop_1)))

                    ⋮
```

Figure 3: Example Tamerlan Interlingua Structure

4.2 Graceful degradation

In that the KANT interlingua is a simple recursive data structure, it contains very little interrelation between elements. Only the interrelations that are necessary for generating output are represented. Furthermore, it is possible for the generation component to fail to realize various portions of the interlingua and still produce acceptable (if incomplete) output. For example, consider the interlingua structure in Figure 2. During generation, if the contents of the `means_with` or `parenthetical` slot were not realized, the resulting sentence would still convey the main idea of the sentence.

Similarly, if an incorrect value is assigned to a feature, an incorrect, but still comprehesible sentence can usually be produced. For example, during the early development of the system, the values of the `number` feature for objects were changed from `SG` and `PL` to `Singular` and `Plural`. For a while after that decision was made, an occasional `(number sg)` or `(number pl)` would appear. In these cases, the generator would usually produce a sentence in which the only error was the number on the noun phrase produced from the object and perhaps the subject-verb agreement if the affected object was mapped to the subject. Otherwise, the sentences were just fine and were certainly comprehensible.

Of course, there are places where a missed feature or semantic role would cause less graceful degradation. For example, failing to realize the contents of the `attach_to` or `theme` slot in the example interlingua structure would produce unacceptable output. There are, in essence, two classes of information in the interlingua — that which is necessary to create acceptable output and that which is not. If the latter is missing, the KANT interlingua may still maintain integrity, while an error or omission within the first class will cause a legitimate failure. Our interlingua differs from others in that we have tried to shift as much information as possible from the former class into the latter, by minimizing interactions between elements.

"I stabbed John." $\Longrightarrow$

[EVENT CAUSE ([THING I],
[EVENT GO_{POSS} ([THING KNIFE-WOUND],
[PATH $TOWARD_{POSS}$
([POSITION AT_{POSS}
([THING KNIFE-WOUND], [THING JOHN])])])])]

Figure 4: Example Dorr [1993] Interlingua Structure

In addition, since the structures are simple, generalized rules can be written for the generation components to handle most constructions, which further minimizes the chance of poor output. If the information for a given sentence were represented as a collection of objects that are connected only by a number of highly interreactive links, this would be more difficult.

Similarly, on the analysis side, if a semantic role cannot be determined for a given piece of information, a generic role may be used instead (i.e. `generic_with` instead of `goal_with`). Generalized rules for mapping these slots can be included on the generation side, which results in a further decrease in loss of information and quality.

4.3 Specification

Throughout the design, a specification document was maintained, to which both analysis and generation modules of the system could refer for the latest interlingua design.

After initial consideration of such issues as lexical precision, meaning preservation, syntactic markedness, metatextual reference, and semantic hierarchies, we established a set of relevant data types and representations. These data structure had to be habitable and mnemonic, since developers can find working with an interlingua difficult because of its high degree of abstraction away from the lexical form of language. Development of large-scale interlingua systems such as ours must not be further complicated by an opaque or obscure data format.

These decisions formed the basis of the specification document and test suite, both of which were iteratively refined.

4.4 Iterative Refinement

One difficulty with interlingual system is what has been the "horizon effect" seen in other endeavors of natural language processing. As work proceeds at one level of definition, specification, and implementation, a certain degree of rigor and expressiveness of the interlingua is attained. Still, however carefully this has been achieved, the expressive power of natural language, even in restricted subdomains, tends to always favor greater complexity. In effect a "new horizon" becomes visible, inviting a further increment to the interlingua development process.

This effect is minimized in two ways by our approach. First, the principle of parsimony combined with the nature of our target domains effectively limits the amount of information that the KANT interlingua needs to represent. There are certain phenomena (pragmatics factors, speaker intentions, discourse levels, etc.), which either are not significant in technical documentation or can be eliminated via rewriting. The KANT interlingua will not have to represent these phenomena and so certain extensions to the horizon become impossible. Second, due to the central role of the interlingua in the system, a rapid prototyping and incremental refinement strategy, similar to the spiral software development model [Boehm, 1985]), is necessary. By planning incremental refinement into the design process, the remaining horizon effect is transformed into a forcing function for each refinement iteration. In essence, the horizon effect becomes part of the design process, rather than a force opposing it.

There are also additional advantages of the incremental approach. As mentioned earlier, in following an incremental refinement strategy in designing the interlingua, both the analysis and generation components of the system could be developed in parallel not only with each other, but also with the development of the interlingua itself.

The interlingua is the totality of information passed by the analysis module to the generator. Since the interlingua plays this central role in the system, its design can easily become a bottleneck for the entire development process, as neither analysis nor generation can proceed without first having the interlingua specified. In addition, the relationships between the concepts represented in the interlingua also evolve incrementally as the related frame-based hierarchical domain model is refined.

By allowing for incremental development, the interlingua's central role in the system, rather than constituting a bottleneck, became for us a focal point of development effort. That is, both sides were able to be proceed using the latest information about the interlingua and provide feedback to the the interlingua design process.

For example, the semantic roles that correspond to English prepositional phrases were not specified until well into the interlingua design process. However, because the heart of the interlingua had already been specified, both the analysis and generation teams were able to work with this underspecified form (albeit without any source prepositional phrases being represented in the interlingua). Similarly, the domain model, which supplies the concepts used to head interlingua frames, was also developed in parallel. It was possible, when necessary to refine the grain size of the conceptual objects (e.g. to account for new subtleties of meaning) without affecting any other parts of the interlingua, and therefore without impacting more than necessary on the analysis and generation efforts.

A final advantage of adopting an incremental refinement strategy from the onset is that information obtained during testing may be used as design feedback. It is not uncommon for many representation problems to go unnoticed until actual text is fed through the MT system. In many cases, this form of feedback can be difficult to integrate back into the interlingual design, because the design and the design methodologies do not support iterative development. In our system, we were able to use the results of testing to create a better interlingua specification, rather than having to retrofit new structure into a existing inflexible base.

While in theory this approach does not eliminate the development bottleneck since the software development could overtake the interlingua development, in practice this is unlikely.

5 Conclusion

In this paper we have identified and described the approach that we have followed in the design and implementation of the interlingua for the KANT knowledge-based MT system. We have shown how, in order to achieve a large-scale practical system, fundamental software research and development principles must be followed. Our experience indicates that such efforts are only possible when the central knowledge representation is sufficiently expressive yet constrained, thorough yet practical, and well-specified yet extensible.

The reasoned approach we describe has been validated by the KANT application discussed, the first instantiation of which is about to be deployed at Caterpillar, Inc. for translation from English to French. The approach has been tested for Japanese, German, and Italian generation and development for large-scale Spanish and German generation components is already underway with additional languages to follow.

6 Acknowledgments

We would like to acknowledge the other members of the KANT team, in particular Jaime Carbonell, Eric Nyberg, Teruko Mitamura, Kathy Baker, Marion Kee, and William Walker. We also appreciate the collaboration of our associates at Carnegie Group Inc. and Caterpillar Inc.

References

[Boehm, 1985] Boehm, B. (1985). A spiral model of software development and enhancement. In *Proceedings of the International Workshop on Software Process and Software Environments.*

[Defrise, 1993] Defrise, C. (1993). Discours et traduction automatique: une approche interlangue basée sur les connaissances. In Bouillon, P. and Clas, A., editors, *La traductique*. Les Presses de l'Université de Montréal.

[Leavitt et al., 1993] Leavitt, J., Franz, A., and Lonsdale, D. (1993). The KANT interlingua specification. Technical Report CMU-CMT-93-143, Center for Machine Translation, Carnegie Mellon University.

[Nirenburg et al., 1992] Nirenburg, S., Carbonell, J., Tomita, M., and Goodman, K. (1992). *Machine Translation: A Knowledge-based Approach*. Morgan Kaufman, San Mateo, CA.

[Nirenburg and Defrise, 1994] Nirenburg, S. and Defrise, C. (1994). Application-oriented computation semantics. In Johnson, R. and Rosner, M., editors, *Computational Linguistics and Formal Semantics*. Cambridge University Press.

[Nyberg and Mitamura, 1992] Nyberg, E. and Mitamura, T. (1992). The KANT system: Fast, accurate, high-quality translation in practical domains. In *Coling-92*.

[Tsujii, 1988] Tsujii, J. (1988). What is a cross-linguistically valid interpretation of discourse? In Maxwell, D., Schubert, K., and Witkam, A. P. M., editors, *New Directions in Machine Translation*. Foris Publishers.

A Goal-Directed Multi-Level Stylistic Analyzer

Pat Hoyt and **Chrysanne DiMarco***
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
cdimarco@logos.uwaterloo.ca

Abstract

Sophisticated natural language processing systems should be able to deal with the subtle but significant effects of style on communication, but the difficulties of representing stylistic knowledge in a formal representation had resulted in only simplistic and heuristic approaches to implementation.

In this paper, we take the problem of formally representing stylistic knowledge as our starting point. It is our belief that stylistic knowledge must first be formalized, rendered in a well-defined representation, before a computational analysis of style can be attempted. And it is our further contention that a formal representation will facilitate a very transparent implementation. We show how a formal representation of syntactic style can be used as the basis for a general-purpose stylistic analyzer that can produce descriptions of the stylistic features of an input sentence at multiple levels of abstraction.

1 The importance of stylistic analysis in natural language processing

The importance of dealing with pragmatic aspects of language in computational systems is undeniable. People communicate a great deal of information through pragmatic nuances, and a knowledge of how these subtleties influence meaning is part of a full understanding of language. Systems that could analyze the effects of style on communication would provide information about the implicit meaning that is contained in a text. And generation systems that could control style would produce text that intentionally conveys a specific pragmatic effect. Both stylistic analysis and generation could be used in applications, such as text critiquing, second-language instruction, and machine translation, for which understanding the effects of how something is said is as important as understanding what is said. Ultimately, computational stylistics should be a part of any system that attempts to deal with 'real-world' language.

But very few natural language understanding systems have attempted to deal with issues of style,[1] and those that do have generally taken a simplistic and heuristic approach. Stylistic analysis has not yet developed the systematic and rigorous methods of syntactic analysis and semantic interpretation. Part of the reason is obvious: understanding style is hard. Stylistic effects are difficult to articulate and even more difficult to define.

In this paper, we take the problem of formally representing stylistic knowledge as our starting point. It is our belief that stylistic knowledge must first be formalized, rendered in a well-defined representation, before a computational analysis of style can be attempted. And it is our further contention that a formal representation will facilitate a very transparent implementation. We show how a formal representation of syntactic style can be used as the basis for a general-purpose stylistic analyzer that could be used as a component of a natural language processing system.

2 A theory of syntactic style

A formal representation of stylistic knowledge should ideally be based on an underlying linguistic theory: there should be a vocabulary of concepts, a clear definition of how the concepts are related, and a systematic way for building new concepts out of existing ones. In our earlier work [DiMarco and Hirst 1993; Green 1992], we presented a computational theory of syntactic style that is a multi-level representation of stylistic grammar rules. This section summarizes the details of this work as presented in [DiMarco and Hirst 1993]. Green [1992] develops the linguistic underpinnings of the theory; Hoyt [1993] presents the representation of the complete theory in a syntactic stylistic grammar.

*Please direct all correspondence to the second author.

[1] By *style*, we do not mean literary style, but rather the style of texts such as high-quality magazines and newspapers, technical manuals, and business correspondence.

2.1 Fundamental concepts

In designing a computational theory of style, we constructed a vocabulary of stylistic concepts at three levels of abstraction:

- *Primitive elements* are stylistically significant syntactic properties of sentence components.
- *Abstract elements* are general stylistic properties of groups of sentences.
- *Stylistic goals* are the writer's intentions for high-level pragmatic properties of text.

At all levels, the guiding principle of the theory is that style is *goal-directed*, that is, linguistic choices are made to achieve specific stylistic goals, such as clarity or abstraction. Therefore, we tie low-level syntactic choices to high-level stylistic goals. The fundamental concepts that are used to integrate the multiple levels of the theory are stylistic *concord* and *discord*, which we define as follows:

Concord: A stylistic construction that conforms to the norm for a given genre.

Discord: A stylistic construction that deviates from the norm.[2]

2.2 Primitive elements of style

At the lowest level of the theory, there are two views of sentence structure, *connective* and *hierarchic*:[3]

Connective ordering: The result of cohesive bonds drawing together components in a linear ordering.

Hierarchic ordering: The result of bonds of subordination and superordination drawing together components in a nested ordering.

The connective and hierarchic orderings are used in the definition of primitive stylistic elements to provide a precise syntactic basis to the theory, yet also allow a mapping to the abstract elements.

We use the terms *conjunct* and *antijunct* with superscripts to indicate the degree of connectivity or disconnectivity. Syntactic components are classified as either $conjunct^5$ or $conjunct^6$ (excessively connective), $conjunct^3$ or $conjunct^4$ (strongly connective), $conjunct^2$ (moderately connective), $conjunct^1$ (mildly connective), and $conjunct^0$ (neutral). Similarly, the terms $antijunct^0$ through $antijunct^4$ are used to indicate increasingly disconnective effects; $conjunct^0$ and $antijunct^0$ are the same.

There is a complementary vocabulary of primitive elements for the hierarchic view. The stylistic effects of syntactic components are correlated with the degree of subordination or superordination; the classifications are analogous to the connective: $subjunct^4$ through $subjunct^0$ (decreasingly subordinate) and $superjunct^0$ through $superjunct^4$ (increasingly superordinate); $subjunct^0$ and $superjunct^0$ are the same.

We adapted the work of Halliday and Hasan [1976] on cohesive relations to assign classifications to the connective elements. Halliday and Hasan consider substitution, including ellipsis, to be the most strictly cohesive relation, followed by reference, and then conjunction. We adopted this ranking, and so we classify intrasentential substitution and ellipsis as strongly connective ($conjunct^3$), reference as moderately connective ($conjunct^2$), and conjunction as mildly connective ($conjunct^1$). We also classify interpolation, parenthetical constructions, as disconnective ($antijunct^2$).

In assigning a hierarchic classification to a syntactic component, we adapted Halliday's [1985] work on *subordination*, specifically, embedding and hypotaxis, and the definition of the term *superordination* by Quirk *et al.* [1985]. We classify embeddings as strongly subordinate, $subjunct^3$, and hypotactic structures as only mildly subordinate, $subjunct^1$.

2.3 Abstract elements of style

The primitive elements of style are combined into patterns of *abstract elements* that describe general stylistic properties related to syntactic parallelism, structure nesting, and linear ordering. The abstract elements are defined as follows:

Homopoise: A sentence with interclausal coordination of syntactically similar components.

Heteropoise: A sentence in which one or more parenthetical components are syntactically 'detached' and dissimilar from the other components at the same level in the parse tree.[4]

Monoschematic: A sentence with a single main clause with simple phrasal subordination and no accompanying subordinate or coordinate clauses.

Centroschematic: A sentence with a central, dominant clause with one or more of the following optional features: complex phrasal subordination, initial dependent clauses, terminal dependent clauses.

Polyschematic: A sentence with more than one central, dominant clause and at least one dependent clause.

Resolution: A shift in stylistic effect that occurs at the end of a sentence and is a move from a relative discord to a stylistic concord.

Dissolution: A shift in stylistic effect that occurs at the end of a sentence and is a move from a relative concord to a stylistic discord.

The remaining abstract elements describe *concordant* or *discordant* stylistic effects in particular positions. The basic elements are *initial concord*, *medial concord*, and *final concord*, with a similar range of *discord* elements.

[2] Discord, in our view, is not necessarily 'bad'. Indeed, it is the strategic use of discord, deviation from the norm, that can give expressiveness to writing.

[3] These two complementary kinds of analysis are implicit in the work of most stylists and rhetoricians.

[4] A heteropoise can be *initial*, *medial*, or *final*, depending on the position of the parenthesis in the sentence.

2.4 Stylistic goals

As we have noted, the abstract elements are defined in terms of the lower-level primitive elements. The abstract elements are in turn used as the basis for the definition of higher-level *stylistic goals*. Stylistic goals can be organized along orthogonal dimensions. For example, a writer might try to be clear, or obscure, or make no effort either way. *Clarity* and *obscurity* are thus opposite ends of a stylistic dimension. Likewise, the goals of *concreteness* and *abstraction* form a dimension, and so do *staticness* and *dynamism*.

We adapted descriptions of stylistic goals from textbooks of style, such as [Kane 1983], and rewrote these descriptions in terms of our abstract elements. Clarity, for example, is characterized by *simple* monoschematic sentences, *centred* centroschematic sentences, and *parallel* homopoisal sentences. Concreteness is associated with sentences that *highlight* a particular component: these are our heteropoises and discords. And *staticness* is characteristic of *'fixed-form'* sentences in which there is little stylistic variation, that is, monoschematic or homopoisal sentences.

3 A stratified grammar of style

3.1 The style of the grammar

Our theory of syntactic style is now the basis for a grammar of style, which in turn will provide a specification for our stylistic analyzer. The hierarchical nature of the theory lends itself naturally to a stratified, context-free grammar. It is useful to think of the grammar as a means of recognizing a particular *style tree*, analogous to a syntactic parse tree. Just as a syntax tree is built up from individual words at the leaf nodes to a whole sentence at the root level, a style tree can be thought of as being built up from primitive elements at the leaf level to stylistic goals at the root. The syntax-tree analogy can be extended: we can consider the syntax-tree nodes to be annotated with stylistic terms, starting with the leaves and working up through the intermediate nodes to the root node.

To illustrate the structure of the grammar, we will present selected rules that build from simple syntactic components to full sentences.[5]

3.2 Level of primitive elements

3.2.1 Basic components

In the development of our theory of style, we were especially concerned with the relationship between style and the structure of the nominal group. As a consequence, a large number of the rules in our grammar involve definitions of premodification and postmodification. These definitions are built up from adjectivals, nouns, prepositional phrases, *etc.* One of the connective rules for postmodification defines conjunct[1] postmodification, which deals with the case of a prepositional phrase, a conjunctive element. In the hierarchic view, a prepositional phrase is classified as subjunct[3] postmodification, as it is an embedded element.

conjunct[1] postmodification ⟶
    prepositional phrase

subjunct[3] postmodification ⟶
    prepositional phrase

We introduce the notion of a *transitional level* in the grammar, in order to clearly separate the levels of primitive elements and abstract elements. At this level, primitive elements are combined into transitional elements, which directly indicate the abstract elements of which they can be a part. For example, the rules below define the kinds of postmodification that can be used in building a monoschematic, centroschematic, or concordant sentence.

monoschematic postmodification ⟶
    subjunct[0] postmodification
    subjunct[3] postmodification *and*
    (nominal group *or* prepositional phrase)

centroschematic postmodification ⟶
    conjunct[i] postmodification *where* $0 \leq i \leq 4$
    subjunct[i] postmodification *where* $0 \leq i \leq 3$

concordant postmodification ⟶
    conjunct[i] postmodification *where* $0 \leq i \leq 4$
    subjunct[i] postmodification *where* $0 \leq i \leq 3$

3.2.2 Noun phrases

The various types of premodification and postmodification are combined into different kinds of noun phrases. In the examples below, we define the kinds of noun phrases that can be incorporated into monoschematic, centroschematic, and concordant sentences.

monoschematic noun phrase ⟶
    noun phrase *with*
        (monoschematic premodification *and*
        monoschematic postmodification)

centroschematic noun phrase ⟶
    noun phrase *with*
        (centroschematic premodification *and*
        centroschematic postmodification)

concordant noun phrase ⟶
    noun phrase *with*
        (concordant premodification *and*
        concordant postmodification)

[5]The rules are taken from Hoyt's [1993] full syntactic stylistic grammar of 240 rules, which is a revised and extended version of the preliminary grammar presented in [DiMarco and Hirst 1993]. The notation used in the grammar is explained in the Appendix to this paper.

3.2.3 Main clauses

In an analogous manner, the rules for the other major sentence components (prepositional phrases, complements, verb phrases, and dependent clauses) are built up from primitive elements to form transitional elements. The various types of majors, or main clauses, can then be defined from component transitional elements, as in the following examples.

monoschematic major ⟶

major *with*
(monoschematic noun phrase *and*
monoschematic verb phrase)

centroschematic major ⟶

major *with*
(centroschematic noun phrase *and*
centroschematic verb phrase)

concordant major ⟶

major *with*
(concordant noun phrase *and*
concordant verb phrase)

3.2.4 Complete sentences

Finally, we define rules for complete sentences, which consist of at least one main clause, with optional dependent clauses. Selected rules are as follows:

monoschematic complete ⟶

monoschematic major

centroschematic complete ⟶

(concordant clause)* centroschematic major
(concordant clause)*

concordant complete ⟶

(concordant clause)* concordant major
(concordant clause)*

initial concordant complete ⟶

concordant major (clause)*

(concordant clause)* major (clause)*

3.3 Levels of abstract elements and stylistic goals

At the level of abstract elements, the various types of complete sentences are used to define stylistic terms such as monoschematic, centroschematic, and initial concord:

monoschematic ⟶

monoschematic complete

centroschematic ⟶

centroschematic complete

initial concord ⟶

initial concordant complete

Finally, at the top level, the abstract elements are used to define stylistic goals. For example, as we described in section 2.4, clarity would be defined by the following rule:

clarity ⟶

monoschematic

centroschematic

homopoise

3.4 An application of the grammar

The following short example illustrates the kind of analysis that the stylistic grammar can be used to produce for the sentence *True, posterity has been kind.*[6]

3.4.1 Primitive-element analysis

The sentence is concordant, for it consists of a concordant main clause, the major, with no subordinate clauses. It begins with a style disjunct, *true*, which is an elliptic adjectival and therefore considered to have a connective, concordant effect, even if used in the initial, parenthetical, position. After the initial disjunct, the sentence continues with the bare noun *posterity*, which, lacking both premodification and postmodification is a minimal, and therefore concordant, noun phrase. The sentence ends with the basic verb phrase *has been kind*, consisting of only the copula *been*, and the concordant, conjunct[1] adjective *kind*; this is an inherently concordant verb phrase.

The sentence is concordant from the hierarchic view as well, for it has the form of a concordant initial heteropoisal complete sentence. This indicates that the sentence begins with a parenthetical construction, which in this case is the disjunct, *true*, a superordinate adjectival. The bare noun *posterity*, lacking both premodification and postmodification, is a monoschematic noun phrase. The verb phrase *has been kind* is basic and therefore monoschematic.

3.4.2 Abstract-element analysis

In the connective view, the significant elements are initial and medial concords—the sentence is both monoschematic and trivially centroschematic. It is also an initial heteropoise.

In the hierarchic view, the sentence is centroschematic and an initial heteropoise. It is the initial disjunct, *true*, that introduces a superordinate effect; this feature makes the sentence slightly too complex to be monoschematic.

[6] The next six paragraphs have been adapted from [DiMarco and Hirst 1993].

3.4.3 Stylistic-goal analysis

The presence of the concords in the connective view, together with the connective and hierarchic centroschematic structures, give the sentence an effect of clarity. In a less obvious manner, the presence of an initial disjunct affects other stylistic goals. Because a superordinate, parenthetical, component is present, the sentence is a heteropoise and therefore considered to be concrete.

To summarize, this is a simple, clear sentence with the slight incongruity of an initial parenthesis to relieve its blandness.

4 A stratified stylistic analyzer

4.1 General design

Our theory of syntactic style is represented by a corresponding set of grammar rules that defines the relationship between syntactic structures and stylistic effects. Now, we will use this grammar of style as the specification for a stylistic analyzer, ASSET, that will produce stylistic parses of input sentences. In designing ASSET, we were influenced by the following considerations:

Evaluation of the theory: We viewed ASSET as an essential tool for testing and evaluating our theory of style.

Parser independence: A syntax-based stylistic analysis of a sentence will obviously include a syntactic parse of the sentence. Thus, an ordinary parser is a necessary part of any stylistic analyzer. Our decision to make ASSET totally independent of the parser was in part theoretical—ASSET would not have to compromise theory because of limitations and/or methodology of the parser—and pragmatic—developing a parser from scratch was beyond the scope of our work.

This requirement meant that the sentence must be parsed before the stylistic analysis. This allows the substitution of parsers within the system with only the requirement that a module be created to transform the output of a particular parser into the specified format for ASSET.

Modularity: Future work on the theory will include refinements to the abstract elements and transition elements, so the prospect of these revisions made modularity, good software engineering practice in any case, a necessity.

Efficiency: ASSET must be reasonably efficient.

Independence from potential uses: The potential applications of a stylistic analyzer include intelligent computer-assisted language instruction (ICALI) and machine translation (MT). At the present state of development of ICALI and MT, it is impossible to know exactly which information and what representation would be most useful. This implied that, in addition to letting the user know which stylistic goal(s), if any, have been met, all stylistic information generated during the analysis must be part of the output of ASSET.

[[[[[none], complement], [[[runs], lexical_verb], verb], verb_phrase], [[[[[[[none], postmodifier], [[[park], lexical_noun], noun], [[[[the], definite_article], adjectival], premodifier], nominal_group], [[in], preposition], prepositional_phrase], postmodifier], [[[man], lexical_noun], noun], [[[[the], definite_article], adjectival], premodifier], nominal_group], noun_phrase], major], complete]

Figure 1: ASSET's input in its list-structure form.

The need to have all stylistic information available further implied that the analysis of one part of the sentence, *e.g.*, the noun phrase, cannot constrain that of another, *e.g.*, the verb phrase. To obtain some degree of efficiency, in spite of the lack of constraints on the analysis, a bottom-up, or leaf-to-root, approach is used. A syntax tree that parallels the syntactic organization of our grammar is the basic structure of ASSET. This tree is represented as a list structure that describes a breadth-first, right-to-left traversal. Figure 1 shows the list structure that is the input to ASSET for the simple sentence *The man in the park runs.*

The parser used in the development of ASSET is Pundit[7] (Prolog UNderstands Integrated Text), chosen because of its fairly large syntactic coverage and its comprehensive treatment of conjunctions. These are necessary features for the analysis of stylistically interesting sentences. Pundit uses a *restrictive grammar*, written as a set of BNF (Backus-Naur Form) rules. Pundit's output consists of a syntactic tree in this BNF form; it is this output that is transformed into the parse tree input into ASSET.

4.2 The representation of the grammar in ASSET

ASSET's processing mechanism is data-driven, so that the grammar rules are represented declaratively in a database. ASSET is essentially a 'tree-walker' that traverses the parse tree, annotating the nodes with stylistic information. The grammar rules have the form shown in Figure 2.[8]

As ASSET walks through the parse tree, it uses the grammar representation to match the pattern of annotations currently recorded at a node. At each stage, ASSET 'knows' what it needs to look for because of the consistency in the way the grammar is constructed: The grammar is *stratificational*,[9] so that elements at each level are composed from elements at the level below. As a consequence, ASSET need only look at a small set of

[7] Pundit is a natural language understanding system developed by the Unisys Corporation.

[8] In Figures 2 and 6, the abstract-element terms *connective, hierarchic, monoschematic, centroschematic*, and *concordant* have been abbreviated to *conn, hier, mono, centro*, and *concord* respectively. The *te* affix indicates a transition-element analysis that is dependent on the corresponding transition-element analysis of the syntactic component's (*i.e.*, the current node's) children.

[9] The stratificational nature of our grammar was influenced by Sydney M. Lamb's work, in particular, *Outline of stratificational grammar*, Georgetown University Press, 1966.

```
postmodification(conn, prepositional_phrase,
                 conjunct1).
postmodification(hier, prepositional_phrase,
                 pp_subjunct3).

postmodification(conn_te, conjunct1,
                 [centro, concord]).
postmodification(hier_te, pp_subjunct3,
                 [mono, concord]).

nominal_group(postmodification(concord),
              [premodification(concord)], concord).
noun_phrase(nominal_group(centro), [], centro).

major(noun_phrase(concord), [verb_phrase(concord)],
      concord).
complete(major(centro), [], centro).

abstract_elements(complete(centro), [], centro).
stylistic_goals(abstract_elements(centro), [],
                clarity).
```

Figure 2: Sample ASSET grammar rules

1: Transform the parser output into format specified for ASSET (TRANSFORMATION MODULE).
2: Annotate the input tree with primitive element classifications (ANNOTATION MODULE).
3: Assign abstract elements to the input sentence (ABSTRACT ELEMENT MODULE).
4: Assign stylistic goal(s) to the input sentence (STYLISTIC GOAL MODULE).
5: Output the annotated tree structure.

Figure 3: The general algorithm for ASSET

possible rules at each stage to decide on the next increment in the annotation of the stylistic parse tree.

4.3 The processing modules

The general algorithm for the ASSET system is shown in Figure 3, along with an accompanying illustration of its architecture in Figure 4. The Transformation Module is responsible for changing Pundit's output into the form, as shown in Figure 1, specified for ASSET.

The Annotation Module is responsible for the task of analyzing the style of the input sentence at the primitive-element and transition-element levels. The algorithm is shown in Figure 5. There are two submodules that annotate the nodes of the input tree with stylistic information:

Primitive Element Module (PEM): This module is responsible for analyzing the appropriate nodes by using the primitive-element layer of our computational theory. Each node is analyzed from both the connective and hierarchical viewpoints. The result of the analysis is a node annotated with the primitive stylistic descriptions: either a conjunct or an antijunct element and either a

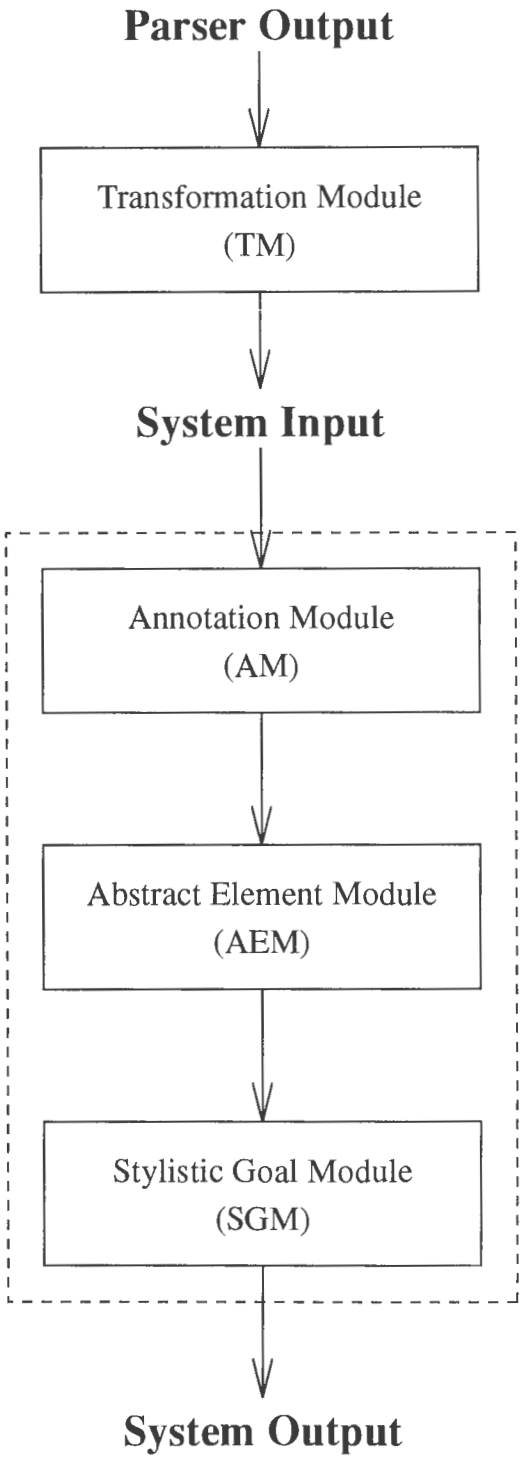


Figure 4: The overall architecture of ASSET

1: Annotate 'near' leaf nodes with primitive-element classifications (PRIMITIVE ELEMENT MODULE).

1a. Analyze from the connective viewpoint (CONNECTIVE MODULE).

1b. Analyze from the hierarchical viewpoint (HIERARCHICAL MODULE).

2: Annotate the rest of the nodes with transition-element classifications (TRANSITION ELEMENT MODULE).

Figure 5: The algorithm for the Annotation Module.

subjunct or a superjunct element.

Transition Element Module (TEM): This module takes the parse tree, previously annotated by the PEM, and annotates the rest of the nodes with abstract-element terms. The TEM uses information provided by the primitive-element classification of nodes lower in the parse tree.

It should be noted that the PEM and the TEM do not work sequentially: because of the bottom-up processing, calls to the TEM occur whenever the PEM has annotated a sufficient number of nodes lower in the parse tree. Thus, calls to the PEM and the TEM are interleaved with each other.

After the primitive-element and transition-element analyses have been completed, the input sentence is then classified in terms of the abstract elements and the stylistic goals. A fully annotated parse tree is input to the Abstract Element Module, which then adds abstract element information to the structure and passes it on to the Stylistic Goal Module. Once the stylistic goals have been determined, the output structure is complete. Figure 6 shows all the stylistic information contained in the output structure for the sentence *The man in the park runs.*

5 Conclusion

Computational stylistic analysis and generation should ideally be components of any sophisticated natural language processing system. However, the difficulties of representing stylistic knowledge in a form amenable to computational implementation meant that only *ad hoc* approaches had previously been attempted in dealing with matters of style in NLP systems.

We have shown how a formal theory of style can be represented by a multi-level grammar that describes the relationship between low-level syntactic structures and high-level stylistic goals. In turn, this grammar has been used as the specification for an implementation, ASSET, which produces stylistic analyses at multiple levels of abstraction. We can foresee such a general-purpose stylistic analyzer used as a component of NLP systems, such as for second-language instruction or machine translation, to produce additional information that contributes to the full understanding of the implicit meaning of a text.

Acknowledgements

We would like to thank Graeme Hirst for helpfully reading earlier drafts of this paper and providing very useful advice. We acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre.

Appendix: Notes on terminology

At all levels of the grammar, the left-hand side of each rule identifies what is being defined, and the right-hand side lists one or more alternative realizations, one per line.

In the grammar, we use various shorthand notations to simplify the presentation of the rules. However, these abbreviated forms can be expanded into standard context-free grammar rules. The shorthand notations are as follows; they are illustrated by particular examples, but are intended for general use:

1. adjectival ⟶ intensifier adjective

   The juxtaposition of terms on the right-hand side of a rule indicates a concatenation of instances of these terms. For example, the rule above allows the intensifier *very* to be followed by the adjective *happy* to form an adjectival, *very happy.*

2. adjectival ⟶ (intensifier)* adjective

   The Kleene star indicates zero or more occurrences of the form within parentheses.

3. noun phrase *with* centroschematic postmodification

   Where a rule has several alternatives, this shorthand notation using *with* abbreviates a long sequence of alternatives (here, the different types of centroschematic postmodification).

4. noun phrase *with*
   (centroschematic premodification *and*
   centroschematic postmodification)

   And indicates that all conditions on the right-hand side of a rule must simultaneously be satisfied by a single constituent.

5. noun phrase *with*
   postmodification *and*
   (nominal group *or* prepositional phrase)

   Or indicates that any one of the conditions on the right-hand side of a rule must be satisfied.

References

DiMarco, Chrysanne and Hirst, Graeme. "A computational theory of goal-directed style in syntax." *Computational Linguistics*, **19**(3), 448–497, September 1993.

```
stylistic_goals(clarity,staticness)

abstract_elements(mono,concord,initial_concord,medial_concord, final_concord)

complete([mono,concord,initial_concord,medial_concord, final_concord])
    major([mono,concord])

    noun_phrase(([centro,mono,concord])
      nominal_group(([centro,mono,concord])

          premodification([conjunct1],[subjunct2], [centro,mono,concord])
            adjectival([conjunct1],[subjunct2])
              definite_article(the)

          noun([conjunct0])
              lexical_noun(man)

          postmodification([conjunct1],[subjunct3], [centro,mono,concord])
            prepositional_phrase([centro,mono,concord])

              preposition(in)
              nominal_group([centro,mono,concord])

                premodification([conjunct1],[subjunct2], [centro,mono,concord])
                  adjectival([conjunct1],[subjunct2])
                    definite_article(the)

                noun([conjunct0])
                     lexical_noun(park)

                postmodification([conjunct0],[subjunct0], [centro,mono,concord])
                  postmodification(none)

    verb_phrase([mono,concord])
      verb(runs)

      complement([mono,concord])
            complement(none)
```

Figure 6: An example of ASSET's output.

Green, Stephen J. *A functional theory of style for natural language generation.* Master's thesis, Department of Computer Science, University of Waterloo, 1992. [University of Waterloo Faculty of Mathematics Technical Report CS-92-48].

Halliday, M.A.K. *Introduction to functional grammar.* Edward Arnold, London, 1985.

Halliday, M.A.K. and Hasan, Ruqaiya. *Cohesion in English.* Longman Group Limited, London, 1976.

Hoyt, Patricia A. *A goal-directed functionally-based stylistic analyzer.* Master's thesis, Department of Computer Science, University of Waterloo, 1993. [University of Waterloo Faculty of Mathematics Technical Report CS-93-48].

Quirk, Randolph, Greenbaum, Sidney, Leech, Geoffrey, and Svartvik, Jan. *A comprehensive grammar of the English language.* Longman Group Limited, 1985.

On Multiple-Valued Deductive Databases

Eli Hagen
Veronica Dahl
School of Computing Science
Simon Fraser University
Burnaby, BC
Canada V5A 1S6
E-mail: {hagen,veronica}@cs.sfu.ca

Abstract

We propose a multiple-valued typed logic that can serve as a query language to deductive databases, and which makes it possible to give more informative answers to queries which would otherwise simply fail or produce misleading answers. This language can also be used as an analyser's internal representation of natural language queries. One important consequence of this work is our view of a database as a mapping into relations with multiple truth values- a view that can be transposed to other query language frameworks as well.

1 Introduction

The idea of using more than two logical values for deductive databases is not new. In [Colmerauer, 1982] and [Dahl, 1982], for instance, three logical values are used for allowing subtler responses to database queries. The inspiration for this third value springs largely from considering the semantics of natural language in order to allow natural language queries.

Recent work on consulting deductive databases through less computationally studied human languages—in particular, American Sign Language—and within a framework of cooperative responses, has pointed to the desirability of further logic value distinctions ([Hagen, 1993]).

In this paper we propose a new class of deductive databases which provides natural, helpful answers to queries, through the use of a multiple valued semantics and of incomplete types for quick reasoning on type hierarchies. These databases are also tailored in particular to be used in conjunction with logic-based natural language analysers, (see e.g. [Abramson and Dahl, 1989; Stabler, 1992; Johnson, 1991]).

Our line of research makes it possible to use logic throughout the whole database system: as the database definition language, as the database query language (after an analyser has translated natural language queries into logic), as the data manipulation language, and as the analyser's language (also see [Pereira and Warren, 1980; Dahl, 1981]). This uniformity of representation helps minimize the interfacing between different components of a natural language consultable deductive database. We use Prolog as our implementation language.

Cooperative responses have long been in the literature of question answering systems (e.g. [Kaplan and Joshi, 1978]). An example would be, for the question: "How many square planets rotate around the sun?", an explanation that there are no square planets, as opposed to the true but uninformative reply: "None", which leaves open the possibility that square planets rotate around other celestial objects.

Many approaches to cooperative responses have been proposed: probability values (e.g., [Lee, 1992], extensions of relational operations onto tables with different types on null values (e.g., [Liu and Sunderraman, 1990], meta language techniques (e.g., [Leveque, 1981; Konolige, 1981; Bowen and Kowalski, 1982]). A survey of the field can be found in [Gaasterland *et al.*, 1992]. To the best of our knowledge, cooperative responses have not been treated within deductive databases through the use of the multiple truth values as we propose. Our approach stresses the provision of cooperative responses from a well defined theoretical standpoint (a rigorous multi valued logic system), combined with a smooth integration with natural language front ends.

2 Multiple Logic Values—Motivation

The truth values *true* and *false* largely retain the meaning they have in a traditional binary logic. However, the new values allow for subtler distinctions than are possible in a binary logic, so some statements that in a binary logic would evaluate to *false* will now evaluate to *pointless, absurd, mixed, unknown,* or *inconsistent.* Next we motivate each of these new logical values.

2.1 Pointless

This value was introduced in L3 (see [Dahl, 1981]) for detection of failed presupposition induced by the definite article. We cover it briefly here for completeness.[1] Consider the following statement:

1) The sales report that John wrote was ready in one day.

[1] In [Dahl, 1981] this value is called *undefined.*

If we evaluate this with respect to a knowledge base representing a world in which Alice, not John, wrote the sales report, it is assigned the value *pointless* rather than *false*, since *false* would imply that the negation of the statement:

2) The sales report that John wrote was not ready in one day.

would have to be acknowledged as *true*. The article *the* presupposes existence and uniqueness of its referent, so if John did not write the sales report, we say that the presupposition failed, and because the presupposition failed, it is pointless to say whether the report was or wasn't ready in one day.

2.2 Absurd

Some statements contradict basic world knowledge rather than making a wrong assumption. Consider the following example:

3) Can Rover speak Latin?

If we evaluate this statement with respect to a knowledge base of ordinary dogs, it evaluates to *absurd* since ordinary dogs don't speak. While in the previous example we can conceive of John having written a sales report, this statement is considered semantically anomalous.

Some statements are syntactically ambiguous, but have only one semantically acceptable interpretation. Humans are not likely to choose the wrong interpretation in these cases, but a computer may, since both interpretations are syntactically correct. Consider the following question:

4) What is the price of a recorder which can play stereo music?

It admits two interpretations: the intended one, where the recorder plays stereo music (i.e., "recorder" is the antecedent of the relative clause) and an unintended one, where the price plays stereo music (i.e., "price" is the antecedent of the relative clause). The latter interpretation should evaluate to *absurd*.

2.3 Vague

The value *vague* helps us deal with statements that are under-specified. For instance, the question

5) Who teaches Norwegian?

could result in the database system sweeping through the domain of all people in the knowledge base, if the interrogative pronoun *who* is used to find a domain for the object queried. Vague-referring pronouns could induce a *vague* truth value, indicating that further domain determination can perhaps be done from the context.

2.4 Mixed

Some relationships distribute a certain property among all the elements of their arguments. For example, the relation **like** in the following example:

6) Ann and Tom like karate.

To evaluate this statement, the database system checks whether Ann likes karate and whether Tom likes karate. If both (none) of them do, the statement evaluates to *true (false)*. But if one likes karate, while the other does not, it evaluates to *mixed*, which is a much more informative reply than what we can expect from a binary logic or L3, where the statement would simply evaluate to *false*.

The value *mixed* is also useful when evaluating relations that are introduced by the word "respectively". For example, the statement

7) Tom and Ann earn \$1000 and \$1100 respectively

evaluates to *mixed* if Tom earns \$1000 while Ann does not earn \$1100 (or if Ann earns \$1100 and Tom does not earn \$1000).

2.5 Unknown

Generally speaking, we shall adhere to the closed world assumption ([Reiter, 1978]), which allows us to infer $\neg A$ from a logical program P, if A is not a logical consequence of P. In logic programming, negation ($\neg A$) is implemented as failure, i.e., only positive information is listed in the database, and if a closed world database system cannot prove a given query, we infer that it is false.

Open world databases admit positive, negative, and unknown information, but it is expensive to implement open worlds since the negative and unknown information usually exceed the positive information by several orders of magnitude. Since one has to distinguish between unknown and negative information, one of them has to be defined explicitly and thus the conciseness allowed by negation-as-default is lost.

However, for relations whose arguments are in a one-to-many relationship, we introduce a way of simulating an open world situation, while only including positive information in the knowledge base. In this situation we distinguish between *false* and *unknown* queries by introducing a metalogical treatment of values retrieved from the knowledge base. Consider the following example:

8) Does Ann live in Paris?

This query can be compiled into something like: "Find the place X where Ann lives; if X is ground and equal to "Paris", evaluate the query to *true*; if X is ground and different from "Paris", evaluate the query to *false*, otherwise to *unknown* (i.e., failure implies *unknown*). The *unknown* value allows us to produce an informative reply instead of a misleading negative answer.

Only for one-to-many relations can we allow failure to imply *unknown*, and for many-to-many relations we must maintain the traditional interpretation of negation as failure. Consider the following example: knowing that Betsy is Pepe's aunt does not, in an open world, authorize us to conclude that Doreen is not, since Pepe can have many aunts. Thus for many-to-many relationships we shall stay within closed worlds, use negation as failure, and only test for *false* versus *unknown* those relationships in which two arguments are in a one-to-many relationship.

This approach should, of course, be used with caution, and it certainly does not solve all open world problems, as we have seen. But it does provide a compromise that allows greater flexibility than completely closed worlds.

2.6 Inconsistent

In a binary logic, some queries evaluate to *false* because they are inconsistent to begin with. For instance, in a world where all secretaries are insured and all part-time employees are not (hence implying that no secretaries are part-time), it would be inconsistent to query:

9) Which part-time secretaries are insured?

Rather than simply answering "None", which does not point out the inconsistency, it would be more helpful to reply, for instance, that all secretaries are insured whereas part-time employees are not.

Rigorously speaking, the difference between *inconsistent* and *absurd* is a matter of degree—*absurd* being assigned when the situation is unimaginable in the world considered, and *inconsistent* being assigned for forbidding situations that are not inconceivable but are disallowed in our database (e.g., integrity constraints). Inconsistent queries could also evaluate to *pointless* since the user presupposes something that is not consistent with the state of the database. However, we choose to separate the *inconsistent* case from the *absurd* and the *pointless* cases since the mistakes are indeed different. We think that database systems should have the flexibility of distinguishing between these values in order to provide appropriately informative responses in each case.

3 Query Evaluation during Parsing

Our database system is designed to work together with a natural language front end, so some queries might be assigned truth values during the natural language parsing stage, while others will be assigned values during query evaluation, i.e., during the database consultation stage. When a query is assigned a truth value during the parsing stage, it is only partially executed. A complete formula representing the query is never generated-i.e., the reply is available without ever having to consult the database system. Only queries that are found not to be semantically anomalous by the parser produce a complete formula and become completely evaluated with respect to the database system.

In order to detect semantic anomalies, we introduce types in the grammar and knowledge base, i.e., relations range over *typed* variables and constants. We briefly review our notion of types (section 3.1, for a complete discussion, see [Dahl, 1991]) before discussing how they are used in query evaluation (section 3.2).

3.1 Incomplete Types

Every constant and variable in our database are assigned an *incomplete type*. For example, `A-[animal | X]`, stands for a variable `A` of type `[animal | X]`. This type is incomplete in that it contains a tail variable `X`, that allows for further instantiation. Thus `[animal | X]` stands for "at least of animal type". Another example: `sammy-[animal, seal, sammy]`, stands for the constant `sammy` of type `[animal, seal, sammy]`. The type representation for constants is closed such that no further instantiation is possible.

Formally an incomplete type is a term of the form: $t_1 \supset \cdots \supset t_{n-1} \supset t \supset V$, where V is a variable ranging over the incomplete type $t_1 \supset \cdots \supset t_{n-1} \supset t$ and where there exists no t_0 such that $t_0 \supset t_1$. In Prolog notation, this would be a list of the form $[t_1, \ldots, t_{n-1}, t \mid V]$, where $\mid$ is a binary infix operator that separates the tail from the rest of the list.

3.2 Query Evaluation

The values *absurd* and *inconsistent* indicate semantic anomalies these logical values are assigned as a result of type incompatibility.

We can require the grammar to enforce type agreement between the arguments of a relation and the arguments of the main verb of a sentence. Returning to example number 3 above:

Can Rover speak Latin?

The concept of speaking may be represented by the following relation in the knowledge base:

speak(Subject-[person | X], Object-[language | Y])

where the first argument of **speak** is of type *person*, while the query may introduce the formula:

speak(rover-[animal,dog,rover], latin-[language,latin]),

where the type of the first argument is *dog*. Since the types of the two first arguments don't match (Subject-[person | X] and rover-[animal, dog, rover] cannot unify) the parse is immediately interrupted and the value *absurd* is assigned to the query. This value is then interpreted by a natural language output module that will print a message indicating the reason for disagreement.

Types can also serve to disambiguate some natural language queries which have more than one syntactically correct reading. Returning to example 4):

What is the price of a recorder which can play stereo music?

If the concept of playing is represented by the following relation in the knowledge base

play(Subject-[inanimate,device|X], Object-[music|Y])

and if the type of 'price' is *price* and the type of 'recorder' is *device*, the reading in which a price is required to play stereo music is made impossible, since [inanimate, device | X] and [inanimate, price | Z] cannot unify. Again the incorrect reading is assigned the truth value *absurd* at the parsing stage and can simply be discarded.

Sometimes ambiguities arise because words mean different things in different situations. Consider the word 'bank' in the following examples:

1. With which bank do you have an account?
2. On which bank did you sit?
3. On which bank did you fish?

In the first example, bank = money bank, in the second, bank = river bank, and in the last example, bank = fishing bank. If the relations **have_account_with**, **sit**, and **fish** all insist on their arguments having different types, the incorrect readings are easily detected at the parsing stage.

Inconsistent queries can be detected through type incompatibility. Recall our previous example where all full time employees and all secretaries are insured. Then the query:

Which part-time secretaries are insured?

introduces a type mismatch that interrupts the parse.

The value *vague* is just a conceptual tool that prompts for further domain determination and is never actually assigned to a query. The domain determination is a consequence of adding semantic information to the grammar and knowledge base. Recall our previous example:

Who teaches Norwegian?

The concept of teaching may be represented by the following relation in the knowledge base:

teach(Subject-[person,teacher|X], Object-[course|Y])

i.e., the first argument of **teach** is of type *teacher*, while the query might introduce the following formula:

teach(QSubject-[person|Z],norwegian-[course,norwegian])

i.e., the interrogative pronoun introduces a first argument of type *person*. Since *teacher* is a subtype of *person*, the two 'Subject' arguments are not incompatible and we take the intersection of the two types ($person \cap teacher = teacher$) to be the further specified type of Q_Subject. Through semantic agreement and unification (Z unifies with [teacher | X] as a result of parsing the query) the vagueness introduced by the interrogative pronoun simply disappears and the search space is considerably narrowed from *person* to *teacher* without ever having derived *vague* an explicit truth value.

Since semantically anomalous queries never reach the database consultation stage, it is unnecessary to include the values *absurd*, *inconsistent* and *vague* in the definition of the logical system underlying our database and query language. All the other values, *true*, *false*, *mixed*, *unknown* and *pointless*, are assigned during query evaluation and therefore have to be defined in the logical system that underlies our deductive database and query language. Out of space considerations, we will not include the formal definitions here, but instead we give an intuitive account of our multi-valued typed logic database and its query language in section 4. The complete formal definiton can be found in [Hagen, 1993]

3.3 Intensional Replies

If the database is *typed*, we can easily obtain intensional replies, and if the user requests an extensional reply, the search space is automatically reduced to the domains that are compatible with respect to type. See [Dahl, 1991] for a discussion on using types for intensional replies.

4 A Typed Multiple-Valued Deductive Database

As discussed earlier, only some of the values need to appear explicitly in the definitions. Every relation in our database evaluates to one of the values *true*, *false*, *mixed*, and *unknown*. However, all four values may not apply to a particular relation, for example, any *many-to-many* relations can *not* evaluate to *unknown* since we adhere to the closed world assumption for these relations. Complete queries evaluate to either *true*, *false*, *mixed*, *unknown*, or *pointless* in accordance with the logical system defined in [Hagen, 1993] (also see section 5).

4.1 Partitioning of Relations

Natural language sentences can introduce different types of plural, e.g., a relation may apply to a whole set as in "The beams are parallel", where the relation **parallel** must apply to the whole set of beams, while in our old example "Ann and Tom like karate" the relation **like** distributes to individual members of the set. Our system can recognize different kinds of plural and since they are treated differently, the relations must be partitioned into the disjoint groups *distributive*, *inherently collective*, *partially collective*, *respective*, and *single attribute relations*. The first four groups are further divided into many-to-many and one-to-many relations according to their arguments' relationship to each other.

4.2 Distributive Relations

Distributive relations are relations which distribute a certain property to all individuals of a set. For example, in the statement "Eli and Dia speak English", the property "speak" distributes to both Eli and Dia and we may infer that the two statements "Eli speaks English" and "Dia speaks English" are both *true*. Distributive relations are divided into *many-to-many* and *one-to-many* relations.

4.2.1 Many-to-many Distributive Relations

A many-to-many relation has two attributes that are in a many-to-many relationship with one another. An instance of a many-to-many distributive relation evaluates to *true*, *false*, or *mixed* as follows:

— If the relation holds on every member of the set, the instance evaluates to *true*.

— If the relation fails on every member of the set, the instance evaluates to *false*.

— If the relation holds on some and fails on the rest of the members of the set, the instance evaluates to *mixed*.

An example: The following instance of the many-to-many distributive relation **speak**:

speak([eli, dia], [english, spanish])
(= "Eli and Dia speak English and Spanish.")

evaluates to:

a. *true* if applied to the knowledge base:
{ **speak**(dia, spanish),
speak(dia, english),
speak(eli, spanish),
speak(eli, english) }

b. *false* if applied to the knowledge base:

{ **speak**(jörg, english) }

c. *mixed* if applied to the knowledge base:
speak(dia, english),
speak(eli, english) }

4.2.2 One-to-many Distributive Relations

A one-to-many relation has two attributes that are in a one-to-many relationship with one another. An instance of a one-to-many distributive relation evaluates to *true, false, unknown,* or *mixed* as follows:

— If the relation holds on every member of the set, the instance evaluates to *true.*

— If the relation fails on every member of the set, the instance evaluates to *false.*

— If the relation is unknown for one or more members of the set, the instance evaluates to *unknown.*

— If the relation holds on some and fails on the rest of the members of the set, the instance evaluates to *mixed.*

An example: The following instance of the one-to-many distributive relation **born_in**:

born_in(finland, [eli, dia, jörg])
(= "Eli, Dia and Jörg were born in Finland.")

evaluates to

a. *true* if applied to the knowledge base:
{ **born_in**(finland, eli),
born_in(finland, dia),
born_in(finland, jörg) }
b. *false* if applied to the knowledge base:
{ **born_in**(norway, eli),
born_in(uruguay, dia),
born_in(germany, jörg) }
c. *mixed* if applied to the knowledge base:
{ **born_in**(norway, eli),
born_in(finland, dia),
born_in(germany, jörg) }
d. *unknown* if applied to the knowledge base:
{ **born_in**(finland, eli),
born_in(germany, jörg) }
e. *unknown* if applied to the knowledge base:
{ **born_in**(finland, eli),
born_in(finland, jörg) }
f. *unknown* if applied to the knowledge base:
{ **born_in**(norway, eli),
born_in(germany, jörg) }

4.3 Inherently Collective Relations

Inherently collective relations are relations where a certain task is done collectively by a whole set of individuals, i.e., each set member is necessary but not sufficient for the successful completion of the task. For example, the statement "Ala, Brigitte and Dia lifted the heavy table" describes a situation where Ala, Brigitte and Dia lifted a heavy table *together,* and we cannot infer that, for example, the statement "Ala and Dia lifted the heavy table" is *true.*

4.3.1 Many-to-many Inherently Collective Relations

An instance of an inherently collective many-to-many relation evaluates to *true* or *false* as follows:

— If the relation holds on the set, the instance evaluates to *true.*

— If the relation fails on the set, the instance evaluates to *false.*

An example: The following instance of the many-to-many inherently collective relation **raise** (Say, a set of nuns raise a set of orphans and the raising is a collective job such that no child is being raised by any particular nun, and vice versa.):

raise([nun1, nun2], [child1, child2, child3])
(="Nun1 and nun2 raised child1, child2, child3.")

evaluates to:

a. *true* if applied to the knowledge base:
{ **raise**([nun1, nun2], [child1, child2, child3]) }
b. *false* if applied to the knowledge base:
{ **raise**([nun1, nun2, nun3], [child1, child2, child3]) }
c. *false* if applied to the knowledge base:
{ **raise**([nun1, nun2], [child1, child2, child3, child4]) }
d. *false* if applied to the knowledge base:
{ **raise**([nun3, nun4, nun5], [child1, child2, child3]) }

4.3.2 One-to-many Inherently Collective Relations

An instance of a one-to-many inherently collective relation evaluates to *true, false,* or *unknown* as follows: — If the relation holds on the set, the instance evaluates to *true.*

— If the relation fails on the set and there is no contradictory information available, the instance evaluates to *unknown.*

— If the relation fails on the set and there is contradictory information available, the instance evaluates to *false.*

An example: The following instance of the one-to-many inherently collective relation **lift**:

lift(table, [ala, dia])
(= "Ala and Dia lifted the table.")

evaluates to:

a. *true* if applied to the knowledge base:
{ **lift**(table, [ala, dia]) }
b. *false* if applied to the knowledge base:
{ **lift**(table, [andrea, kaci]) }
c. *false* if applied to the knowledge base:
{ **lift**(table, [ala, dia, jörg]) }
d. *unknown* if applied to the knowledge base:
{ **lift**(bookcase, [jörg, allan]) }

4.4 Partially Collective Relations

Partially collective relations are also relations where a certain task is done collectively by a whole set of individuals, but where a subset of the original set may be sufficient to satisfy the relation. For example, if we define **lift** in our previous example to be a partially collective

relation, we can infer that the statement "Dia and Ala lifted the heavy table" is true from the statement "Ala, Brigitte and Dia lifted the heavy table".

4.4.1 Many-to-many Partially Collective Relations

An instance of a many-to-many partially collective relation evaluates to *true* or *false* as follows.

— If the sets of the instance are subsets of sets in the knowledge base, the instance evaluates to *true*.

— If the sets of the instance are not subsets of sets in the knowledge base, the instance evaluates to *false*.

An example: Reconsider the nuns and children example above, but assume that **raise** is a partially collective relation instead of an inherently collective relation.

> **raise**([nun1, nun2], [child1, child2, child3])
> (= "Nun1 and nun2 raised child1, child2, child3.")

evaluates to:

a. *true* if applied to the knowledge base:
{ **raise**([nun1, nun2], [child1, child2, child3]) }
b. *true* if applied to the knowledge base:
{ **raise**([nun1, nun2, nun3], [child1, child2, child3]) }
c. *true* if applied to the knowledge base:
{ **raise**([nun1, nun2], [child1, child2, child3, child4]) }
d. *false* if applied to the knowledge base:
{ **raise**([nun3, nun4, nun5], [child1, child2, child3]) }
e. *false* if applied to the knowledge base:
{ **raise**([nun2, nun4, nun5], [child1, child2, child3]) }

4.4.2 One-to-many Partially Collective Relations

An instance of a one-to-many partially collective relation evaluates to *true, false*, or *unknown* as follows.

— If the sets of the instance are subsets of sets in the knowledge base, the instance evaluates to true.

— If the sets of the instance are subsets of sets in the knowledge base that contradicts the instance, the instance evaluates to false.

— If the sets of the instance are not subsets of sets in the knowledge base, the instance evaluates to unknown.

An example: The following instance of the one-to-many partially collective relation **meet**:

> **meet**(park, [ala, dia]) (= "Dia and Ala met in the park.")

evaluates to:

a. *true* if applied to the knowledge base:
{ **meet**(park, [ala, dia]) }
b. *true* if applied to the knowledge base:
{ **meet**(park, [eli, ala, dia]) }
b. *false* if applied to the knowledge base:
{ **meet**(school, [ala, dia]) }
c. *false* if applied to the knowledge base:
{ **meet**(school, [eli, ala, dia]) }
d. *unknown* if applied to the knowledge base:
{ **meet**(park, [jörg, allan]) }

Note, we can collapse the two classes of collective relations into one class if we introduce another logical value *yes_but* and make the partially collective relations that now evaluate to *true*, evaluate to *yes_but*, while we keep the interpretations of *true*, *false*, and *unknown* as defined above for inherently collective relations.

4.5 Respective Relations

Unlike distributive and collective relations, respective relations do not reflect an inherent property of a given concept. The most common way of introducing a respective relation is to include the word "respectively" in a statement. Since respective relations are not introduced by a concept itself, both distributive and collective relations can participate in respective relations. Consider the following examples,

— The one-to-many distributive relation **born_in**(*country*, *human*); From the statement "Dia and Ala were born in Uruguay and Poland (respectively)" we may infer that the statements "Dia was born in Uruguay" and "Ala was born in Poland" are both *true*, and that, for example, the statements "Dia was born in Poland" and "Dia was born in Norway" are false.

— The one-to-many collective relation **meet**(*place*, *human*); From the statement "Brigitte and Edwin and Eli and Jörg met in the park and at school respectively" we may infer that the statements "Brigitte and Edwin met in the park" and "Eli and Jörg met at school" are true.

It is the job of the natural language front end to detect the words that introduce respective relations.

Since all the relations discussed earlier can participate in respective relations, we need to create a corresponding "respective" symbol for each one of them such that the database system can distinguish between respective and non-respective queries. The respective symbol is only for identification purposes, and the query is evaluated with respect to **r**. In this document, we simply create new symbols by attaching the suffix **_resp** to the original symbol **r** (e.g., **speak** becomes **speak_resp**).

There is one class of respective relations for each class of distributive, and collective relations and we name these *respective distributive, respective inherently collective* and *respective partially collective*. In addition to new "respective" constraints, respective relations inherit the argument properties of the original relation (e.g., a one-to-many relation is still one-to-many). Consequently, each respective class is divided into many-to-many and one-to-many relations. We next discuss these two cases for distributive respective relations.

4.5.1 Many-to-many Respective Relations

An instance of a many-to-many respective relation evaluates to *true, false*, or *mixed* as follows:

— If the relation holds on every member of the set, the instance evaluates to *true*.

— If the relation fails on every member of the set, the instance evaluates to *false*.

— If the relation holds on some and fails on the rest of the members of the set, the instance evaluates to *mixed*.

An example: The following instance of the many-to-many respective relation **speak_resp**:

speak_resp([[eli, jörg], [veronica, dia]], [german, spanish]) (= "Eli and Jörg and Veronica and Dia speak German and Spanish respectively.")

evaluates to:

a. *true* if applied to the knowledge base:
{ **speak**(eli, german),
speak(jörg, german),
speak(veronica, spanish),
speak(dia, spanish) }
b. *false* if applied to the knowledge base:
{ **speak**(brigitte, german) }
c. *mixed* if applied to the knowledge base:
{ **speak**(dia, spanish),
speak(jörg, german),

4.5.2 One-to-many Respective Relations

An instance of a one-to-many respective relation evaluates to *true, false, unknown*, or *mixed* as follows:
— If the relation holds on every member of the set, the instance evaluates to *true*.
— If the relation fails on every member of the set, the instance evaluates to *false*.
— If the relation is unknown for one or more members of the set, the instance evaluates to *unknown*.
— If the relation holds on some and fails on the rest of the members of the set, the instance evaluates to *mixed*.
An example: The following instance of the one-to-many respective relation **earn_resp**:

earn_resp([1000, 1100], [brigitte, ann]) (= "Brigitte and Ann earn $1000 and $1100 respectively.")

evaluates to:

a. *true* if applied to the knowledge base:
{ **earn**(1000, brigitte),
earn(1100, ann) }
b. *false* if applied to the knowledge base:
{ **earn**(900, brigitte),
earn(1200, ann) }
c. *mixed* if applied to the knowledge base:
{ **earn**(1000, brigitte),
earn(1200, ann) }
d. *unknown* if applied to the knowledge base:
{ **earn**(1000, allan) }

5 A Logical System for a Subset of Natural Language

We briefly summarize the logical system underlying our multi-valued database system. The NL analyser builds formulae of the input queries from: set formulae s, statement formulae e, and integer formulae n. A set formula can take any of the forms:

- a list of constants
- a variable
- *those*(V, e), where V is a variable and e is a statement formula

A statement formula e can take any of the forms:

- $for(V, e_1, e_2)$, where V is a variable, e_1, e_2 are statement formulae
- $r(s_1, \ldots, s_n)$, where r is a relational symbol (eg., **speak, raise**), $s_1, \ldots, s_n$ are set formulae
- $and(e_1, e_2)$, where e_1, e_2 are statement formulae
- $if(e_1, e_2)$, where e_1, e_2 are statement formulae
- $not(e_1)$, where e_1 is a statement formula
- $equal(n_1, n_2)$, where n_1, n_2 are integer formulae
- $greater\_than(n_1, n_2)$, where n_1, n_2 are integer formulae

An integer formula n, can take any of the following forms:

- $j \in \mathcal{N}$
- card(s), where s is a set formula

In a well defined situation a statement formula will evaluate to *true, false, mixed, unknown*, or *pointless*, a set formula will evaluate to a set, and an integer formula will evaluate to an integer. For a complete definition of the logical system see [Hagen, 1993].

6 Concluding Remarks

In this article, we have motivated the use of multiple truth values to provide more helpful answers to natural language querying of deductive databases, and we have introduced a query language based on these new values.

We have also integrated these notions into the formal definition of a multivalued deductive database and of an associated query language's semantics. Our approach has been tested within a system that translates American Sign Language into our multi-valued logic for the purpose of direct consultation of database information using sign language [Hagen, 1993].[2] This particular application provided interesting feedback to our approach itself given its unique features. For instance, the distinction between some of the kinds of plurals we propose is actually explicit in ASL syntax, whereas in oral natural languages it cannot be recovered from the syntax. With minor variants, this query language can be adapted to suit different concrete applications.

Other approaches to cooperative answering propose more sophisticated capabilities than the ones proposed here, e.g., how to add information of interest to a response even though the query does not mention the topic [Cuppens and Demolombe, 1988] or how to relax queries when a response to the original query fails [Chu *et al.*, 1991; Gaasterland *et al.*, 1992]. Each of these approaches offers solutions to individual problems in cooperative answering, while our approach offers solutions to several of the problem areas (presuppositions, misconceptions, and intensional answers) within a single framework. It

[2] This prototype assumes a previous phase of translation from visual input into a symbolic representation, which is its input.

integrates a reasonable amount of cooperative answering within a system which is based on a rigorous logic system serving simultaneously as a natural language representation system and as a database query language. With this first step in the integration of cooperative answers within such a framework, we hope to provide a solid basis for extensions into more sophisticated cooperative features which can maintain our focus on a rigorous theoretical standpoint and a smooth integration with natural language front ends.

Many of the features we have introduced in our formalization and our treatment of queries have been inspired on previous work on natural language processing. From this point of view, the present work is a practical consequence of our advocacy for cross-fertilization between Linguistics and AI ([Dahl, 1993]). With the present work we hope to inspire more research into the interactions between language and deductive databases.

Acknowledgements

We are grateful to the Logic and Functional Programming Lab at Simon Fraser University, in which this work was developed, and to CSS, LCCR and the School of Computing Science for the use of their facilities. This research was supported by NSERC grants no. #31-611024 and #31-61519036, by PRG grant #13871180, and by CSS grants # 872050 and #872049, and Simon Fraser University .

References

[Abramson and Dahl, 1989] H. Abramson and V. Dahl. *Logic Grammars.* Symbolic Computation AI Series. Springer Verlag, 1989.

[Bowen and Kowalski, 1982] K. A. Bowen and R. A. Kowalski. Amalgamating Language and Metalanguage in Logic Programming. In K. L. Clark and S.-A. Tärnlund, editor, *Logic Programming*, pages 153–172. Academic Press, New York, 1982.

[Chu *et al.*, 1991] W. W. Chu, Q. Chen, and R. Lee. Cooperative Answering via Type Abstraction Hierarchy. In S. M. Deen, editor, *Cooperating Knowledge Based Systems 1990*, pages 271–290. Springer Verlag, 1991.

[Colmerauer, 1982] A. Colmerauer. An Interesting Subset of Natural Language. In K.L. Clark and S.-A. Tärnlund, editor, *Logic Programming*, pages 45–66. Academic Press, Inc., 1982.

[Cuppens and Demolombe, 1988] F. Cuppens and R. Demolombe. Cooperative Answering: a Methodology to Provide Intelligent Access to Databases. In *Proc. of 2nd Int'l. Conf. on Expert Database Systems*, pages 333–353, 1988.

[Dahl, 1981] V. Dahl. Translating Spanish into Logic through Logic. *American Journal of Computational Linguistics*, 7(3):149–164, 1981.

[Dahl, 1982] V. Dahl. On Database Systems Development Through Logic. *ACM Transactions on Database Systems*, 7(1):102–123, 1982.

[Dahl, 1991] V. Dahl. Incomplete Types for Logic Databases. *Applied Math. Letters*, 4(3):25–28, 1991.

[Dahl, 1993] V. Dahl. What the Study of Language Can Contribute to AI. *AI Communications*, 6(2):92–106, 1993.

[Gaasterland *et al.*, 1992] T. Gaasterland, P. Godfrey, and J. Minker. An Overview of Cooperative Answering. *J. of Intelligent Systems*, 1(2), 1992.

[Hagen, 1993] E. Hagen. A Flexible American Sign Language Interface to Deductive Databases. Master's thesis, School of Computing Science, Simon Fraser University, 1993.

[Johnson, 1991] M. Johnson. Deductive Parsing: The Use of Knowledge of Language. In Berwick et al., editor, *Principle-Based Parsing: Computation and Psycholinguistics.* Kluwer Academic Pubishers, 1991.

[Kaplan and Joshi, 1978] S. J. Kaplan and A. K. Joshi. Cooperative Responses: An Application of Discourse Inference to Data Base Query Systems. In *Proc. Canadian Society For Computational Studies of Intelligence*, 1978.

[Konolige, 1981] K. Konolige. A Metalanguage Representation of Relational Databases for Deductive Question-Answering Systems. In *Proc., 7th Int'l. Joint Conf. on Artificial Intelligence*, pages 496–503, 1981.

[Lee, 1992] S. K. Lee. Imprecise and Uncertain Information in Databases: An Evidential Approach. In *Proc. 8th Int'l. Conf. on Data Engineering*, pages 614–621, 1992.

[Leveque, 1981] H. Leveque. The Interaction with Incomplete Knowledge Bases: A Formal Treatment. In *Proc., 7th Int'l. Joint Conf. on Artificial Intelligence*, pages 240–245, 1981.

[Liu and Sunderraman, 1990] K.-C. Liu and R. Sunderraman. Indefinite and Maybe Information in Relational Databases. *ACM Transactions on Database Systems*, 15(1):1–39, 1990.

[Pereira and Warren, 1980] F. C. N. Pereira and D. H. D Warren. Definite Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Transition Networks. *Artificial Intelligence*, 13:231–278, 1980.

[Reiter, 1978] R. Reiter. On Closed World Databases. In H. Gallaire and J. Minker, editor, *Logic and Databases*, pages 55–76. Plenum Press, 1978.

[Stabler, 1992] E. Stabler. Implementing Government Binding Theories. In R. Levine, editor, *Formal Grammar: Theory and Implementation.* Oxford University Press, 1992.

Learning Repetition in String Transformations

Natascha O. Schüler and Bruce A. MacDonald
Department of Computer Science, University of Calgary,
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
email: bruce@cpsc.ucalgary.ca

Abstract

This paper describes a system, L-EBE, that learns text rewrite rules containing loops, by learning a description of both the input pattern and how that pattern is transformed into the output. Rules are learned from input/output examples. The work is an extension of the non–iterative EBE system built by Nix [1983; 1985]. Loop annotations are introduced to EBE's language, which describes a sequence of constant and variable substrings. Constraints on the examples ensure the detection of nested loops with different numbers of iterations. L-EBE has been implemented in Scheme and integrated with the Gnu Emacs editor, so it is convenient for users to present examples to the learner, and also execute learned transformation rules during editing.

1 Introduction

Text editing on a computer can be a laborious and repetitive manual chore (for example bibliography translation and formatting). Most text editors have alleviated this a little with simple fixed text replacement commands and macro recording techniques, but these are inflexible. Some provide powerful regular expression replacement, but this can be awkward and therefore is not widely used. Convenient automation of repetitive text editing operations would remove much of the unwanted burden on the user, leaving more freedom for creative writing. Computers must become more attuned to human instruction and interaction, rather than expecting humans to learn complicated formal command languages [MacDonald, 1991].

From a user's viewpoint, a convenient method of automation would observe a few editing examples and then be able to take over the remainder of the task, as well as remembering it for future use. The editing examples provide cases of the input, editing operations,and resulting output, which might be analyzed by a machine learning system to produce an appropriate editing transformation. However, a user's editing behaviour is specific to the editor command set, may vary from case to case since there are a myriad of ways to carry out an editing task, and can be difficult to analyze. Our goal in this work is to show that text transformations with loops can be learned from the input/output behaviour alone. Although this means throwing away potentially useful trace information, it provides a more portable automation system that is less sensitive to user behaviour.

$\%\uparrow_{loop_1} \mathrm{A}_{\sqcup} g_1{:}C^*{,}_{\sqcup} g_2{:}U^*.g_3{:}U^*.$ <eol>
$\%\downarrow^{loop_1} \mathrm{D}_{\sqcup} g_4{:}N^*$<eol>
$\%\mathrm{T}_{\sqcup} g_5{:}C^*$<eol>
$\%\mathrm{J}_{\sqcup} g_6{:}C^*$<eol>
$\%\mathrm{O}_{\sqcup} g_7{:}A^*$<eol>
$\%\mathrm{K}_{\sqcup}{}^*$
$\Rightarrow$
$\uparrow_{loop_1} g_1{,}_{\sqcup} g_2{.}g_3{.}$; $\downarrow^{loop_1}(g_4)$,<eol>
"g_5",$_{\sqcup} g_6$,<eol>
g_7.

Figure 1: A Bibliography Transformation in L-EBE (after gap merging). This is the iterative string transformation L-EBE learns from the examples of Table 1.

This paper describes L-EBE, a system for learning text transformations with loop structures and variables, from input/output behaviour. Given a user–supplied set of input/output text examples L-EBE learns a rewrite rule expressed as the transformation of an input pattern to an output one; a pattern comprises constant strings and variables, annotated with loop structure, unary boolean predicates, and unary transformations. Table 1 shows an example that formats "refer" style bibliography entries. Figure 1 shows the transformation learned, and will be explained in later sections.

L-EBE is an extension of EBE [Nix, 1983; Nix, 1985], which learns non–iterative rewrite rules for the editing domain. Briefly, L-EBE first creates a non–iterative pattern for the input examples, using EBE's method of (a) finding the longest common subsequence, and (b) inserting variables. Then L-EBE extracts repeating structure to form a set of possible iterative input patterns. Finally, variable substitutions are used to create an output pattern for each example, and these are merged to form the final output pattern, along with the corresponding input pattern. We assume the teacher is benevolent, but not aware of the internals of L-EBE.

Input Examples	Output Examples
%A␣Abi-Ezzi,␣S.S.<eol> %D␣1986<eol> %T␣An␣implementer's␣view␣of␣PHIGS<eol> %J␣Computer␣Graphics␣and␣Applications<eol> %O␣February<eol> %K␣*<eol>	Abi-Ezzi,␣S.S.;␣(1986),<eol> "An␣implementer's␣view␣of␣PHIGS",␣ Computer␣Graphics␣and␣Applications,<eol> February.<eol>
%A␣Addis,␣T.R.<eol> %A␣Hinton,␣G.E.<eol> %D␣1987<eol> %T␣A␣framework␣for␣knowledge␣elicitation<eol> %J␣Proc␣First␣European␣Conference␣on␣Knowledge␣ Acquisition<eol> %O␣September<eol> %K␣*<eol>	Addis,␣T.R.;␣Hinton,␣G.E.;␣(1987),<eol> "A␣framework␣for␣knowledge␣elicitation",␣ Proc␣First␣European␣Conference␣on␣Knowledge␣ Acquisition,<eol> September.<eol>
%A␣Allen,␣J.F.<eol> %A␣Koomen,␣J.A.<eol> %D␣1986<eol> %T␣Planning␣using␣a␣temporal␣world␣model<eol> %J␣Artificial␣Intelligence<eol> %O␣March<eol> %K␣*<eol>	Allen,␣J.F.;␣Koomen,␣J.A.;␣(1986),<eol> "Planning␣using␣a␣temporal␣world␣model",␣ Artificial␣Intelligence,<eol> March.<eol>
%A␣Allen,␣J.F.<eol> %D␣1983<eol> %T␣Maintaining␣knowledge␣about␣temporal␣instances<eol> %J␣Comm␣ACM<eol> %O␣May<eol> %K␣*<eol>	Allen,␣J.F.;␣(1983),<eol> "Maintaining␣knowledge␣about␣temporal␣instances",␣ Comm␣ACM,<eol> May.<eol>

Table 1: A Text Transformation Task with Iteration. The input examples are formatted in "refer" style, while the outputs are plain text.

The input pattern learning component is closely related to extended regular string pattern learning from positive examples [Shinohara, 1982]. A string pattern is a string over domain and variable alphabets [Angluin, 1979; Angluin, 1980]. Angluin describes an algorithm for learning a suitable k–variable pattern in which a set of *feasible patterns* is defined for each of the strings in a sample, by constraining the number of constant and variable symbols as well as the locations of variable symbols. Automata are constructed from these patterns, and the intersection of these automata then defines a pattern for the entire sample. The algorithm is polynomial for one-variable patterns, but worse than polynomial for k–variable patterns. In an extended regular string pattern each variable occurs exactly once, variables are ordered canonically in the pattern, and null substitutes are allowed [Shinohara, 1982]. A suitable pattern is learned by first finding the longest common substring of the examples, then inserting variables so that the pattern matches the entire sample. The algorithm is polynomial for arbitrary k, and Nix's [1983; 1985] gap pattern synthesis bears a strong resemblance to it (although Nix does not allow null substitutes).

The rest of this section introduces the representation and an example of iterative text transformation. Section 2 presents the L-EBE system. Section 3 presents results for the example in Table 1. Section 4 mentions related work.

1.1 Representing Iterative Text Transformations

The input "gap" pattern and the output "replacement expression" pattern in a text transformation are each represented as a string composed from (a) the domain symbols, (b) the "gap" variable alphabet with annotations, and (c) the set of loop start and end symbols $\uparrow_{loop_j}$ and $\downarrow^{loop_j}$ respectively. The domain symbol set may be a finite alphabet of symbols (such as characters), or the set of strings over an alphabet, if the input is tokenised. By convention gap variables are denoted g_1, g_2, ... A gap pattern must meet these conditions:

- Each variable must be followed by a non–empty domain string.
- Each inter–gap constant sequence s_j following gap g_j must occur only as the suffix in the concatenation of a gap variable substitute for g_j and s_j, so that gap variables can be determined without backtracking. In addition a gap substitution must satisfy any boolean symbol class annotation (which is similar to a type check). A substitute meeting these two conditions is a legal one.
- Any loop sequence must contain at least one gap, ie., constant loops will not be considered.
- Gaps cannot be adjacent to one another. While this is easily described in non–iterative gap patterns, in the iterative gap pattern this also prevents a loop subsequence beginning and ending with a gap.

- Any loop subsequence must be followed by a constant (not a gap) so that the end of the loop can be determined.
- Loops may be nested, but may not overlap.

The set of substitutions that enables a gap pattern to match an input example is referred to as the *parse* of that example. The replacement expression must meet these criteria:

- It must match each output example when the respective input variable substitutions are made.
- The variables must be a subset of the input variable set.
- Any loop must contain at least one gap, ie., must be driven by a gap variable.
- Each gap variable occurring in the replacement expression must be nested (by loops) at the same depth as that variables appear in the gap pattern.
- Loops may be nested, but may not overlap.

These conditions reduce the effort the learning algorithm must expend in finding loops. However, a systematic analysis of the benefits has yet to be completed [Schüler, 1993].

Annotations Unrestricted gap variable formation overgeneralizes what may be substituted in the gap, so L-EBE restricts gap substitutions by annotations for symbol classes from the hierarchy in Figure 2. Gap substitution is overly restrictive if a legal gap substitute is merely copied to the output string as specified by the replacement expression, so L-EBE allows gap substitutes to be transformed by a composition of a small number of functions. The current implementation allows up to one basic function (selected from: lowercase, uppercase, capitalize) and one tabular function (selected from: month#, month, ord, ord# for converting between month names and numbers, and between numbers and ordinals). These function names are given as annotations to the replacement expression gap, while a boolean test is annotated to the gap in the input pattern to ensure that substitutes are members of the domain of any tabular function. So the examples

3.4.1991, ... ⇒ The␣third␣of␣April,␣1991 ...
5.8.1992, ... ⇒ The␣fifth␣of␣August,␣1992 ...

produce the transformation

$$g_1{:}N^*{:}ord\#?\,.\,g_2{:}N^*{:}month\#?\,.\,g_3{:}N^*\,,\ldots \Rightarrow$$
$$\text{The}\,␣\,g_1{:}ord\,␣\,\text{of}\,␣\,g_2{:}month{:}capitalize:\ g_3\,,\ldots$$

This gap pattern initially searches for any text that matches the regular expression

$$[0\text{-}9]*\,.\,[0\text{-}9]*\,.\,[0\text{-}9]*\,,\ldots$$

but discards matches in which the substitute for g_1 is not in the domain of the function *ord* or if the gap substitute for g_2 is not a valid month number. The substitute for g_1 is converted to an ordinal name and the substitute for g_2 to a capitalized month name.

A gap pattern matches any string for which (a) legal gap substitutes can be found, (b) inter–gap constants match, and (c) loop patterns match zero or more occurrences in the string. An output is generated by substituting for transformed gap variables in the replacement expression, expanding loops by the number of gap substitutes found.

During learning we seek a *descriptive* gap pattern. Each member of the set of descriptive gap patterns has a maximum of domain symbols and a minimum of gaps, and matches all the input examples.

2 Learning Text Transformations with Iteration

We make five simplifying assumptions about the input/output examples, to ease the difficulty of finding loops. First, (1) each of the examples must show at least one iteration of every loop that is to be learned. If there are different numbers of iterations in different examples then: (2) the loop body must be terminated by a constant, and (3) when a loop begins with a gap variable, then the gap substitutes must not have a prefix in common with the loop body terminating constant. If there are several iterations in each example then: (4) only loops of zero and one or more repetitions are learned, and (5) if nested loops are to be learned, the first iteration of any loop must be a complete prototype of that loop (ie the first iteration must contain the nested loops). These assumptions are somewhat ad hoc, and require further study.

The learning of iterative gap patterns is best described in three stages: (I) creation of a single gap pattern without loops; (II) creation of potential looping gap patterns; and (III) the construction of a replacement expression. Our initial aim was to make the minimum of changes to the methods used by Nix [1983; 1985]. There is not room to describe the algorithm details here; a brief description and some insight is given, while full details appear in [Schüler, 1993].

2.1 Step I: Constructing Non-Looping Gap Patterns

The criteria for a descriptive gap pattern suggest a division of this first step into: (a) finding the constants of the pattern; and then (b) inserting the minimum number of gaps into the constant sequence to make it match all of the input examples. The constant sequence is approximated by the longest common subsequence [Hirshberg, 1975] of the input examples. To complete a descriptive gap pattern, gaps are inserted into the *LCS* so that the pattern matches each of the input examples individually. When that has been achieved the pattern may have extraneous gaps (not needed for matching), so these are deleted.

2.2 Step II: Constructing Gap Patterns with Loops

When step I is used on inputs that contain repeating sequences, we can make some useful observations about the resulting gap pattern and substitutions. Along with the requirement that any examples show at least one iteration of each loop, the observations enable a technique for forming loops when there is repetition.

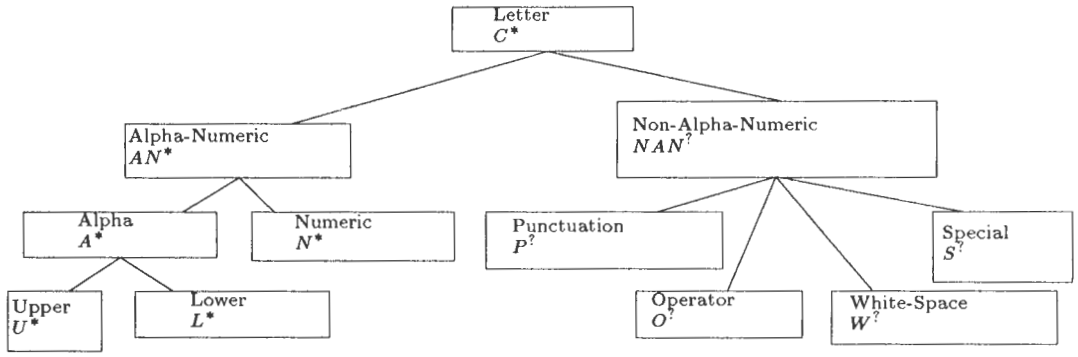


Figure 2: Character Class Hierarchy for Gap Annotation.

1. If there is repetition in the examples, and more than one iteration is shared across all the examples — such as two initials shared by each of the examples in Table 1 — then this repetition is also found in the gap pattern. In figure 3 a. the boxed sequences are repetitions that could be loops in the iterative gap pattern.

2. If there is repetition in the examples, but not all the examples share the same number of iterations, then the shared iterations can be found in the gap pattern as noted above. Any remaining iterations are matched by the last gap in the last repetition. This last gap in the loop and the constant string following it will be referred to as the *endgap* and *endconstant* of the loop. In Table 1, all examples share one author,[1] but the second and third examples each have two authors. The subsequence "%A$_{\sqcup}g_1$,$_{\sqcup}g_2$.g_3.<eol>" in figure 3 a. matches the initial iteration of the list of authors, while gap variable g_3 also matches both substitutes for g_3 and any extra iterations that an example may have. In example 2 of figure 3 b. gap g_3 matches both its substitute "R" for the current iteration and another complete iteration ".<eol>% A$_{\sqcup}$Hinton,$_{\sqcup}$G.E",[2] while in example 1 g_3 matches no further iterations.

Therefore, loop identification finds loops in the gap pattern by looking for repetitions (*repeat loops*) and examining an initial parse for the examples (*parse loops*). If the endconstant is non–null, the endgap can be found, and this gap's parses can be split to reveal different iterations of the loop, and these can be analyzed to find a gap pattern that forms the loop body (by recursively invoking the procedure for finding gap patterns). If this gap pattern for the loop body also matches the repetition preceding the endgap, then a parse loop has been identified.

[1]Remember: One iteration is the minimum required for the examples given.

[2]Note that g_3 matches the constant end of the current iteration and the beginning of the next iteration up to and including the substitute for g_3 in the last iteration of the loop.

However, loop locations cannot be confirmed until a replacement expression is generated. For this reason, L-EBE constructs *all* possible gap patterns with loops or zero and one or more repetitions, passing these to step III, along with the initial non–iterative gap pattern. These different gap patterns are ordered by decreasing specificity of the input examples: loops found through 1 above generalize the pattern, while loops found through 2 specialize the pattern. The set of potential gap patterns is represented as a nondeterministic finite automaton, where multiple transitions represent different alternatives (only one of these can be a transition on a constant or gap variable, the rest are on loop symbols).

2.3 Step III: Replacement expression synthesis with loops

The basic process in generating replacement expressions is to form a finite state automaton for each input/output example, using the parse of the input to construct the output. Such an automaton represents all the possible replacement expressions, including possible functions as annotations. The intersection of these automata represents the possible replacement expressions. The expression we desire is the shortest accepting path through the automaton created by this intersection.

The basic idea for finding loops in the replacement expression is to find all possible loops in the replacement expressions for each output by noting those gap variables that appear in loops in the iterative gap pattern (*intra-example loop construction*), while the intersection of these replacement expressions selects the correct loops (*inter-example loop construction*) [similar to ETAR's [Heise, 1989] intra- and inter-loop construction].

After constructing the automaton for the set of non–looping replacement expressions[3], intra-loop construction scans this machine for the locations of gap variables that occur in loops in the iterative gap pattern (in descending order of nesting level) and attempts to construct loops around them. Looking at two adjacent gaps in the replacement machine:

[3]These may be invalid, because the nesting levels of gaps, ie. the loops surrounding them, have not been considered.

%A ␣ g_1 , ␣ [g_2 . | g_3 .] <eol>
%D ␣ g_4 <eol>
%T [␣ g_5 | ␣ g_6 | ␣ g_7 | ␣ g_8 | ␣ g_9] <eol>
%J [␣ g_{10} | ␣ g_{11}] <eol>
%O ␣ g_{12} <eol>
%K ␣ *

a.

	Example#1	Example#2	Example#3	Example#4
g_1	"Abi-Ezzi"	"Addis"	"Allen"	"Allen"
g_2	"S"	"T"	"J"	"J"
g_3	"S"	"R.<eol>%A␣Hinton ,␣G.E"	"F.<eol>%A␣Koomen ,␣J.A"	"F"
g_4	1986	1987	1986	1983
g_5	"An"	"A"	"Planning"	"Maintaining"
g_6	"implementer's"	"framework"	"using"	"knowledge"
g_7	"view"	"for"	"a"	"about"
g_8	"of"	"knowledge"	"temporal"	"temporal"
g_9	"PHIGS"	"acquisition"	"world␣module"	"instances"
g_{10}	"Computer"	"Proc"	"Artificial"	"Comm"
g_{11}	"Graphics␣ and␣ Applications"	"First␣European␣Conference ␣on ␣Knowledge␣ Acquisition"	"Intelligence"	"ACM"
g_{12}	"February"	"September"	"March"	"May"

b.

Figure 3: The Effects of Iterative Examples on a. the Non-Iterative Gap Pattern and b. its Parse (Examples are from Table 1).

1. If the two gaps are different (ie., they have different numbers), but are both nested in loops at the same depth in the input, then the two gaps may be part of one iteration of the same loop, or they may belong to two separate loops.
2. If the two gaps are the same, and they appear in loops in the iterative gap pattern, then there must be a loop in the replacement expressions such that each of the gaps corresponds to a different iteration of that loop.
3. If the two gaps are nested at different depths in the input, then the difference in depths denotes the number of loop transitions, ie., loop starts or ends, that must occur between them in the replacement expression.

These motivate the search for loops in the replacement expression. Intra-example loop construction finds loops in the replacement expressions from the bottom up, ie., loops that are nested at the deepest level are located first. This order is necessary because these *inner loops* may be necessary for locating and merging *outer loops.*

Under the following conditions a loop will appear in the intersected machine:

1. Both replacement machines have a loop at the states that are currently being merged and the loop bodies (ie., the transitions between the loop begin and end symbol) can be matched.
2. At the states that are currently being merged one machine has a loop and the other has a marked sequence, and the loop body and part of the marked sequence (starting from the current state) can be matched.

When an output example contains only a single iteration of a loop, then it is impossible to determine the boundaries of the loop for that example. So during intra-example loop generation, any of the transitions in the replacement machine for that example may be *marked.* The boundaries of that loop are identified in inter-example loop generation when that replacement machine is intersected with others, for which the loop boundaries could be identified.

At the end of the merging process, if any marked transitions remain (ie. there are gaps that are not at the correct nesting level) then no replacement expression is found and the next iterative gap pattern is examined.

After a successful intersection of all the replacement machines, the shortest replacement expression is selected. In terms of loops, this means that a path involving a loop is selected over alternate non-loop paths because the loop contains a gap and therefore any of the alternate paths are invalid with respect to the input expression. That is, if a loop was constructed then its gap variables are nested at the correct level with respect to the iterative gap pattern. For any alternate paths that do not involve these loops, the gap variable must therefore be nested at the wrong level.

Finally, if there is a sequence of the same gaps separated by whitespace in both the gap pattern and the replacement expression, then these may be merged in to one single gap, and any character class annotation generalized to cover all the classes of the merged gaps.

2.4 User hints

What type of examples are suitable for L-EBE? This is a question that requires further study. The following are hints and guidelines for users.

- *Use common elements in the examples only when intended.* The stronger the differences are between the constant and the variable parts of the programs to be learned, the easier it is for L-EBE to learn them. This does not in general affect whether a program can be learned or not, but how many examples must be provided for learning.
- *Show at least two examples of each feature that L-EBE should pick up on.* Common features can only be discovered with two or more examples of them.
- *When giving examples for a loop, show examples of one or more iterations.* L-EBE assumes that at least one iteration of a loop will be shown at all times, but examples with more than one iteration are necessary to recognize the loop.

3 Results

L-EBE has been implemented and integrated into the Emacs editor [Stallman, 1987] by Schuler [1993]. The learning system is a Scheme process that will discover a gap program (if there is one) from examples passed to it from Emacs. Furthermore, when a transformation is executed in Emacs the Scheme learner returns a regular expression for the gap pattern it has learned and this expression determines the text that the transformation will be applied to. Once input text for the transformation has been located, the Scheme learner returns its replacement string. Furthermore a transformation can be saved to and loaded from files by the learner.

The interface to the L-EBE learner is implemented in Emacs and it allows a user to: construct new programs by presenting examples, execute a learned program in Emacs, and save and load programs previously learned. The Emacs interface performs any communication with the Scheme learner by sending it: examples for which a transformation is to be learned, text to be transformed by the current program, or commands to reset, save or restore programs.

The main goal of L-EBE was to show that looping text transformations can be learned from just input/output behavior. Because there are no previous methods or theory for learning loops represented in text transformations, the goal was to find an approach to the problem. It was not the intent of this work to find a complete solution, nor the best one. Other subgoals for L-EBE were to learn text transformations that involve background knowledge, and to learn any transformations that EBE does.

Testing of L-EBE has involved running it on a variety of different examples. Throughout the process of designing and implementing the system, L-EBE was run successfully on many of the examples presented in Nix's PhD thesis [1983]. Other examples tested include all those listed in [Schüler, 1993], as well as many other examples designed to test various components of the loop learning system. Figure 1 shows the looping transformation that was learned when L-EBE was presented with the examples from Table 1.

L-EBE is a system that aims at helping users automate editing tasks in an interactive editing environment. Therefore both the number of examples that must be provided to teach a looping transformation and the running time of the system are important factors for success in any real setting. While a full theoretical evaluation of L-EBE is beyond the scope of this work, some comments can be made about example and running time complexity of the system.

Table 2 shows that it took 36 seconds to learn the transformation on a Sun Sparc 2 from the first three examples. While this is reasonable in terms of the examples necessary for learning, too much time is required for it to be useful in an interactive setting — especially as the time required for learning is expected to increase as more examples are presented. The timing pinpoints two inefficiencies in the design and implementation of loop learning in L-EBE:

- Before beginning to learn a replacement expression, the iterative gap patterns are separated out of the representation and then used individually in replacement synthesis. Most of the time spent on gap pattern synthesis is attributed to this splitting of iterative gap patterns.
- When the first three examples are presented to the system for learning, there is a tremendous difference in the time for learning loops in the different replacement machines. More than half of the time required to learn the entire program is spent in loop construction for the replacement machine of the first example. The first example, in comparison to the other two, is the only one where there is only one iteration of the author loop. For this example, the absolute outer bounds of the loop must be found, and this is where most of the time is spent in loop learning for the replacement expression.

4 Related work

TELS [Mo, 1989; Mo and Witten, 1990] learns editing tasks from examples of editing commands that transform input text. The command set is constrained to typing, selection, cut and paste, in order to ease the learning problem. In TELS a non–interactive phase processes the first example, finding loops by matching action subsequences, and inducing something similar to an annotated gap pattern for the loop body. During the interactive phase the learner asks for user confirmation on further examples, and the user may modify the learner's behaviour by correcting its predictions.

Shinohara's Data Entry by Example system [1982] learns a template for data entry from examples presented by the user. The templates learned are non–iterative gap

Component	Example 1	Examples 1,2		Examples 1,2,3		
LCS	0ms[a]	129ms		164ms		
Insert Gaps	1ms	23ms		24ms		
Initial Parse	2ms	38ms		34ms		
Input Loop Learning	4ms	43ms		4028ms		
Split Gap Patterns[b]	556ms	492ms		9760ms		
Gap Pattern Synthesis[c]	777ms	897ms		14199ms		
Construct Replacement Machine	1ms	58ms	72ms	45ms	64ms	48ms
Maximum Loop Region[d]	—	0ms	0ms	20776ms	0ms	0ms
Output Loop Learning	—	0ms	0ms	20792ms	21ms	167ms
Intersect Replacement Machines	0ms	9ms		176ms	46ms	
Replacement Expression Synthesis[c]	72ms	198ms		21559ms		
Transformation Synthesis[c]	1091ms	1291ms		35977ms		

[a]Less than 0.5ms.
[b]Separate all iterative gap patterns out of the representation for the set of iterative gap patterns.
[c]Note that timings for all components have not been included.
[d]Find the maximum possible region of a loop when there is only one iteration in the output example.

Table 2: Time for running the Examples from Table 1 in L-EBE

patterns. Once the template has been built from examples, the system can help the user by requesting input only for the variables in the template.

SVS [Baltes, 1991] learns disjunctive string concepts to be applied in learning operating system tasks in the UNIX domain. The system is presented with a sequence of examples, where the first example is considered positive and functions as a prototype of the concept to be learned. If the learner cannot classify one of the examples then the user acts as an oracle to provide classification.

VanLehn and Ball [1987] describe a learning method for context–free grammars that uses a variation of the version space representation. This solves two problems: there is an infinite set of grammars that are consistent with any finite set of examples so that the upper boundary in the version space may be infinite, and a partial order for grammars is undefined. By considering only *simple* and *reduced* grammars, the set of grammars consistent with any finite presentation is also finite, yet any context–free grammar can be generated with them. A *derivational version space* is defined, which is a superset of the reduced version space, yet has the property that a predicate can be found that partially orders its elements.

A number of authors have discussed approaches to synthesizing LISP functions from input/output examples (e.g. [Biermann, 1978; Biermann and Smith, 1979; Jouannaud and Guiho, 1977; Kodratoff, 1979; Summers, 1977]).

ETAR [Heise, 1989] is a robot task acquisition system that learns procedures by recording movements when a robot is led through a task by the user. ETAR makes use of a *focus of attention* mechanism, that determines which objects are important for different parts of a task, and thereby reduces the search-space for constructing the target procedure. In ETAR's loop induction, loops are constructed in two phases. In the *intra-example* processing phase each trace is grouped into sequences of actions that share the same focus of attention. Loops revolve around key groups (groups that involve key actions). The first key group marks the first iteration — further iterations of that loop are found by scanning for more key groups that have the same key event and the same focus–of–attention type (which depends on whether the objects in the focus of attention are explicit parameters to the task or not). Finally iterations are merged into a loop. The *inter-example* processing phase merges the traces from different examples to construct the final procedure. Traces are merged on a group by group basis which again takes the focus of attention into consideration. Special rules determine how loops are matched against other loops and subsequences of groups.

5 Conclusion

We have described the L-EBE system, which learns text rewrite rules containing loops, by learning a description of both the input pattern and how that pattern is transformed into the output. The work is an extension of the non–iterative EBE system built by Nix. Loop annotations are introduced to EBE's language, which describes a sequence of constant and variable substrings. Constraints on the examples ensure the detection of nested loops with different numbers of iterations. L-EBE has been implemented in Scheme and integrated with the Gnu Emacs editor, making it convenient for users to present examples to the learner, and also execute learned transformation rules during editing.

Constraints on the presentation of examples, and between the gap pattern and the replacement expression, present a working framework for learning text transformations with an arbitrary number of substring repetitions, from input/output examples. Learning systems must present an appropriately human-oriented environment to the user if repetitive text editing is to be automated.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada, and the University of Calgary.

References

[Angluin, 1979] Dana Angluin. Finding patterns common to a set of strings. In *Proceedings, 11th Annual Symposium on Theory of Computing*, pages 130–141, 1979.

[Angluin, 1980] Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

[Baltes, 1991] Jacky Baltes. A symmetric version space algorithm for learning disjunctive string concepts. In *Proc. Fourth UNB Artificial Intelligence Symposium*, pages 55–65, September 1991. [Also, unpublished, 1992, pp.1-33.].

[Biermann and Smith, 1979] Alan W. Biermann and Douglas R. Smith. A production rule mechanism for generating lisp code. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9(5):260–276, May 1979.

[Biermann, 1978] Alan W. Biermann. The inference of regular lisp programs from examples. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-8(8):585–600, August 1978.

[Heise, 1989] Rosanna Heise. Demonstration instead of programming: Focussing attention in robot task acquisistion. Master's Thesis 89/360/22, University of Calgary, September 1989.

[Hirshberg, 1975] David Hirshberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, June 1975.

[Jouannaud and Guiho, 1977] J. P. Jouannaud and G. Guiho. Inference of functions with an interactive system. *Machine Intelligence*, 1977.

[Kodratoff, 1979] Yves Kodratoff. A class of functions synthesized from a finite number of examples. *International Journal of Computer and Information Science*, 8(6), 1979.

[MacDonald, 1991] Bruce A. MacDonald. Instructable systems. *Knowledge Acquisition*, 3:381–420, December 1991.

[Mo and Witten, 1990] Dan H. Mo and Ian H. Witten. Learning text editing tasks from examples: a procedural approach. Technical Report 90/394/18, University of Calgary, 1990.

[Mo, 1989] Dan H. Mo. Learning text procedures from examples. Master's thesis, Department of Computer Science, University of Calgary, Dec 1989.

[Nix, 1983] Robert P. Nix. *Editing by Example.* PhD thesis, Yale University, October 1983.

[Nix, 1985] Robert P. Nix. Editing by example. *ACM Transactions on Programming Languages and Systems*, 7(4):600–621, October 1985.

[Schüler, 1993] Natascha O. Schüler. L-ebe: Learning iterative editing by example. Master's thesis, Unversity of Calgary, April 1993. In Preparation.

[Shinohara, 1982] Takeshi Shinohara. Polynomial inference of extended pattern languages. In *Proceedings of Software Science and Enginieering*, pages 115–127, 1982.

[Stallman, 1987] Richard Stallman. *GNU Emacs Manual. Version 18.* Free Software Foundation, sixth edition, March 1987.

[Summers, 1977] P. D. Summers. A methodology for lisp program construction from examples. *Journal of the ACM*, 24(1), 1977.

[VanLehn and Ball, 1987] Kurt VanLehn and William Ball. A version-space approach to learning context-free grammars. *Machine Learning*, 2(1):39–74, 1987.

A Concept-Based Knowledge Discovery Approach in Databases *

Xiaohua Hu‡, Nick Cercone‡, Jiawei Han+
‡Department of Computer Science, University of Regina
Regina, SK, Canada, S4S 0A2
+School of Computing Science, Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
e-mail: { xiaohua,nick } @cs.uregina.ca

Abstract

An efficient concept-based induction method is developed for knowledge discovery in databases. The method integrates the machine learning paradigm, especially learning-from-examples techniques, with set-oriented database operations and extracts generalized data from data in databases. An attribute-oriented concept tree ascension technique is applied in generalization which substantially reduces the computational complexity of the database learning process. With the assistance of knowledge about concept hierarchies, data relevance, and expected rule forms, different kinds of knowledge rules, including characteristic rules, equality rules and inheritance rules associated with concepts at different levels in the concepts hierarchies can be discovered efficiently. In this paper we focus our attention on the automatic discovery of knowledge associated with concept hierarchies in database.

1 Introduction

Knowledge Discovery is the extraction of implicit, useful information from data. Knowledge discovery from a database is a form of machine learning. The growth in the size and number of existing databases far exceeds human abilities to analyze the data; this situation creates both a need and an opportunity to extract knowledge from databases.

In this paper, we present a method for discovering knowledge in relational databases. Our method integrates a machine learning paradigm, especially *learning from example* techniques, with database operations and extracts generalized data from actual data in databases. A key to the approach is attribute-oriented concept tree ascension for generalization which applies well-developed set-oriented database operations and substantially reduces the computational complexity of the database learning process. Initially the data in the database is generalized to a prime relation table which contains all essential information about the data in the database. Then the prime table is further processed to learn different kinds of rules, including characteristics rules, equality rules and inheritance rules associated with concepts at different levels in the concept hierarchies.

This paper is organized as follows. Primitives for knowledge discovery in databases are introduced in Section 2. The principles and algorithms for knowledge discovery in databases are presented in Section 3. The test of our method using a set of animal data is illustrated in Section 4. The discussion and conclusion are presented in Section 5.

2 Primitives for Knowledge Discovery in Databases

Three primitives should be provided for the specification of a learning task: *task-relevant data*, *background knowledge*, and *the expected representations of the learning results*. For illustrative purposes, we only examine relational databases, however, the results can be generalized to other kinds of databases.

2.1 Data relevant to the discovery process

A database may store a large amount of data, of which only a portion may be relevant to a specific

*The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council, and the participation of PRECARN Associates Inc.

learning task. For example, to characterize the features of *mammal* in *animal*, only the data relevant to *mammal* in *animal* are appropriate to the learning process. Relevant data may extend over several relations. A query can be used to collect task-relevant data from the database. Task-relevant data can be viewed as examples for learning processes suggesting that, *learning-from-examples* should be an important strategy for knowledge discovery. Most *learning-from-examples* algorithms partition the set of examples into *positive* and *negative* sets and perform *generalization* using the positive data and *specialization* using the negative ones [Gietterich and Michalski, 1983]. Unfortunately, a relational database does not explicitly store negative data, and thus no explicitly specified negative examples can be used for specialization. Therefore, a database induction process relies primarily on generalization, which should be performed cautiously to avoid over-generalization.

When learning a characteristic rule, relevant data are collected into one class, the target class, for generalization. When learning an equality rule, it is necessary to analyze the relationship for the data value among the different attributes for the target class. When learning a discrimination rule, it is necessary to collect data into two classes, the target class and the contrasting class(es). The data in the contrasting class(es) imply that such data cannot be used to distinguish the target class from the contrasting one(s), that is, they are used to exclude the properties shared by both classes.

2.2 Background knowledge

Concept hierarchies represent necessary background knowledge which are used to control the generalization process. Different levels of concepts are often organized into a taxonomy of concepts. The concept taxonomy can be partially ordered according to a general-to-specific ordering. The most general concept is the null description (described by the reserved word "any"), and the most specific concepts correspond to the specific values of the attributes in the database [Cai et al, 1991]. Using a concept hierarchy, the rules learned can be represented in terms of generalized concepts and stated in a simple and explicit form, which is desirable to most users.

Concept hierarchies can be provided by knowledge engineers or domain experts. This provision is reasonable for even large databases since a concept tree registers only the *distinct* discrete attribute values or ranges of numerical values for an attribute which in general, are not very large and can be supplied by a domain expert.

2.3 Representation of learning results

From a logical point of view, each tuple in a relation is a logical formula in conjunctive normal form, and a data relation is characterized by a large set of disjunctions of such conjunctive forms. Thus, both the data for learning and the rules discovered can be represented in either relational form or in first-order predicate calculus.

3 Principles and Algorithms For Learning

A new method is presented which is based on the attributed-oriented concept ascension techniques discussed. The key to the approach is an attribute-oriented concept tree ascension technique for generalization. The general idea of basic attribute-oriented induction is one in which generalization is performed attribute by attribute using attribute removal and concept tree ascension. As a result, different tuples may be generalized to identical ones, and the final generalized relation may consist of only a small number of distinct tuples. In the procedure of generalization, the tuples in database are generalized to the desirable level, the table gained at this stage is called prime relation. The prime relation contains all of the essential information of the original data in the database. The prime relation is processed further to discover different kinds of rules associated with the concept hierarchy.

Definition 1 *An attribute is generalizable if there are a large number of distinct values in the relation but there exists a concept hierarchy for the attribute (i.e., there are higher level concepts which subsume these attribute values). Otherwise, it is nongeneralizable.*

Definition 2 *An attribute in a relatively large relation is desirable for consideration for generalization if the number of distinct values it contains does not exceed a user-specified desirability threshold (usually 6 or less).*

If an attribute is nongeneralizable, then it should be removed in the generalization process. Attribute removal corresponds to the generalization rule, *dropping conditions* [Michlski, 1983]. Consider a tuple as a set of conjuncts in the logical form; an attribute value, together with its attribute name, form one of the conjuncts. By removing a conjunct, a constraint is eliminated and the concept is generalized. If there are a

large set of distinct values for an attribute, the large set of values must be generalized. However, if there is no higher level concept provided for the attribute, it cannot be further generalized by ascending the concept tree. Therefore, the attribute should be eliminated in generalization. Attribute removal can also be viewed as a generalization of the attribute to the most general concept (ANY) and then removed from the representation.

If an attribute is generalizable, then it should be generalized to a higher level concept value by concept tree ascension techniques. Concept tree ascension corresponds to the generalization rule, *climbing generalization trees* [Michlski, 1983]. If there exists a higher level concept for the value in the concept tree, then the substitution of the value in each tuple in the relation by the corresponding higher level concept makes the tuple cover more cases than the original one, and thus it generalizes the tuple.

Our method is performed in 4 steps. First, a set of data relevant to the learning task is collected by a database query. Second, the collected data is then generalized by removal of nongeneralizable attributes and by performing concept-tree ascension (replacing lower-level attribute values in a relation using the concept hierarchy) on each generalizable attribute until the attribute becomes desirable (i.e., containing only a small number of distinct values). The identical generalized tuples in the relation are merged into one with a special internal attribute, *vote*, associated to register how many original tuples are generalized to this resultant tuple. The generalized relation obtained at this stage is called the *prime relation* and saved for later use. Third, we further simplify the generalized relation and map it into the feature table, then analyze the feature table and infer different kinds of rules. Finally, we examine the prime relation once more and infer the inheritance rules associated with the concept hierarchies.

A *prime relation* R_p for a set of data R stored in the relational table is an intermediate relation generalized from relation R by removing nongeneralizable attributes and generalizing each attribute to a *desirable level.* Let a *desirability threshold* be available for each attribute, which could be set by default or specified by the user or an expert, based on the semantics of the attributes and/or the expected forms of generalized rules. A prime relation maintains the relationship among generalized data in different attributes for a frequently inquired-of data set. It can be used for extraction of various kinds of generalized rules. The following algorithm extracts the prime relation R_p from a set of data R stored in relational table.

Algorithm 1 *Extraction of the prime relation from a set of data R*

Input: (i) A set of task-relevant data R (obtained by a relation query and are stored in a relation table), a relation of arity n with a set of attributes A_i ($1 \leq i \leq n$); (ii) a set of concept hierarchies, H_i, where H_i is a hierarchy on the generalized attribute A_i, if available; and (iii) a set of desirability thresholds T_i for each attribute A_i

Output. The prime relation R_p

Method

1. $R_t := R$; /* R_t is a temporary relation. */
2. **for** each attribute A_i ($1 \leq i \leq n$) of R_t **do** {
   if A_i is nongeneralizable **then** remove A_i;
   if A_i is not desirable but generalizable **then** generalize A_i to desirable level;

   /* Generalization is implemented as follows. Collect the distinct values in the relation and compute the lowest desirable level L on which the number of distinct values will be no more than T_i by synchronously ascending the concept hierarchy from these values. Generalize the attribute to this level L by substituting for each value A_i's with its corresponding concept H_i at level L. */
   }

   /* Identical tuples in the generalized relation R_t are merged
   with the number of identical tuples registered in *vote* */ .
3. $R_p := R_t$

For example, suppose we have an animal relation for some zoo as depicted in Table 1 and the concept hierarchy for the attribute "Animal" as depicted in Figure 1:

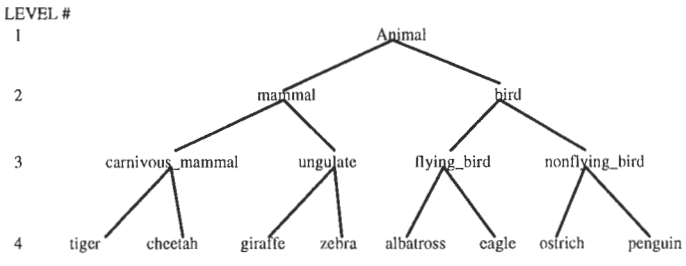


Figure 1: Conceptual Hierarchy of the Animal World

In the initial relation, the first attribute "Label" is the key to the relation, the key value is distinct for each

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S
1	tiger	Y	pt	fd	N	claw	meat	Y	N	Y
2	cheetah	Y	pt	fd	N	claw	meat	Y	N	Y
3	giraffe	Y	bt	sd	N	hoof	grass	Y	N	N
4	zebra	Y	bt	sd	N	hoof	grass	Y	N	N
5	ostrich	N	N	sd	Y	claw	grain	N	Y	N
6	penguin	N	N	sd	Y	web	fish	N	N	N
7	albotross	N	N	sd	Y	claw	grain	N	Y	Y
8	eagle	N	N	fd	Y	claw	meat	N	Y	N
9	viper	N	pt	fd	N	N	meat	N	N	N

Abbreviations: H: Hair, F: Feather, T: Teeth, S: Swim
pt:pointed, bt: blunted, fd:forward, sd:side.

Table 1: An Animal World.

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	V
1	cmammal	Y	pt	fd	N	claw	meat	Y	N	Y	2
2	ungulate	Y	bt	sd	N	hoof	grass	Y	N	N	2
3	nonfly	N	N	sd	Y	claw	grain	N	Y	N	1
4	nonfly	N	N	sd	Y	web	fish	N	N	N	1
5	flyingb	N	N	sd	Y	claw	grain	N	Y	Y	1
6	flyingb	N	N	fd	Y	claw	meat	N	Y	N	1
7	viper	N	pt	fd	N	N	meat	N	N	N	1

Abbreviations: H: Hair, F: Feather, T: Teeth, S: Swim
pt:pointed, bt: blunted, fd:forward, sd:side.

Table 2: The Prime Relation Table.

tuple in the relation . If there is no higher level concept provided for such an attribute in the concept tree, the value for the attribute cannot be generalized and it should be removed in the generalization process. Other candidate key attributes or nonkey attributes can be eliminated under a similar condition. The next attribute "Animal", has 9 distinct values, which is greater than the threshold value for our desirable level (assume the desirability threshold is 6), the concept-tree ascension technique is applied, applying algorithm 1; the attribute is generalized to the desirable level (level 3) $\{carnivorous\_mammal, ungulate, flying\_bird, nonflying\_bird\}$ in the conceptual hierarchy. We examine then the other attributes and since all of them are already at the desirable level, prime relation is obtained as shown in Table 2:

The derivation and storage of prime relations for frequently inquired-of data sets may facilitate the extraction of different kinds of generalized rules from the prime relation. Further generalization can be performed on prime relations to derive characteristic or inheritance rules if there are still many tuples in the prime relation. Based upon different interests, a generalized relation can be directly mapped into different feature tables. We have the following algorithm for the extraction of a feature table from a generalized relation.

Algorithm 2 *Feature table T_A extraction for an attribute A from the generalized relation R'.*

Input: A generalized relation R' consists of (i) an attribute A with distinct values a_1,...,a_m, m is the number of distinct values for A (ii) j other attributes $B_1, ..., B_j$, j is the number of attributes in the relation R' except A (suppose different attributes have unique distinct values), and (iii) a special attribute, *vote*.
Output. The feature table T_A
Method.

1. The feature table T_A consists of $m+1$ rows and $l+1$ columns, where l is the total number of distinct values in all the attributes. Each slot of the table is initialized to 0.

2. Each slot in T_A (except the last row) is filled by the following procedure,

   for each row r in R' **do** {
   for each attribute B_j in R' **do**
   $T_A[r.A, r.B_j] := T_A[r.A, r.B_j] + r.vote$;
   $T_A[r.A, vote] := T_A[r.A, vote] + r.vote$; }

3. The last row p in T_A is filled by the following procedure:

   for each column s in T_A **do**
   for each row t (except the last row p) in T_A **do**
   $T_A[p, s] := T_A[p, s] + T_A[t, s]$;

In our example, in order to obtain the feature table, the prime relation is further generalized by substituting the concept at level 3 by those at level 2, resulting in the generalized relation as shown in Table 3.

The feature table is then extracted from the generalized relation by using algorithm 2 based on the attribute "Animal" and the result is shown in Table 4 (since we are interested in learning for Animal). Different feature tables can be extracted from the generalized relation based on the interest in different attributes. The extracted feature table is useful for derivation of the relationships between the classification attribute and other attributes at a high level. For example, the generalized rule *All animals with hair are mammals* can be extracted from Table 4 based upon the fact the class *mammal* takes all the votes with *Hair* count.

We present two algorithms for discovering different kinds of rules, characteristic and equality, and inheritance from a from database system.

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	V
1	mammal	Y	pt	fd	N	claw	meat	Y	N	Y	2
2	mammal	Y	bt	sd	N	hoof	grass	Y	N	N	2
3	bird	N	N	sd	Y	claw	grain	N	Y	N	1
4	bird	N	N	sd	Y	web	fish	N	N	N	1
5	bird	N	N	sd	Y	claw	grain	N	Y	Y	1
6	bird	N	N	fd	Y	claw	meat	N	Y	N	1
7	other	N	pt	fd	N	N	meat	N	N	N	1

Abbreviations: H: Hair, F: Feather, T: Teeth, S: Swim
pt:pointed, bt: blunted, fd:forward, sd:side.

Table 3: A generalized relation.

Animal	Hair		Teeth			..	Feather		..	Swim		vote
	y	n	p	b	n	..	y	n	..	y	n	
mammal	4	0	2	2	0	..	0	4	..	4	0	4
bird	0	4	0	0	4	..	4	0	..	1	3	4
others	0	1	1	0	0	..	0	1	.	0	1	1
total	4	5	3	2	4	..	4	5	..	5	4	9

Abbreviation: p:pointed b:blunt.

Table 4: The feature table for the attribute animal.

Algorithm 3 *An attribute-oriented induction for discovering characteristic and equality rules associated with the concept hierarchy.*

Input: (i) the prime relation obtained by Algorithm 1 (ii) a concept hierarchy table. (iii) a threshold N for the total number of tuples in the final generalized relation
Output: A set of characteristic rules and equality rules.
Method.

1. Generalize the prime relation further by performing attribute-oriented concept ascension technique until the number of the tuples is equal or less than the threshold value N

2. Using feature-table extraction algorithm (Algorithm 2), extract a feature table T_A from the prime relation based upon a certain attribute A.

3. Assume that there are total I classes, i.e., there are I distinct values for attribute A, $A_1, \ldots, A_I$. Also, assume that there are J attributes: $C_1, \ldots, C_J$, for the data in the feature table. Using K_j to denote the number of distinct values for attribute J_j. According to the feature table, two probability values, $b_{i,j,k}$ and $c_{i,j,k}$, are associated with the k-th value ($k = 1, \ldots, K_j$) of the j-th attribute ($j = 1, \ldots, J$) in the i-th class ($i = 1, \ldots, I$). Notice that the number of tuples associated with the k-th value of the j-th attribute in the i-th class is denoted by $a_{i,j,k}$.

$$b_{i,j,k} = a_{i,j,k}/total.$$
$$c_{i,j,k} = a_{i,j,k}/vote.$$

where $b_{i,j,k}$ represents the probability of $a_{i,j,k}$ in the entire database and $c_{i,j,k}$ denotes the probability of $a_{i,j,k}$ in the i-th class.

4. Extract characteristic rules and equality rules based on the probability for each distinct value of every attribute in each class in the feature table T_A. This is performed as follows.

   for each class do {

   if $b_{i,j,k} = c_{i,j,k} = 1$

   then the following rule is inferred.

$$A_j = T_A[i,j,k] \quad \leftrightarrow \quad Class = C_i.$$

   if $b_{i,j,k} = 1$ *and* $c_{i,j,k} < 1$

   then the following rule is inferred.

$$A_j = T_A[i,j,k] \quad \rightarrow \quad Class = C_i.$$

   if $b_{i,j,k} < 1$ *and* $c_{i,j,k} = 1$

   then include $A_j = T_A[i,j,k]$ as a component for the corresponding characteristic rule for the i-th class.

   if $b_{i,j,k} \neq 1$ *and* $c_{i,j,k} \neq 1$ *and* $b_{i,j,k} * c_{i,j,k} \leq r_{frequency}$

   then ignore this value

   else include the value as one of the characteristic values for the attribute.

   /* Since data in a database may be distributed along the full spectrum of the possible values, it is impossible to obtain a meaningful rule for such kinds of data without using possible quantitative information. Various techniques can be developed for rule extraction using quantitative information. Our method treats data which occur rarely in the database as exceptional or noise data and filters it using $r_{frequency}$, where a small $r_{frequency}$ indicates that the data occurs with a very low frequency ratio. */ }.

5. Simplify the learned rules.

   If the distinct data value set of an attribute covers the entire set of values for the attribute, remove this attribute and its associated values from the

rule. Otherwise, compare the number of the values appearing as the characteristic values for the attribute with the total number of distinct values for the attribute. If the difference is larger than some pre-set number, the 'not' operator is introduced to the rules to simplify it.

6. Discover equality rules for different attributes based on the feature table.

   For each class C_i, for any two attributes j_1 and j_2 that relate the k_1-th value in the j_1-th attribute and k_2-th value in the j_2-th attribute, if $a_{i,j_1,k_1} = a_{i,j_2,k_2} = vote$, infer the following rule.

   $$A_{j_1} = T_A[i, j_1, k_1] \quad \leftrightarrow \quad A_{j_2} = T_A[i, j_2, k_2].$$

   * The next highest concept is the concept one level below the most generalized concept "*any*". □

Algorithm 4 *Attribute-oriented algorithm for discovering inheritance rules associated with concepts for different levels in the concept hierarchy.*

Input (i) the prime relation obtained by Algorithm 1, and (ii) the concept hierarchy tables. (iii) the attribute name ANAME (we intend to learn rules associated with the concept hierarchy for attribute ANAME)
Output A set of inheritance rules associated with concepts at different levels in the concept hierarchy of attribute ANAME.
Method.

1. Attach one class attribute to the prime relation (called E-attribute, E means extra).

2. Extract the concept hierarchy H for the attribute ANAME from the concept hierarchy tables

3. (Iterative Step) descend one level starting from the next highest generalized concept in the concept hierarchy H until reaching the desired level of the concept hierarchy. At each decent do the following:

   (a) Fill the E-attribute with the higher concept value and the corresponding attribute (attribute ANAME) with the concept value one level down of the E-attribute value in the concept hierarchy H.

   (b) Extract the related data, and store them in the temporary relation.

   (c) Project off the corresponding attributes which have the same values for all the low level concepts within the same higher concept from the temporary relation.

   (d) Find the inheritance rules: for each temporary relation, those remaining attributes which have different values for different lower level concepts but within the same higher concept category will be chosen as the component to form the corresponding inheritance rule. □

4 An Example

In this section, we use a data set from [Winston and Horn, 1984] to demonstrate algorithm 3 and algorithm 4. Given the animal world relation shown in Table 1 and the concept hierarchy for the attribute "Animal" depicted in Figure 1, Algorithm 3 is demonstrated as follows:

first step: Applying algorithm 1 to Table 1, results in the prime relation of Table 2. Next, further generalize Table 2 to the generalized relation as shown in Table 3.

second step: Extract the feature table based on the attribute "Animal" depicted in Table 4.

third step: Examine the values in the feature table; there are three classes for animal category mammal, bird and other. For $Class = mammal$ and $Hair = yes$, we have $a_{1,1,1} = 4$, $b_{1,1,1} = c_{1,1,1} = 1$ because $Class = mammal$ appears four times, and the total tuples for $Class = mammal$ is four. However $Hair = yes$ appears only four times in the entire table, so a rule can be inferred as follows:
$(Hair = yes) \leftrightarrow (Class = mammal)$.

similarly we obtain:
$(Milk = yes) \leftrightarrow (Class = mammal)$
$(Class = mammal) \rightarrow (Feet = claw \vee hoof) \wedge (Eats = meat \vee grass)$

for Class=bird:
$(Feather = yes) \leftrightarrow (Class = bird)$
$(Class = bird) \rightarrow (Feet = claw \vee web) \wedge (Eats = grain \vee fish \vee meat)$

fourth step: Simplify the above rules; count the number of values appearing as characteristic values for the attribute and compare them with the total number of distinct values for the attribute. If the difference is larger than some threshold (for example, 2) then the "not" operator is introduced to the rules to simplify the forms of the discovered rules.

Take the following rule as an example.
$(Class = bird) \rightarrow (Feet = claw \vee web) \wedge (Eats = grain \vee fish \vee meat)$

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	E
1	cmammal	Y	pt	fd	N	claw	meat	Y	N	Y	m
2	ungulate	Y	bt	sd	N	hoof	grass	Y	N	N	m
3	nonfly	N	N	sd	Y	claw	grain	N	Y	N	b
4	nonfly	N	N	sd	Y	web	fish	N	N	N	b
5	flyingb	N	N	sd	Y	claw	grain	N	Y	Y	b
6	flyingb	N	N	fd	Y	claw	meat	N	Y	N	b
7	viper	N	pt	fd	N	N	meat	N	N	N	o

Abbreviations: H:Hair, F:Feather, T:Teeth, S:Swim
m:mammal, b:bird, o:other, pt:pointed, bt: blunted, fd:forward, sd:side.

Table 5: A temporary relation after the substitution.

Since there are four distinct values: *meat, grass, grain* and *fish* for the attributes *Eats* and *Eats* takes three values out of four in the above rule, we can use $(Eats \neq grass)$ instead of $(eats = grain \vee fish \vee meat)$ as a component for this rule. Thus the rule is simplified as

$(Class = bird) \rightarrow (Feet \neq hoof) \wedge (Eats \neq grass)$

similarly, the rule:

$(Class = mammal) \rightarrow (Feet = claw \vee hoof) \wedge (Eats = meat \vee grass)$

can be simplified as

$(Class = mammal) \rightarrow not(Feet = web) \wedge (Eats = meat \vee grass)$

The last step is to analyze the data between different attributes and find the relationship between them to infer equality rules: for example, for Hair=yes, Feather=no,

$(Hair = yes) \leftrightarrow (Feather = No)$

$(Hair = yes) \leftrightarrow (Milk = yes)$

:

$(Feathers = yes) \leftrightarrow (Milk = No)$

Next we demonstrate the usefulness of Algorithm 4. The prime relation table is illustrated in Table 2 and the concept hierarchy for "Animal" is shown in Figure 1.

Attach the E_attribute to the Table 2 as shown as the right most column in Table 5, we do this by put the values of the next higher-level concept (level 2) in Figure 1 for attribute E and the corresponding animal value in level 3. For example, if the E attribute value is *mammal*, then the corresponding animal value in the animal attribute should be *carnivorous mammal* and *ungulate*, resulting in the temporary relation shown in Table 5:

From Table 5, the data related to *mammal* and *bird* are extracted, resulting in the temporary Tables 6 and

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	E
1	cmammal	Y	pt	fd	N	claw	meat	Y	N	Y	m
2	ungulate	Y	bt	sd	N	hoof	grass	Y	N	N	m

Abbreviations: H:Hair, F:Feather, T:Teeth, S:Swim
m:mammal, b:bird, o:other, pt:pointed, bt: blunted, fd:forward, sd:side.

Table 6: A temporary relation for mammal.

Animal	H	T	EY	F	Feet	Eat	M	Fly	S	E
nonflyb	N	N	sd	Y	claw	grain	N	N	N	b
nonflyb	N	N	sd	Y	web	fish	N	N	N	b
flyingb	N	N	sd	Y	claw	grain	N	Y	N	b
flyingb	N	N	fd	Y	claw	meat	N	Y	N	b

Abbreviations: H:Hair, F:Feather, T:Teeth, S:Swim
m:mammal, b:bird, o:other, pt:pointed, bt: blunted, fd:forward, sd:side.

Table 7: A temporary relation for bird.

7. Observe that *Hair, Feather, Milk, Fly* and *Swim* do not distinguish *mammals* but *Teeth, Eye, Eat and Feet* do distinguish *mammals* in Table 6. Thus the following rules are generated.

$(Class = mammal) \wedge (Teeth = pointed) \rightarrow (Animal = carnivorous\_mammal)$

$(Class = mammal) \wedge (Teeth = blunt) \rightarrow (Animal = ungulate)$

$(Class = mammal) \wedge (Eye = forward) \rightarrow (Animal = carnivorous\_mammal)$

$(Class = mammal) \wedge (Eye = side) \rightarrow (Animal = ungulate)$

$(Class = mammal) \wedge (Feet = claw) \rightarrow (Animal = carnivorous\_mammal)$

$(Class = mammal) \wedge (Feet = hoof) \rightarrow (Animal = ungulate)$

$(Class = mammal) \wedge (Eats = meat) \rightarrow (Animal = carnivorous\_mammal)$

$(Class = mammal) \wedge (Eats = grass) \rightarrow (Animal = ungulate)$

In a similar manner for bird, based on Table 7, we can derive the following rules:

$(Class = bird) \wedge (Fly = yes) \rightarrow (Animal = flying\_bird)$

$(Class = bird) \wedge (Fly = no) \rightarrow (Animal = nonflying\_bird)$

5 Discussion & Conclusion

A general framework has been presented for discovery many kinds of rules in databases. Our algorithms adopt the attribute-oriented conceptual ascension technique; attribute-oriented induction provides a simple and efficient way to learn different kinds of knowledge rules in relational databases. In general, our method adopts the concept tree ascending technique which follows the notion of *the version space method*, a typical method of *learning-from-examples* [Gietterich and Michalski, 1983]. However our method takes advantages of the organization of relational database systems.

The major benefit of our method when compared with the version space method is processing efficiency. The version space method adopts tuple-oriented generalization. In contrast, our method adopts attribute-oriented generalization which treats a concept hierarchy of each attribute as a factored version space and performs generalization on individual attributes. Factoring the version space can significantly improve the computational efficiency. Suppose there are p nodes in each concept tree and there are k concept trees (attributes) in thee relation. The total size of factorized version space should be p^k [Subrammanian and Feignbaum, 1986]. The search space for attribute-oriented generalization is much smaller than the one for tuple-oriented generalization.

We have extended our previous research for knowledge discovery in databases. Our approach applies an attribute-oriented concept tree ascension technique in generalization which integrates the machine learning methodology with set-oriented database operations and extracts generalized data from actual data in databases. Our method substantially reduces the computational complexity of the database learning processes. Different knowledge rules, including characteristic rules, equality rules, inheritance rules can be discovered efficiently using the attribute-oriented approach.

References

[Cai et al, 1991] Cai Y.D., Cercone N. and Han J, Attribute-Oriented Induction in Relational databases, *Knowledge Discovery in Database, AAAI/MIT Press*, G.Piatetsky-Shapiro and W.J. Frawley (eds) pp. 213-228, 1991

[Gietterich and Michalski, 1983] Dietterich T.G. and Michalski R. S, A Comparative Review of Selected Methods for Learning from Examples, *in Machine Learning: An Artificial Intelligence Approach, Vol. 1.* Michalski et. al. (eds), Morgan Kaufmann, pp 41-82, 1983.

[Frawlwy et al, 1991] Frawley W. J., Piatetsky G. and Matheus C. J, Knowledge Discovery in Database : An Overview, *Knowledge Discovery in Database, AAAI/MIT Press*, G.Piatetsky-Shapiro and W.J. Frawley (eds), pp. 1-27, 1991.

[Han et al, 1992] Han J, Cai Y. and Cercone N, Knowledge Discovery in Databases: An Attribute-Oriented Approach, *Proceeding of the 18th VLDB Conference*, Vancouver , B.C., Canada, pp 340-355, 1992

[Michlski, 1983] Mickalski R.S, A Theory and Methodology of Inductive Learning, *in Machine Learning: An Artificial Intelligence Approach, Vol. 1.* Michalski et. al. (eds), Morgan Kaufmann, 1983, pp 41-82.

[Subrammanian and Feignbaum, 1986] Subrammanian D and Feignbaum J, Factorization in experiment generalization, *Proc. 1986 AAAI Conf.* Philadelphia, PA, Aug. 1986, 518-522

[Winston and Horn, 1984] Winston P and Horn B.K,, *LISP*, Reading,Mass.: Addison _Wesley, 1984.

A Concept-Based Knowledge Discovery Approach in Databases *

Xiaohua Hu‡, Nick Cercone‡, Jiawei Han+
‡Department of Computer Science, University of Regina
Regina, SK, Canada, S4S 0A2
+School of Computing Science, Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
e-mail: { xiaohua,nick } @cs.uregina.ca

Abstract

An efficient concept-based induction method is developed for knowledge discovery in databases. The method integrates the machine learning paradigm, especially learning-from-examples techniques, with set-oriented database operations and extracts generalized data from data in databases. An attribute-oriented concept tree ascension technique is applied in generalization which substantially reduces the computational complexity of the database learning process. With the assistance of knowledge about concept hierarchies, data relevance, and expected rule forms, different kinds of knowledge rules, including characteristic rules, equality rules and inheritance rules associated with concepts at different levels in the concepts hierarchies can be discovered efficiently. In this paper we focus our attention on the automatic discovery of knowledge associated with concept hierarchies in database.

1 Introduction

Knowledge Discovery is the extraction of implicit, useful information from data. Knowledge discovery from a database is a form of machine learning. The growth in the size and number of existing databases far exceeds human abilities to analyze the data; this situation creates both a need and an opportunity to extract knowledge from databases.

In this paper, we present a method for discovering knowledge in relational databases. Our method integrates a machine learning paradigm, especially *learning from example* techniques, with database operations and extracts generalized data from actual data in databases. A key to the approach is attribute-oriented concept tree ascension for generalization which applies well-developed set-oriented database operations and substantially reduces the computational complexity of the database learning process. Initially the data in the database is generalized to a prime relation table which contains all essential information about the data in the database. Then the prime table is further processed to learn different kinds of rules, including characteristics rules, equality rules and inheritance rules associated with concepts at different levels in the concept hierarchies.

This paper is organized as follows. Primitives for knowledge discovery in databases are introduced in Section 2. The principles and algorithms for knowledge discovery in databases are presented in Section 3. The test of our method using a set of animal data is illustrated in Section 4. The discussion and conclusion are presented in Section 5.

2 Primitives for Knowledge Discovery in Databases

Three primitives should be provided for the specification of a learning task: *task-relevant data*, *background knowledge*, and *the expected representations of the learning results*. For illustrative purposes, we only examine relational databases, however, the results can be generalized to other kinds of databases.

2.1 Data relevant to the discovery process

A database may store a large amount of data, of which only a portion may be relevant to a specific

*The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council, and the participation of PRECARN Associates Inc.

learning task. For example, to characterize the features of *mammal* in *animal*, only the data relevant to *mammal* in *animal* are appropriate to the learning process. Relevant data may extend over several relations. A query can be used to collect task-relevant data from the database. Task-relevant data can be viewed as examples for learning processes suggesting that, *learning-from-examples* should be an important strategy for knowledge discovery. Most *learning-from-examples* algorithms partition the set of examples into *positive* and *negative* sets and perform *generalization* using the positive data and *specialization* using the negative ones [Gietterich and Michalski, 1983]. Unfortunately, a relational database does not explicitly store negative data, and thus no explicitly specified negative examples can be used for specialization. Therefore, a database induction process relies primarily on generalization, which should be performed cautiously to avoid over-generalization.

When learning a characteristic rule, relevant data are collected into one class, the target class, for generalization. When learning an equality rule, it is necessary to analyze the relationship for the data value among the different attributes for the target class. When learning a discrimination rule, it is necessary to collect data into two classes, the target class and the contrasting class(es). The data in the contrasting class(es) imply that such data cannot be used to distinguish the target class from the contrasting one(s), that is, they are used to exclude the properties shared by both classes.

2.2 Background knowledge

Concept hierarchies represent necessary background knowledge which are used to control the generalization process. Different levels of concepts are often organized into a taxonomy of concepts. The concept taxonomy can be partially ordered according to a general-to-specific ordering. The most general concept is the null description (described by the reserved word "any"), and the most specific concepts correspond to the specific values of the attributes in the database [Cai et al, 1991]. Using a concept hierarchy, the rules learned can be represented in terms of generalized concepts and stated in a simple and explicit form, which is desirable to most users.

Concept hierarchies can be provided by knowledge engineers or domain experts. This provision is reasonable for even large databases since a concept tree registers only the *distinct* discrete attribute values or ranges of numerical values for an attribute which in general, are not very large and can be supplied by a domain expert.

2.3 Representation of learning results

From a logical point of view, each tuple in a relation is a logical formula in conjunctive normal form, and a data relation is characterized by a large set of disjunctions of such conjunctive forms. Thus, both the data for learning and the rules discovered can be represented in either relational form or in first-order predicate calculus.

3 Principles and Algorithms For Learning

A new method is presented which is based on the attributed-oriented concept ascension techniques discussed. The key to the approach is an attribute-oriented concept tree ascension technique for generalization. The general idea of basic attribute-oriented induction is one in which generalization is performed attribute by attribute using attribute removal and concept tree ascension. As a result, different tuples may be generalized to identical ones, and the final generalized relation may consist of only a small number of distinct tuples. In the procedure of generalization, the tuples in database are generalized to the desirable level, the table gained at this stage is called prime relation. The prime relation contains all of the essential information of the original data in the database. The prime relation is processed further to discover different kinds of rules associated with the concept hierarchy.

Definition 1 *An attribute is generalizable if there are a large number of distinct values in the relation but there exists a concept hierarchy for the attribute (i.e., there are higher level concepts which subsume these attribute values). Otherwise, it is nongeneralizable.*

Definition 2 *An attribute in a relatively large relation is desirable for consideration for generalization if the number of distinct values it contains does not exceed a user-specified desirability threshold (usually 6 or less).*

If an attribute is nongeneralizable, then it should be removed in the generalization process. Attribute removal corresponds to the generalization rule, *dropping conditions* [Michlski, 1983]. Consider a tuple as a set of conjuncts in the logical form; an attribute value, together with its attribute name, form one of the conjuncts. By removing a conjunct, a constraint is eliminated and the concept is generalized. If there are a

large set of distinct values for an attribute, the large set of values must be generalized. However, if there is no higher level concept provided for the attribute, it cannot be further generalized by ascending the concept tree. Therefore, the attribute should be eliminated in generalization. Attribute removal can also be viewed as a generalization of the attribute to the most general concept (ANY) and then removed from the representation.

If an attribute is generalizable, then it should be generalized to a higher level concept value by concept tree ascension techniques. Concept tree ascension corresponds to the generalization rule, *climbing generalization trees* [Michlski, 1983]. If there exists a higher level concept for the value in the concept tree, then the substitution of the value in each tuple in the relation by the corresponding higher level concept makes the tuple cover more cases than the original one, and thus it generalizes the tuple.

Our method is performed in 4 steps. First, a set of data relevant to the learning task is collected by a database query. Second, the collected data is then generalized by removal of nongeneralizable attributes and by performing concept-tree ascension (replacing lower-level attribute values in a relation using the concept hierarchy) on each generalizable attribute until the attribute becomes desirable (i.e., containing only a small number of distinct values). The identical generalized tuples in the relation are merged into one with a special internal attribute, *vote*, associated to register how many original tuples are generalized to this resultant tuple. The generalized relation obtained at this stage is called the *prime relation* and saved for later use. Third, we further simplify the generalized relation and map it into the feature table, then analyze the feature table and infer different kinds of rules. Finally, we examine the prime relation once more and infer the inheritance rules associated with the concept hierarchies.

A *prime relation* R_p for a set of data R stored in the relational table is an intermediate relation generalized from relation R by removing nongeneralizable attributes and generalizing each attribute to a *desirable level.* Let a *desirability threshold* be available for each attribute, which could be set by default or specified by the user or an expert, based on the semantics of the attributes and/or the expected forms of generalized rules. A prime relation maintains the relationship among generalized data in different attributes for a frequently inquired-of data set. It can be used for extraction of various kinds of generalized rules. The following algorithm extracts the prime relation R_p from a set of data R stored in relational table.

Algorithm 1 *Extraction of the prime relation from a set of data R*

Input: (i) A set of task-relevant data R (obtained by a relation query and are stored in a relation table), a relation of arity n with a set of attributes A_i ($1 \leq i \leq n$); (ii) a set of concept hierarchies, H_i, where H_i is a hierarchy on the generalized attribute A_i, if available; and (iii) a set of desirability thresholds T_i for each attribute A_i

Output. The prime relation R_p

Method

1. $R_t := R$; /* R_t is a temporary relation. */
2. **for** each attribute A_i ($1 \leq i \leq n$) of R_t **do** {
   if A_i is nongeneralizable **then** remove A_i;
   if A_i is not desirable but generalizable **then** generalize A_i to desirable level;
   /* Generalization is implemented as follows. Collect the distinct values in the relation and compute the lowest desirable level L on which the number of distinct values will be no more than T_i by synchronously ascending the concept hierarchy from these values. Generalize the attribute to this level L by substituting for each value A_i's with its corresponding concept H_i at level L. */
   }
   /* Identical tuples in the generalized relation R_t are merged
   with the number of identical tuples registered in *vote* */ .
3. $R_p := R_t$

For example, suppose we have an animal relation for some zoo as depicted in Table 1 and the concept hierarchy for the attribute "Animal" as depicted in Figure 1:

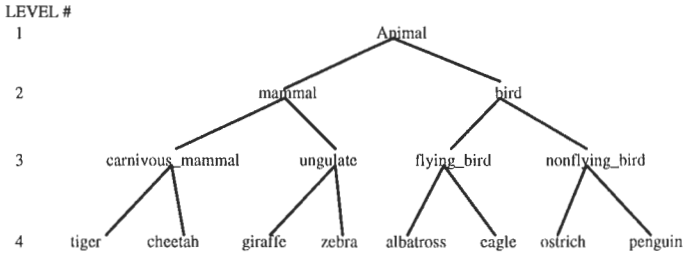


Figure 1: Conceptual Hierarchy of the Animal World

In the initial relation, the first attribute "Label" is the key to the relation, the key value is distinct for each

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S
1	tiger	Y	pt	fd	N	claw	meat	Y	N	Y
2	cheetah	Y	pt	fd	N	claw	meat	Y	N	Y
3	giraffe	Y	bt	sd	N	hoof	grass	Y	N	N
4	zebra	Y	bt	sd	N	hoof	grass	Y	N	N
5	ostrich	N	N	sd	Y	claw	grain	N	Y	N
6	penguin	N	N	sd	Y	web	fish	N	N	N
7	albotross	N	N	sd	Y	claw	grain	N	Y	Y
8	eagle	N	N	fd	Y	claw	meat	N	Y	N
9	viper	N	pt	fd	N	N	meat	N	N	N

Abbreviations: H: Hair, F: Feather, T: Teeth, S: Swim
pt:pointed, bt: blunted, fd:forward, sd:side.

Table 1: An Animal World.

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	V
1	cmammal	Y	pt	fd	N	claw	meat	Y	N	Y	2
2	ungulate	Y	bt	sd	N	hoof	grass	Y	N	N	2
3	nonfly	N	N	sd	Y	claw	grain	N	Y	N	1
4	nonfly	N	N	sd	Y	web	fish	N	N	N	1
5	flyingb	N	N	sd	Y	claw	grain	N	Y	Y	1
6	flyingb	N	N	fd	Y	claw	meat	N	Y	N	1
7	viper	N	pt	fd	N	N	meat	N	N	N	1

Abbreviations: H: Hair, F: Feather, T: Teeth, S: Swim
pt:pointed, bt: blunted, fd:forward, sd:side.

Table 2: The Prime Relation Table.

tuple in the relation . If there is no higher level concept provided for such an attribute in the concept tree, the value for the attribute cannot be generalized and it should be removed in the generalization process. Other candidate key attributes or nonkey attributes can be eliminated under a similar condition. The next attribute "Animal", has 9 distinct values, which is greater than the threshold value for our desirable level (assume the desirability threshold is 6), the concept-tree ascension technique is applied, applying algorithm 1; the attribute is generalized to the desirable level (level 3) $\{carnivorous\_mammal, ungulate, flying\_bird, nonflying\_bird\}$ in the conceptual hierarchy. We examine then the other attributes and since all of them are already at the desirable level, prime relation is obtained as shown in Table 2:

The derivation and storage of prime relations for frequently inquired-of data sets may facilitate the extraction of different kinds of generalized rules from the prime relation. Further generalization can be performed on prime relations to derive characteristic or inheritance rules if there are still many tuples in the prime relation. Based upon different interests, a generalized relation can be directly mapped into different feature tables. We have the following algorithm for the extraction of a feature table from a generalized relation.

Algorithm 2 *Feature table T_A extraction for an attribute A from the generalized relation R'.*

Input: A generalized relation R' consists of (i) an attribute A with distinct values $a_1,...,a_m$, m is the number of distinct values for A (ii) j other attributes $B_1, ..., B_j$, j is the number of attributes in the relation R' except A (suppose different attributes have unique distinct values), and (iii) a special attribute, *vote*.
Output. The feature table T_A
Method.

1. The feature table T_A consists of $m+1$ rows and $l+1$ columns, where l is the total number of distinct values in all the attributes. Each slot of the table is initialized to 0.

2. Each slot in T_A (except the last row) is filled by the following procedure,

   for each row r in R' **do** {
   for each attribute B_j in R' **do**
   $T_A[r.A, r.B_j] := T_A[r.A, r.B_j] + r.vote;$
   $T_A[r.A, vote] := T_A[r.A, vote] + r.vote;$ }

3. The last row p in T_A is filled by the following procedure:

   for each column s in T_A **do**
   for each row t (except the last row p) in T_A **do**
   $T_A[p, s] := T_A[p, s] + T_A[t, s];$

In our example, in order to obtain the feature table, the prime relation is further generalized by substituting the concept at level 3 by those at level 2, resulting in the generalized relation as shown in Table 3.

The feature table is then extracted from the generalized relation by using algorithm 2 based on the attribute "Animal" and the result is shown in Table 4 (since we are interested in learning for Animal). Different feature tables can be extracted from the generalized relation based on the interest in different attributes. The extracted feature table is useful for derivation of the relationships between the classification attribute and other attributes at a high level. For example, the generalized rule *All animals with hair are mammals* can be extracted from Table 4 based upon the fact the class *mammal* takes all the votes with *Hair* count.

We present two algorithms for discovering different kinds of rules, characteristic and equality, and inheritance from a from database system.

6	bird	N	N	fd	Y	claw	meat	N	Y	N	1
7	other	N	pt	fd	N	N	meat	N	N	N	1

Abbreviations: H: Hair, F: Feather, T: Teeth, S: Swim
pt:pointed, bt: blunted, fd:forward, sd:side.

Table 3: A generalized relation.

Animal	Hair		Teeth			..	Feather		..	Swim		vote
	y	n	p	b	n	..	y	n	..	y	n	
mammal	4	0	2	2	0	..	0	4	..	4	0	4
bird	0	4	0	0	4	..	4	0	..	1	3	4
others	0	1	1	0	0	..	0	1	.	0	1	1
total	4	5	3	2	4	..	4	5	..	5	4	9

Abbreviation: p:pointed b:blunt.

Table 4: The feature table for the attribute animal.

Algorithm 3 *An attribute-oriented induction for discovering characteristic and equality rules associated with the concept hierarchy.*

Input: (i) the prime relation obtained by Algorithm 1 (ii) a concept hierarchy table. (iii) a threshold N for the total number of tuples in the final generalized relation
Output: A set of characteristic rules and equality rules.
Method.

1. Generalize the prime relation further by performing attribute-oriented concept ascension technique until the number of the tuples is equal or less than the threshold value N

2. Using feature-table extraction algorithm (Algorithm 2), extract a feature table T_A from the prime relation based upon a certain attribute A.

3. Assume that there are total I classes, i.e., there are I distinct values for attribute A, $A_1, \ldots, A_I$. Also, assume that there are J attributes: $C_1, \ldots, C_J$, for the data in the feature table. Using K_j to denote the number of distinct values for attribute J_j. According to the feature table, two probability values, $b_{i,j,k}$ and $c_{i,j,k}$, are associated with the k-th value ($k = 1, \ldots, K_j$) of the j-th attribute ($j = 1, \ldots, J$) in the i-th class ($i = 1, \ldots, I$). Notice that the number of tuples associated with the k-th value of the j-th attribute in the i-th class is where $b_{i,j,k}$ represents the probability of $a_{i,j,k}$ in the entire database and $c_{i,j,k}$ denotes the probability of $a_{i,j,k}$ in the i-th class.

4. Extract characteristic rules and equality rules based on the probability for each distinct value of every attribute in each class in the feature table T_A. This is performed as follows.

   for each class do {

   if $b_{i,j,k} = c_{i,j,k} = 1$

   then the following rule is inferred.

   $$A_j = T_A[i,j,k] \quad \leftrightarrow \quad Class = C_i.$$

   if $b_{i,j,k} = 1$ *and* $c_{i,j,k} < 1$

   then the following rule is inferred.

   $$A_j = T_A[i,j,k] \quad \rightarrow \quad Class = C_i.$$

   if $b_{i,j,k} < 1$ *and* $c_{i,j,k} = 1$

   then include $A_j = T_A[i,j,k]$ as a component for the corresponding characteristic rule for the i-th class.

   if $b_{i,j,k} \neq 1$ *and* $c_{i,j,k} \neq 1$ *and* $b_{i,j,k} * c_{i,j,k} \leq r_{frequency}$

   then ignore this value

   else include the value as one of the characteristic values for the attribute.

   /* Since data in a database may be distributed along the full spectrum of the possible values, it is impossible to obtain a meaningful rule for such kinds of data without using possible quantitative information. Various techniques can be developed for rule extraction using quantitative information. Our method treats data which occur rarely in the database as exceptional or noise data and filters it using $r_{frequency}$, where a small $r_{frequency}$ indicates that the data occurs with a very low frequency ratio. */ }.

5. Simplify the learned rules.

   If the distinct data value set of an attribute covers the entire set of values for the attribute, remove this attribute and its associated values from the

rule. Otherwise, compare the number of the values appearing as the characteristic values for the attribute with the total number of distinct values for the attribute. If the difference is larger than some pre-set number, the 'not' operator is introduced to the rules to simplify it.

6. Discover equality rules for different attributes based on the feature table.

   For each class C_i, for any two attributes j_1 and j_2 that relate the k_1-th value in the j_1-th attribute and k_2-th value in the j_2-th attribute, if $a_{i,j_1,k_1} = a_{i,j_2,k_2} = vote$, infer the following rule.

   $$A_{j_1} = T_A[i, j_1, k_1] \quad \leftrightarrow \quad A_{j_2} = T_A[i, j_2, k_2].$$

   * The next highest concept is the concept one level below the most generalized concept "*any*". □

Algorithm 4 *Attribute-oriented algorithm for discovering inheritance rules associated with concepts for different levels in the concept hierarchy.*

Input (i) the prime relation obtained by Algorithm 1, and (ii) the concept hierarchy tables. (iii) the attribute name ANAME (we intend to learn rules associated with the concept hierarchy for attribute ANAME)
Output A set of inheritance rules associated with concepts at different levels in the concept hierarchy of attribute ANAME.
Method.

1. Attach one class attribute to the prime relation (called E-attribute, E means extra).
2. Extract the concept hierarchy H for the attribute ANAME from the concept hierarchy tables
3. (Iterative Step) descend one level starting from the next highest generalized concept in the concept hierarchy H until reaching the desired level of the concept hierarchy. At each decent do the following:
   (a) Fill the E-attribute with the higher concept value and the corresponding attribute (attribute ANAME) with the concept value one level down of the E-attribute value in the concept hierarchy H.
   (b) Extract the related data, and store them in the temporary relation.
   (c) Project off the corresponding attributes which have the same values for all the low level concepts within the same higher concept from the temporary relation.
   (d) Find the inheritance rules: for each temporary relation, those remaining attributes which have different values for different lower level concepts but within the same higher concept category will be chosen as the component to form the corresponding inheritance rule. □

4 An Example

In this section, we use a data set from [Winston and Horn, 1984] to demonstrate algorithm 3 and algorithm 4. Given the animal world relation shown in Table 1 and the concept hierarchy for the attribute "Animal" depicted in Figure 1, Algorithm 3 is demonstrated as follows:

first step: Applying algorithm 1 to Table 1, results in the prime relation of Table 2. Next, further generalize Table 2 to the generalized relation as shown in Table 3.

second step: Extract the feature table based on the attribute "Animal" depicted in Table 4.

third step: Examine the values in the feature table; there are three classes for animal category mammal, bird and other. For $Class = mammal$ and $Hair = yes$, we have $a_{1,1,1} = 4$, $b_{1,1,1} = c_{1,1,1} = 1$ because $Class = mammal$ appears four times, and the total tuples for $Class = mammal$ is four. However $Hair = yes$ appears only four times in the entire table, so a rule can be inferred as follows:
$(Hair = yes) \leftrightarrow (Class = mammal)$.

similarly we obtain:
$(Milk = yes) \leftrightarrow (Class = mammal)$
$(Class = mammal) \rightarrow (Feet = claw \vee hoof) \wedge (Eats = meat \vee grass)$

for Class=bird:
$(Feather = yes) \leftrightarrow (Class = bird)$
$(Class = bird) \rightarrow (Feet = claw \vee web) \wedge (Eats = grain \vee fish \vee meat)$

fourth step: Simplify the above rules; count the number of values appearing as characteristic values for the attribute and compare them with the total number of distinct values for the attribute. If the difference is larger than some threshold (for example, 2) then the "not" operator is introduced to the rules to simplify the forms of the discovered rules.

Take the following rule as an example.
$(Class = bird) \rightarrow (Feet = claw \vee web) \wedge (Eats = grain \vee fish \vee meat)$

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	E
1	cmammal	Y	pt	fd	N	claw	meat	Y	N	Y	m
2	ungulate	Y	bt	sd	N	hoof	grass	Y	N	N	m
3	nonfly	N	N	sd	Y	claw	grain	N	Y	N	b
4	nonfly	N	N	sd	Y	web	fish	N	N	N	b
5	flyingb	N	N	sd	Y	claw	grain	N	Y	Y	b
6	flyingb	N	N	fd	Y	claw	meat	N	Y	N	b
7	viper	N	pt	fd	N	N	meat	N	N	N	o

Abbreviations: H:Hair, F:Feather, T:Teeth, S:Swim
m:mammal, b:bird, o:other, pt:pointed, bt: blunted, fd:forward, sd:side.

Table 5: A temporary relation after the substitution.

Since there are four distinct values: *meat, grass, grain* and *fish* for the attributes *Eats* and *Eats* takes three values out of four in the above rule, we can use $(Eats \neq grass)$ instead of $(eats = grain \vee fish \vee meat)$ as a component for this rule. Thus the rule is simplified as

$(Class = bird) \rightarrow (Feet \neq hoof) \wedge (Eats \neq grass)$

similarly, the rule:

$(Class = mammal) \rightarrow (Feet = claw \vee hoof) \wedge (Eats = meat \vee grass)$

can be simplified as

$(Class = mammal) \rightarrow not(Feet = web) \wedge (Eats = meat \vee grass)$

The last step is to analyze the data between different attributes and find the relationship between them to infer equality rules: for example, for Hair=yes, Feather=no,

$(Hair = yes) \leftrightarrow (Feather = No)$
$(Hair = yes) \leftrightarrow (Milk = yes)$
:
$(Feathers = yes) \leftrightarrow (Milk = No)$

Next we demonstrate the usefulness of Algorithm 4. The prime relation table is illustrated in Table 2 and the concept hierarchy for "Animal" is shown in Figure 1.

Attach the E_attribute to the Table 2 as shown as the right most column in Table 5, we do this by put the values of the next higher-level concept (level 2) in Figure 1 for attribute E and the corresponding animal value in level 3. For example, if the E attribute value is *mammal*, then the corresponding animal value in the animal attribute should be *carnivorous mammal* and *ungulate*, resulting in the temporary relation shown in Table 5:

From Table 5, the data related to *mammal* and *bird* are extracted, resulting in the temporary Tables 6 and

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	E
1	cmammal	Y	pt	fd	N	claw	meat	Y	N	Y	m
2	ungulate	Y	bt	sd	N	hoof	grass	Y	N	N	m

Abbreviations: H:Hair, F:Feather, T:Teeth, S:Swim
m:mammal, b:bird, o:other, pt:pointed, bt: blunted, fd:forward, sd:side.

Table 6: A temporary relation for mammal.

Animal	H	T	EY	F	Feet	Eat	M	Fly	S	E
nonflyb	N	N	sd	Y	claw	grain	N	N	N	b
nonflyb	N	N	sd	Y	web	fish	N	N	N	b
flyingb	N	N	sd	Y	claw	grain	N	Y	N	b
flyingb	N	N	fd	Y	claw	meat	N	Y	N	b

Abbreviations: H:Hair, F:Feather, T:Teeth, S:Swim
m:mammal, b:bird, o:other, pt:pointed, bt: blunted, fd:forward, sd:side.

Table 7: A temporary relation for bird.

7. Observe that *Hair, Feather, Milk, Fly* and *Swim* do not distinguish *mammals* but *Teeth, Eye, Eat and Feet* do distinguish *mammals* in Table 6. Thus the following rules are generated.

$(Class = mammal) \wedge (Teeth = pointed) \rightarrow (Animal = carnivorous\_mammal)$
$(Class = mammal) \wedge (Teeth = blunt) \rightarrow (Animal = ungulate)$
$(Class = mammal) \wedge (Eye = forward) \rightarrow (Animal = carnivorous\_mammal)$
$(Class = mammal) \wedge (Eye = side) \rightarrow (Animal = ungulate)$
$(Class = mammal) \wedge (Feet = claw) \rightarrow (Animal = carnivorous\_mammal)$
$(Class = mammal) \wedge (Feet = hoof) \rightarrow (Animal = ungulate)$
$(Class = mammal) \wedge (Eats = meat) \rightarrow (Animal = carnivorous\_mammal)$
$(Class = mammal) \wedge (Eats = grass) \rightarrow (Animal = ungulate)$

In a similar manner for bird, based on Table 7, we can derive the following rules:

$(Class = bird) \wedge (Fly = yes) \rightarrow (Animal = flying\_bird)$
$(Class = bird) \wedge (Fly = no) \rightarrow (Animal = nonflying\_bird)$

5 Discussion & Conclusion

A general framework has been presented for discovery many kinds of rules in databases. Our algorithms adopt the attribute-oriented conceptual ascension technique; attribute-oriented induction provides a simple and efficient way to learn different kinds of knowledge rules in relational databases. In general, our method adopts the concept tree ascending technique which follows the notion of *the version space method*, a typical method of *learning-from-examples* [Gietterich and Michalski, 1983]. However our method takes advantages of the organization of relational database systems.

The major benefit of our method when compared with the version space method is processing efficiency. The version space method adopts tuple-oriented generalization. In contrast, our method adopts attribute-oriented generalization which treats a concept hierarchy of each attribute as a factored version space and performs generalization on individual attributes. Factoring the version space can significantly improve the computational efficiency. Suppose there are p nodes in each concept tree and there are k concept trees (attributes) in thee relation. The total size of factorized version space should be p^k [Subrammanian and Feignbaum, 1986]. The search space for attribute-oriented generalization is much smaller than the one for tuple-oriented generalization.

We have extended our previous research for knowledge discovery in databases. Our approach applies an attribute-oriented concept tree ascension technique in generalization which integrates the machine learning methodology with set-oriented database operations and extracts generalized data from actual data in databases. Our method substantially reduces the computational complexity of the database learning processes. Different knowledge rules, including characteristic rules, equality rules, inheritance rules can be discovered efficiently using the attribute-oriented approach.

References

[Cai et al, 1991] Cai Y.D., Cercone N. and Han J, Attribute-Oriented Induction in Relational databases, *Knowledge Discovery in Database, AAAI/MIT Press*, G.Piatetsky-Shapiro and W.J. Frawley (eds) pp. 213-228, 1991

[Gietterich and Michalski, 1983] Dietterich T.G. and Michalski R. S, A Comparative Review of Selected Methods for Learning from Examples, *in Machine Learning: An Artificial Intelligence Approach, Vol. 1.* Michalski et. al. (eds), Morgan Kaufmann, pp 41-82, 1983.

[Frawlwy et al, 1991] Frawley W. J., Piatetsky G. and Matheus C. J, Knowledge Discovery in Database : An Overview, *Knowledge Discovery in Database, AAAI/MIT Press*, G.Piatetsky-Shapiro and W.J. Frawley (eds), pp. 1-27, 1991.

[Han et al, 1992] Han J, Cai Y. and Cercone N, Knowledge Discovery in Databases: An Attribute-Oriented Approach, *Proceeding of the 18th VLDB Conference*, Vancouver , B.C., Canada, pp 340-355, 1992

[Michlski, 1983] Mickalski R.S, A Theory and Methodology of Inductive Learning, *in Machine Learning: An Artificial Intelligence Approach, Vol. 1.* Michalski et. al. (eds), Morgan Kaufmann, 1983, pp 41-82.

[Subrammanian and Feignbaum, 1986] Subrammanian D and Feignbaum J, Factorization in experiment generalization, *Proc. 1986 AAAI Conf.* Philadelphia, PA, Aug. 1986, 518-522

[Winston and Horn, 1984] Winston P and Horn B.K,, *LISP*, Reading,Mass.: Addison _Wesley, 1984.

Incorporating canonical discriminant attributes in classification learning

Simon P. Yip
Dept. of Computer Science
Swinburne University
Hawthorn 3122, Australia

Geoffrey I. Webb
School of Computing and Maths
Deakin University
Geelong 3217, Australia

Abstract

This paper describes a method for incorporating canonical discriminant attributes in classification machine learning. Though decision trees and rules have semantic appeal when building expert systems, the merits of discriminant analysis are well documented. For data sets on which discriminant analysis obtains significantly better predictive accuracy than symbolic machine learning, the incorporation of canonical discriminant attributes can benefit machine learning. The process starts by applying canonical discriminant analysis to the training set. The canonical discriminant attributes are included as additional attributes. The expanded data set is then subjected to machine learning. This enables linear combinations of numeric attributes to be incorporated in the classifiers that are learnt. Evaluation on the data sets on which discriminant analysis performs better than most machine learning systems, such as the Iris flowers and Waveform data sets, shows that incorporating the power of discriminant analysis in machine classification learning can significantly improve the predictive accuracy and reduce the complexity of classifiers induced by machine learning systems.

1. Introduction

Attribute-based or selective inductive classification learning algorithms aim to develop procedures capable of correctly classifying instances of disjoint classes. The condition parts of the classifiers are based on the values of attributes provided in the examples. These algorithms have not in general supported the derivation of conditions based on relationships between attributes. "It is obvious that if the class description is outside the description space that is defined in terms of available attributes or features, then it can only be learnt by extending that space. Indeed, it is possible that the relevant attributes or best features that could be used in the class description may not be explicit or included in the examples" [Elio and Watanabe, 1991]. The issue of constructing new attributes or features is closely related to constructive induction [e.g. Rendell and Seshu, 1990; Michalski, 1983a, Bloedorn and Michalski, 1991]. This paper describes methods of constructing new attributes by incorporating discriminant analysis.

Discriminant analysis is another popular classification method [e.g. Klecka, 1980]. There are two major types of discriminant analysis. Parametric methods assume normal distribution of the attributes while the nonparametric methods have no such assumption. Though discriminant analysis is a powerful classification method, unlike symbolic machine learning, the classifiers it develops do not have the semantic appeal of decision trees and rules. The latter offers modularized clearly explained formats for describing a decision procedure and are compatible with a human's reasoning procedures and expert system knowledge bases. Unlike parametric discriminant analysis, machine learning systems do not depend on the assumption that the attributes are normally distributed and uncorrelated. Previous research has shown that both symbolic machine learning and statistical techniques produce superior classifiers to those produced by the other on differing data sets [Weiss and Kapouleas, 1989; Holte, 1993; Breiman et al, 1984].

This paper describes techniques for incorporating parametric discriminant analysis in symbolic machine learning. Previous machine learning systems which attempt to incorporate parametric discriminant analysis include CART [Breiman et al., 1984] and LMDT (Linear machine decision tree) [Utgoff and Brodley, 1991]. In these systems, linear combinations of attributes are searched and evaluated before each node of a decision tree is created. In CART, which uses piecewise linear discriminants, the computation cost increases tremendously as the number of attributes and nodes increases. In LMDT, a complicated encoding and weight training system is implemented at each node. Another approach is used by SWAP1 [Weiss and Indurkhya, 1991] where discriminant functions are transformed to binary attributes. One binary attribute per class is added to the attribute space. Each binary attribute represents the classification result of a discriminant function. The system reports rules such as:

If (LD1 & (x > 109)) then class=1;

where x is a continuous attribute and *LD1* represents the condition that the instance is classified by a set of linear discriminant functions as class *1*. The use of such attributes greatly reduces the case with which the rule is comprehended. In the above approaches, the discriminant functions are based on the equation:

$$f_i(x) + ln(P(C_i)) > f_j(x) + ln(P(C_j)) \quad \forall \quad i \neq j$$

For each class C_i, $f_i(x)$, a linear function of the set of attributes, x, is derived. An unknown case is classified by applying the functions and choosing the class whose linear score is the largest. Another discriminant analysis technique is canonical discriminant analysis which is based on a different type of function. This paper reports methods of deriving and incorporating canonical discriminant attributes in classification learning.

2. Incorporating canonical discriminant analysis

Canonical discriminant analysis is a dimension-reduction technique related to principle component analysis and canonical correlation. [e.g. Klecka, 1980]. It derives combinations of attributes to maximise the difference of the centroid of different classes. This research investigates incorporating canonical discriminant analysis in inductive classification learning. A canonical discriminant function is a linear combination of the discriminating attributes. It has the following mathematical form:

$$f_{km} = u_o + u_1X_{1km} + u_2X_{2km} + ... + u_pX_{pkm}$$

where f_{km}=the value (score) on the canonical discriminant function for case m in the class k; X_{ikm}= the value on discriminant attribute X_i for case m in class k; u_i=coefficients which produce the desired characteristics in the function.

The maximum number of unique functions that can be derived is equal to the number of classes minus one or the number of attributes, whichever is fewer. The coefficients (the u's) for the each function are derived so as to maximise the distance between the class centroids. A class centroid is a imaginary point which has coordinates that are the class's mean on each of the attributes. In discriminant analysis, classification is a separate activity. The canonical discriminant functions can be used to predict the class to which an unseen case most likely belongs. Several classification procedures exist, but they all use the notion of comparing the case's position to each of the class centroids in order to locate the closest centroid. Since canonical functions aim to maximise the distance between class centroids, they can be utilised to transform the instance space (space containing training instances for learning) so as to maximise the linear separability of cases. As symbolic machine learning systems seek to develop linear partitions of the instance space, in this research, we incorporate the canonical function(s) as additional attribute(s) in the attribute space before submitting the expanded data set to inductive classification learning. Two classification learning systems are employed, C4.5 [Quinlan, 1993] and Einstein [Webb, 1992a]. C4.5, is decision tree based while Einstein, based on the algorithm DLG [Webb, 1992b], a variant of Aq [Michalski, 1983a], induces decision rules. To illustrate, suppose we have the following data:

X	Y	Z	Class
2	9	11	P
4	8	12	P
7	3	15	P
5	12	20	N
15	7	12	N
11	9	10	N
2	8	17	Q
3	10	15	Q
7	6	20	Q

With C4.5 [Quinlan, 1993], the decision tree induced is:

X > 7 : N (2.0)
X <= 7 :
| Z <= 15 : P (4.0/1.0)
| Z > 15 : Q (3.0/1.0)

With Einstein [Webb, 1992a], the rules induced are:

If (X <= 7 & Y <= 9 & Z <= 15) then class=P [3]
If (X >= 5.00 & Y >= 7.00) then class=N [3]
If (6<=Y<=10 & Z >= 15) then class=Q [3]

To incorporate canonical discriminant analysis, we can perform the following. By applying canonical discriminant analysis (available from most statistical packages such as SAS®, [1990]), canonical functions are derived. With three classes, two canonical discriminant functions are derived. The raw coefficients of the first canonical function for attributes *X*, *Y* and *Z* are 0.711, 0.903 and 0.226, respectively. Since the relative values of canonical attributes are the focus of classification, we can ignore the constant term in the canonical function. Thus, the value of the first canonical attribute (CAN1) for the first case is thus equal to: 2*0.711 + 9*0.903 + 11*0.226 = 12.04. Similarly, we can derive other canonical attribute values. The expanded data set is as follows:

X	Y	Z	CAN1	CAN2	class
2	9	11	12.04	3.28	P
4	8	12	12.78	3.45	P
7	3	15	11.08	3.95	P
5	12	20	18.91	5.74	N
15	7	12	19.70	3.13	N
11	9	10	18.21	2.79	N
2	8	17	12.49	4.83	Q
3	10	15	14.56	4.37	Q
7	6	20	14.92	5.42	Q

Submitting the expanded data set to a classification learning algorithm, the following concise trees or rules are derived:

With C4.5 [Quinlan, 1993], the decision tree induced is:

CAN1 > 14.92 : N (3.0)
CAN1 <= 14.92 :
| CAN2 <= 3.95 : P (3.0)
| CAN2 > 3.95 : Q (3.0)
With Einstein [Webb, 1992a], the rules are:
If (CAN1 <= 12.78 & CAN2 <= 3.95) then class=P [3]
If (CAN1 >= 18.21) then class=N [3]
If (CAN1 <= 14.92 & CAN2 >= 4.37) then class=Q [3]

From the theoretical perspective, incorporating canonical discriminant attributes is a form of empirical constructive induction. According to the framework for constructive induction developed by Rendell & Seshu, [1990], creating new attributes from existing attributes is termed feature construction. Feature construction can supplement the deficiency of selective induction in learning *hard* concepts. A concept is hard if its attributes have accurate class membership information but the concept cannot be learned by selective inductive methods. Hard concepts are characterised by dispersed and oddly shaped *peaks* in the instance space. Feature construction is the process of bringing together uniform regions that are dispersed in the instance space. Examination of the scatter plots on the above example supports the theoretical perspective.

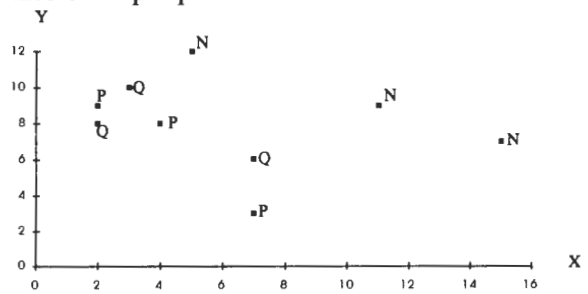


Figure 1: Scatter plot of X-Y

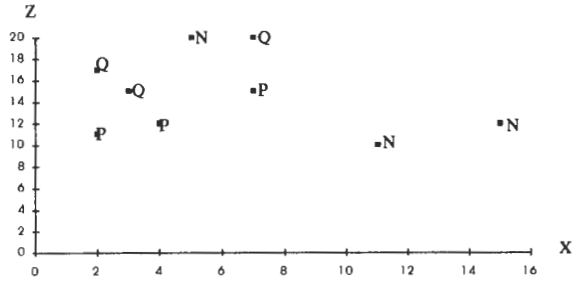


Figure 2: Scatter plot of X-Z

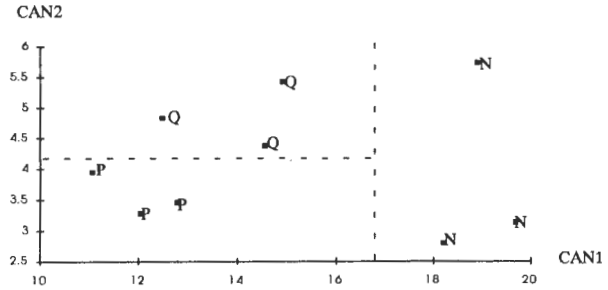


Figure 3: Scatter plot of canonical attributes (dotted lines showing simple decision surfaces)

In the scatter plots of (X-Y) and (X-Z), i.e. Fig.1 and 2, we can observe that the class members are dispersed. No simple decision surfaces can be found. In the scatter plot of the canonical attributes (Fig.3), the class members are grouped together and simple decision surfaces are easily found.

3. CAF (Canonical attribute finding)

We call the process of deriving and incorporating canonical attributes as CAF. The objective of the procedure is to find combinations of existing attributes that can contribute to the discrimination performance of existing attributes. When they are derived, each of the combinations is transformed into a single attribute and added to the attribute space. The application of CAF is indicated when the predictive accuracy of discriminant analysis for the domain is significantly higher than that obtained by the machine learning system under focus. The algorithm can be expressed as follows:

Algorithm: CAF
Input: a training set of examples
Output: an expanded training set of examples (*ET*)
Begin
    raw canonical coefficients ←canonical discriminant analysis on attributes;
    canonical attribute values ←(attribute values, raw canonical coefficients);
    ET ←Extend the descriptions of examples to include canonical attribute(s) as additional attribute(s);
End.

4. CCAF (Clustering before Canonical attribute finding)

Existing methods [e.g. Breiman et al., 1984; Utgoff & Brodley, 1991] for finding good attribute combinations involve search at each node when constructing the decision tree. Such methods involve high computation costs. If CAF is applied to subsets of data or at each node in building decision trees, the search cost at each node can be significantly reduced, but the computation cost to apply CAF for every node remains. CCAF is a method for tackling part of this problem. This method starts by applying clustering to re-classify the training set before deriving canonical attributes. It is useful when the possible partitions of the data are different from that given in the training examples. Two main categories of clustering methods exist: conceptual clustering [e.g. Michalski, 1983b] and cluster analysis [e.g. Everitt, 1980]. Since the objective is to derive canonical attributes, cluster analysis is used in this research. The common cluster analysis methods are based on agglomerative hierarchical clustering procedures. Each observation begins in a cluster by itself. Two clusters can

be merged to form a new cluster that replaces the two old clusters. Various clustering methods differ in how the distance between two clusters is computed. In this study, we use Ward's minimum-variance method.

By re-classifying a training set of examples into clusters before deriving canonical attributes, we can capture the partition information of clusters. CCAF is indicated when the predictive accuracy obtained by discriminant analysis for the domain is significantly higher than that of the machine learning system under focus. In this research, we set the maximum number of clusters to two times the number of different classes. The algorithm can be restated as follows:

Algorithm: CCAF
Input: a training set of examples
Output: an expanded training set of examples (*ET*)
Begin
    clusters ←cluster analysis on attributes;
    raw canonical coefficients ←canonical discriminant analysis on attributes with clusters as classes;
    canonical attribute values ←(attribute values, raw canonical coefficients)
    ET ←Extend the descriptions of examples to include canonical attribute(s) as additional attribute(s);
End.

5. Evaluation

Previous studies comparing discriminant analysis with classification machine learning have found that each approach outperforms the other on different sets of data. [Weiss & Kapouleas, 1989; Holte, 1993; Breiman et al, 1984]. Since we are interested in improving the performance of machine learning by incorporating canonical discriminant analysis, we select the ones on which discriminant analysis performs better. In this research, we use the Iris plants and Waveform data sets [Murphy & Aha, 1994]. The statistical package SAS® [1990] is used for canonical discriminant and cluster analysis. The machine learning systems used are C4.5 [Quinlan, 1993] and Einstein [Webb, 1992a].

5.1 Study 1 (Iris flower data)

In this study, we use the widely examined Iris flower data set with 150 examples. Each example consists of four numeric-valued attributes: sepal length, sepal width, petal length and petal width in centimetres. There are 3 classes of species: Iris setosa, Iris versicolor and Iris virginica. To enable comparison with other learning algorithms, this study uses the "leave-one-out" cross-validation method [e.g. Breiman et al., 1984]. The Chi-square test for correlated samples is used to compare predictive accuracy under different methods and the pair-wise t-test to compare complexity of induced classifiers. In the following tabulation of results, complexity refers to the number of rules or nodes in the classifier; CAF+C4.5, for example, represents the method of treating the data set with CAF before submitting to C4.5:

Method	Accuracy(%)	Complexity
(1) C4.5		
(pruned)	94.67	10.79
(rules)	95.3	4.02
(2) CAF+C4.5		
(pruned)	98	5
compare (1):	(χ^2=5; p≤.05)	(t=81.32; p≤.0005)
(rules)	96.67	3.96
compare (1):	(χ^2=1)	(t=2.77; p≤.025)
(3) CCAF+C4.5		
(pruned)	94	10.8
compare (1):	(χ^2=0.33)	(t=-0.38)
(rules)	94.67	4.05
compare (1):	(χ^2=0.33)	(t=-1.91)
(4) Einstein	96	7.03
(5) CAF+Einstein	96	5.98
compare (4):	(χ^2=0)	(t=36.21; p≤.0005)
(6) CCAF+Einstein	95.3	6.05
compare (4):	(χ^2=0.33)	(t=39.04; p≤.0005)

In the above tabulation, we observe that by deriving and adding canonical attributes in a data set with CAF, the performance of induced decision trees or rules can be significantly improved and the complexity significantly reduced. The results of applying CCAF are insignificant. The best result of incorporating canonical discriminant analysis can be compared to other methods that use leave-one-out evaluation design:

	Accuracy(%)
(5) CAF+C4.5(pruned) [this paper]	98
(6) Linear discriminant [this paper]	98
(7) Quadratic discriminant [this paper]	97.33
(8) Nearest neighbor, k=1 [this paper]	96.67
(9) CART [Weiss & Kapouleas, 1989]	95.3
(10) EACH [Salzberg, 1991]	95.3
(11) Neural net [Weiss & Kapouleas, 1989]	96.7
(12) PVM [Weiss & Kapouleas, 1989]	96.0
(13) SWAP1 [this paper]	97.33
(14) SWAP1+discriminant[1] [this paper]	96.67

In the above comparison to other methods, the predictive accuracy of C4.5 (pruned tree) is improved to equal that of linear discriminant analysis. The effect of CAF can be further examined by plotting the performance vs. training size graph. In this study, 20% of the data set is used as the evaluation set and the training set consists, in turns, of 40%, 60% or 80% of the data set. The performance of the induced trees or rules

[1] Represents SWAP1 with discriminant analysis option

of each training set is evaluated over 10 runs. The comparative predictive accuracy and complexity is illustrated in the following graphs:

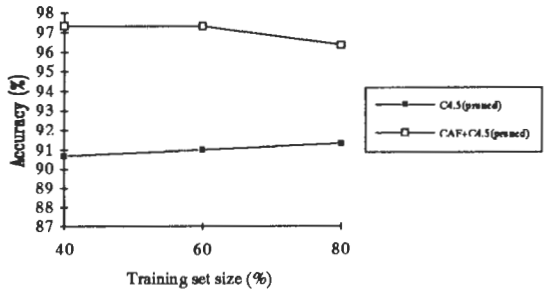


Figure 4: CAF+C4.5(pruned) vs. C4.5(pruned) Accuracy-Training_size-plot on Iris data

In Figure 4, the predictive accuracy of CAF+C4.5(pruned) is significantly better than that of C4.5(pruned) at all three training set sizes ($t_{40\%}$=6.71, p≤ .0005; $t_{60\%}$=6.05, p≤.0005; $t_{80\%}$=6.71, p≤.0005). In the above presentation, "$t_{40\%}$", for example, represents the t-value when the training set size equals 40% of the data set.

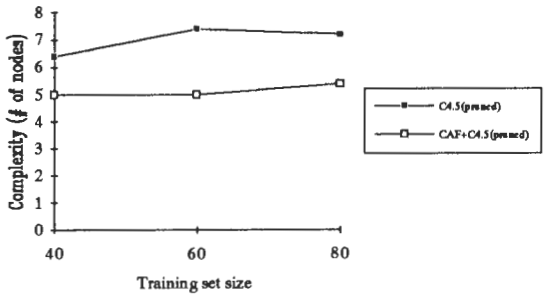


Figure 5: CAF+C4.5(pruned) vs. C4.5(pruned) Complexity-Training_size-plot on Iris data

Figure 5 shows that the complexity of pruned trees induced by CAF+C4.5 is significantly less than that of C4.5 alone, at all three training sizes ($t_{40\%}$=3.25, p≤.005; $t_{60\%}$=4.81, p≤.0005; $t_{80\%}$=4.58, p≤.005). The pattern of the performance vs. training size graphs of C4.5 rules is similar to that of C4.5 pruned trees.

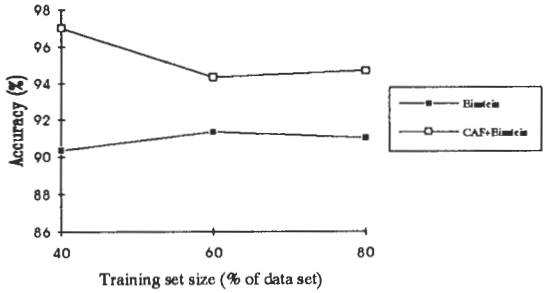


Figure 6: CAF+Einstein vs. Einstein Accuracy-Training_size-plot on Iris data

In Figure 6, we can observe that the predictive accuracy of CAF+Einstein is significantly better than that of Einstein alone at all three training set sizes ($t_{40\%}$=2.33, p ≤.025; $t_{60\%}$=2.59, p≤.025; $t_{80\%}$=4.71, p≤.005).

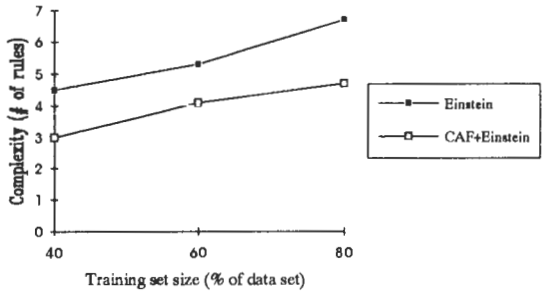


Figure 7: CAF+Einstein vs. Einstein Complexity-Training_size-plot on Iris data

Figure 7 shows that the complexity of rules learned by CAF+Einstein is significantly less that learned by Einstein alone at all three training sizes ($t_{40\%}$=7.75, p≤ .0005; $t_{60\%}$=2.45, p≤.025; $t_{80\%}$=7.75, p≤.0005).

In this study, we observe that by deriving and incorporating canonical discriminant attributes in machine learning, we can significantly improve the predictive accuracy and reduce the complexity of classifiers induced from various size of the training data. The predictive accuracy of CAF+C4.5(pruned) can be compared to that of other methods as follows:

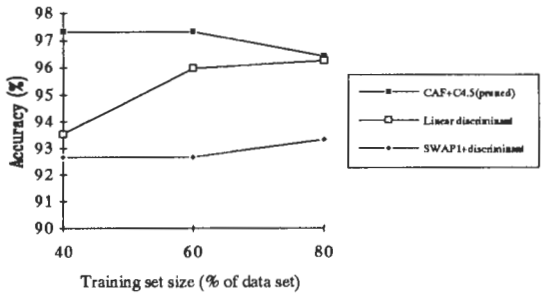


Figure 8: CAF+C4.5(pruned) vs. Linear discriminant vs. SWAP1+discriminant Accuracy-Training_size-plot on Iris data

Figure 8 shows that the predictive accuracy of CAF+C4.5(pruned) is better than that of Linear discriminant analysis ($t_{40\%}$=2.46, p≤.025; $t_{60\%}$=2.08; $t_{80\%}$=0.13) and SWAP1+discriminant ($t_{40\%}$=3.78, p≤ .005; $t_{60\%}$=3.5, p≤.005; $t_{80\%}$=5.02, p≤.0005). The performance of CAF+C4.5(pruned) at training set size of 40% is particularly notable.

5.2 Study 2

5.2.1 Waveform data set

In this study, we use the waveform data set used by the CART system [Breiman et al., 1984]. The data were generated with the program published in the UCI data

base [Murphy & Aha, 1994]. It is a three class problem based on the three waveforms $h_1(t)$, $h_2(t)$ and $h_3(t)$ graphed as follows:

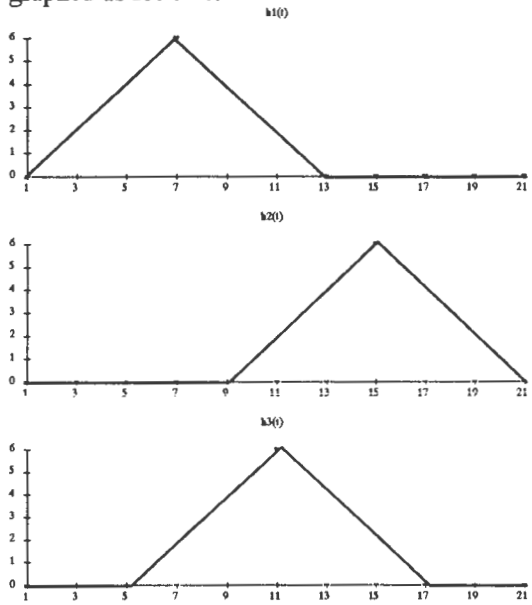


Figure 9: The three underlying waveforms

Each class consists of a random convex combination of two of these waveforms sampled at the integers with noise added. The measurement vectors are of 21 dimensions: $\mathbf{x}=(x_1,..x_{21})$. To generate each vector **x**, a uniform random number u and 21 random numbers $\varepsilon_1,....$ ε_{21} normally distributed with mean zero and variance 1 are generated:

For class 1 vectors, set: $x_m=uh_1(m) + (1-u)h_2(m) + \varepsilon_m$
For class 2 vectors, set: $x_m=uh_1(m) + (1-u)h_3(m) + \varepsilon_m$
For class 3 vectors, set: $x_m=uh_2(m) + (1-u)h_3(m) + \varepsilon_m$
where m=1,... 21

In order to compare performance with that of other studies, in this research, training sets of 300 examples using prior probabilities of (1/3, 1/3, 1/3) and a test data set of 5000 records are generated. The mean performance over 10 runs are as follows:

Method	Accuracy(%)	Complexity
(1) C4.5		
(pruned)	71.08	57.6
(rules)	70.44	14.9
(2) CAF+C4.5		
(pruned)	76.1	43.8
compare (1):	(t=8.15, p≤.0005)	(t=6.65, p≤.0005)
(rules)	76.58	10.3
compare (1):	(t=8.30; p≤.0005)	(t=2.9, p≤.01)
(3)CCAF+C4.5		
(pruned)	78.47	46.0
compare (1):	(t=11.05; p≤.0005)	(t=5.41, p≤.0005)
(rules)	78.91	11.6
compare (1):	(t=10.42; p≤.0005)	(t=2.41, p≤.025)
(4) Einstein	71.53	11.0
(5) CAF+Einstein	73.83	9.8
compare (4):	(t=6.23, p≤.0005)	(t=4.33, p≤.005)
(6) CCAF+Einstein	73.51	10.1
compare (4):	(t=4.5, p≤.005)	(t=3.0, p≤.01)

In the above tabulation, we observe that by incorporating canonical discriminant attributes in machine learning, the predictive performance can be significantly improved and the complexity of classifiers significantly reduced. The best result of incorporating canonical discriminant attributes in this study can be compared to other learning systems as follows:

Method	Accuracy(%)
(7) Linear discriminant [this paper]	80.72
(8) Quadratic discriminant [this paper]	78.45
(9) CCAF+C4.5 (pruned) [this paper]	78.47
CCAF+C4.5 (rules) [this paper]	78.91
(10) CART [Breiman et al., 1984]	72
(11) Nearest neighbor [Breiman et al., 1984]	78
(12) CART with 55 attributes added [Breiman et al., 1984]	80
(13) CART with linear combination [Breiman et al., 1984]	80
(14) SWAP1[2] [this paper]	73.06
(15) SWAP1+discriminant [this paper]	79.0

In method (12), the 55 new attributes added were based on the averages, $X_{m1,m2}$, over the attributes from m_1 to m_2 for odd values of m_1 & m_2, where

$$X_{m1,m2} = 1/(m_2-m_1+1) \sum_{m=m1}^{m2} X_m, \quad m_2 > m_1$$

In method (13), the linear attribute combination algorithm used by CART [Breiman et al, 1984] involves repetitive search for the best combination of attributes at each node to make up the best split when generating the classification tree. The computation cost of this method is high and the attribute evaluation function is system dependent. By using CCAF as an independent pre-machine learning step, the predictive performance of C4.5 rules is increased from 70.44% to 78.91%, and the classifier complexity is reduced from 14.9 rules to 11.6 rules. However, the accuracy performance is still less than that of linear discriminant analysis (t=2.99, p≤.01) and SWAP1+discriminant (t=0.14).

In this evaluation, we showed that by incorporating canonical discriminant analysis as a pre-symbolic classification learning step, the predictive accuracy and complexity of classifiers can be significantly improved when compared to classification learning alone.

[2] The data set, which contains noise, is offset by +4, because SWAP1 accepts only positive numbers.

5.2.2 Waveform data set with noise

In the evaluation of the CART system, waveform data sets containing the original 21 attributes plus 19 noise attributes were also used. In this study, we use a similar noisy data set generated by the published program of UCI database. Again, a training data set with 300 examples and a testing data set with 5000 cases were generated. Because of system limitations of Einstein and SWAP1, only C4.5 is used in this part. The results, based on 10 runs can be presented as follows:

Method	Accuracy(%)	Complexity
(1) C4.5		
(pruned)	68.63	58.0
(rules)	67.80	12.30
(2) CAF+C4.5		
(pruned)	73.35	38.60
compare (1):	(t=7.72, p≤.0005)	(t=12.93, p≤.0005)
(rules)	72.89	8.8
compare (1):	(t=7.02; p≤.0005)	(t=5.22, p≤.0005)
(3) CCAF+C4.5		
(pruned)	74.45	46.20
compare (1):	(t=7.3; p≤.0005)	(t=5.5, p≤.0005)
(rules)	75.06	12.0
compare (1):	(t=7.7; p≤.0005)	(t=0.2)

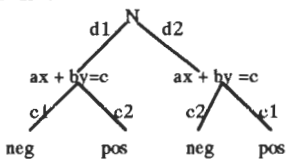
In the above tabulation, we also observe that by incorporating canonical discriminant attributes in machine learning, the predictive accuracy can be significantly improved and the complexity of classifiers significantly reduced. The best result of incorporating canonical discriminant attributes in this study can be compared to other learning systems as follows:

Method	Accuracy(%)
(4) CCAF+C4.5 (rules) [this paper]	75.06
(5) Linear discriminant analysis [this paper]	76.26
(6) Quadratic discriminant [this paper]	70.72
(7) CART [Breiman et al., 1984]	72
(8) Nearest neighbor [Breiman et al., 1984]	38

By using CCAF as an independent pre-machine learning step, the predictive accuracy of C4.5 rules is increased from 67.8% to 75.06% but still slightly less than that of linear discriminant analysis (t=1.65). In this study, very noisy data sets are used. Evaluation shows that by re-classifying the data with cluster analysis and deriving canonical discriminant attributes as additional attributes before submitting to classification learning, the predictive accuracy can be significantly improved and classifier complexity significantly reduced when compared to classification learning alone.

5.3 Study 3

Parametric discriminant analysis is quite robust to violation of the assumption that the attributes are normally distributed. The purpose of this study is to illustrate the incorporation of canonical discriminant analysis on mixed attributes and the merit of CCAF under certain conditions. In this study, we use an artificial data set generated with the following decision tree in mind:



In the above decision tree, N is a discrete attribute; x and y are continuous attributes; $d1$ and $d2$ are different discrete values; a, b, c, $c1$, and $c2$ are continuous values where $a \neq b$; $c1 \neq c2$. To generate an artificial set, we set $d1$=2, $d2$=3, a=3, b=2, $c1$=135 and $c2$=150. The class values are {pos, neg}. With the above decision tree in mind, we generated 400 cases, with 100 for each of the 4 leaves. For each case, x is assigned a random value between 1 and 40. With 10-fold cross validation, the performance of different methods can be presented as follows:

Method	Accuracy(%)	Complexity
(1) C4.5		
(pruned)	81.0	92.2
(rules)	84.0	37.9
(2) CAF+C4.5		
(pruned)	74.25	109.2
compare (1):	(t=-3.62, p≤.005)	(t=-2.58, p≤.025)
(rules)	75	40.8
compare (1):	(t=-4.19, p≤.005)	(t=-1.39)
(3) CCAF+C4.5		
(pruned)	98.75	8.2
compare (1):	(t=8.63, p≤.0005)	(t=19.82, p≤.0005)
(rules)	98.75	4.1
compare (1):	(t=7.17, p≤.0005)	(t=25.54, p≤.0005)
(4) Einstein	90.5	40.1
(5) CAF+Einstein	85.5	42.3
compare (4):	(t=-1.96)	(t=-1.99)
(6) CCAF+Einstein	100	4.4
compare (4):	(t=6.22, p≤.0005)	(t=52.2 p≤.0005)

	Accuracy(%)
(7) Linear discriminant	43.5
(8) Quadratic discriminant	100
(9) Nearest neighbor (k=1)	100
(10) SWAP1	96.5
(11) SWAP1+discriminant	55.75

As illustrated above, the use of CAF worsens the performance of classifiers obtained by the machine learning systems, but the use of CCAF leads to significant increases in predictive accuracy and decreases in classifier complexity. The improved performance is comparable to that of quadratic discriminant and nearest

neighbour methods. The incorporation of linear combinations of mixed attributes may reduce the semantic appeal of classifiers. Alternatively, linear combinations of continuous attributes only can be derived.

6. Discussion and Conclusion

In this paper, we presented two methods: CAF and CCAF, which incorporate the power of discriminant analysis into symbolic machine learning by deriving canonical discriminant attributes and adding them to the original attribute space. The expanded data set is then subjected to classification learning. Evaluation on data sets on which discriminant analysis performs better than most machine learning systems, shows that such techniques can significantly improve the performance of the machine learning systems. Linear combinations of attributes are derived with low search and computation costs. Stepwise discriminant analysis can also be used to reduce the number of terms in the linear combination. Alternatively, terms with coefficients close to zero can be discarded. Experiments on other data sets suggest that the better in accuracy performance of discriminant analysis over selective induction, the more significant is the positive effect on selective induction by incorporating canonical discriminant analysis.

In conclusion, discriminant analysis and symbolic inductive machine learning have been two important techniques in classification learning. Each has its own advantages and limitations. This paper demonstrates methods for combining these techniques. With a pre-machine learning step to derive and incorporate canonical discriminant attributes, we can significantly improve the predictive accuracy and decrease complexity of classifiers obtained by existing symbolic machine learning systems.

Acknowledgments

The authors wish to thank Ross Quinlan for C4.5 and Sholom Weiss for SWAP1.

References

[Bloedorn and Michalski, 1991] Bloedorn, E. and Michalski, R.S. (1991) Data driven constructive induction in AQ17-PRE A method and experiments. In *Proceedings of the third international conference on tools for AI*. San Jose, CA. 30-37.

[Breiman et al., 1984] Breiman, L., Friedman, J.H., Olshen, R.O. & Stone, C.J. (1984) *Classification and regression trees*. Wadsworth International Group, Belmont, California.

[Elio and Watanabe, 1991] Elio, R. & Watanabe, L. (1991) An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning*. 7, 7-44.

[Everitt, 1980] Everitt, B.S. (1980) *Cluster analysis*. 2nd edition, London: Heineman Educational Books.

[Holte, 1993] Holte, R.C. (1993) Very simple classification rules perform well on most commonly used data sets. *Machine learning*, 11, 1: 63-91.

[Klecka, 1980] Klecka, W.R. (1980) *Discriminant analysis*. Sage: Beverley Hills.

[Michalski, 1983a] Michalski, R.S. (1983) A theory and methodology of inductive learning. In Michalski, R.S., Carbonnel, J,G. & Mitchell, T.M. (Ed.) *Machine learning: an artificial intelligence approach*. Springer-Verlag.

[Michalski, 1983b] Michalski, R.S. (1983) Learning from observations: conceptual clustering. In Michalski, R.S., Carbonnel, J,G. & Mitchell, T.M. (Ed.) *Machine learning: an artificial intelligence approach*. Springer-Verlag.

[Murphy and Aha, 1994] Murphy, P.M. & Aha, D. (1994) UCI repository of machine learning databases. Irvine, CA: University of California, Dept. of information and Computer Science.

[Quinlan, 1993] Quinlan, R. (1993) *C4.5 Programs for machine learning*. Morgan Kaufmann.

[Rendell and Seshu, 1990] Rendell, L. & Seshu, R. (1990) Learning hard concepts through constructive induction: framework and rationale. *Computational intelligence*, 6: 247-270

[Salzberg, 1991] Salzberg, S. (1991) A nested hyper-rectangle learning method. *Machine learning*, 6:251-276.

[SAS® ,1990] SAS Institute Inc., Cary, NC: 27513

[Utgoff and Brodley, 1991] Utgoff, P.E. & Brodley, C.E. (1991) Linear machine decision trees. COINS Technical Report 91-10, University of Massachusetts, Amherst, MA.

[Webb, 1992a] Webb, G. (1992) Man-machine collaboration for knowledge acquisition. In *Proceedings of the 5th Australian joint conference on Artificial intelligence*. World Scientific, 329-334.

[Webb, 1992b] Webb, G. (1992) Learning disjunctive characteristic descriptions by least generalisation, *Technical report C92/9, Deakin University*, Geelong 3217

[Weiss and Kapouleas, 1989] Weiss, S.M. and Kapouleas, I. (1989) An empirical comparison of pattern recognition, neural nets and machine learning classification methods. In *Proceedings of the 11th international joint conference on artificial intelligence (IJCAI)* Detroit, MI : Morgan Kaufmann, 781-787.

[Weiss and Indurkhya, 1991] Weiss, S.M. and Indurkhya, N. (1991) Reduced complexity rule induction. In *Proceedings of the 12th international joint conference on artificial intelligence (IJCAI)*, Sydney: Morgan Kaufmann, 678-684.

Compiling Relational Data into Disjunctive Structure: Empirical Evaluation. *

Rina Dechter
Information and Computer Science
University of California, Irvine, CA 92717
dechter@ics.uci.edu

Eddie Schwalb
Information and Computer Science
University of California, Irvine, CA 92717
eschwalb@ics.uci.edu

Abstract

Recent work in knowledge compilation suggests that relations which can be described precisely by either *Horn theories* or *tree constraint networks* are identifiable in output polynomial time. Algorithms for computing approximations using these languages were also proposed. Upon testing such approximations on artificially generated and real life data, it was immediately observed that they yield numerous superfluous models. As a result, although certain entailment queries can be answered reliably, these methods may be ineffective for a large class of membership queries.

To improve the approximation quality, we investigate here the *k-decomposition problem*, that is, determining whether a relation can be described by a *disjunction* of k tractable theories. The paper discusses the complexity of this task, outlines several algorithms for computing both exact and approximate k-decompositions, and evaluates the potential of this approach empirically. We focus on the class of *tree constraint networks* and *Horn theories* and report results on artificially generated relations and on three real life cases. Our experiments show that for uniform random relations, the quality of upper bound approximations improves as k increases. However, when we require very high accuracy, decomposition is not effective since k grows linearly with the size of the data. When the data comes from a near-tractable source, the approach is useful. Experiments show that for the King Rook King problem the generalizing power of such methods is comparable to that of recently developed learning algorithms.

1 Introduction

Recently, frameworks for approximating intractable theories and relations using tractable languages were proposed using the notions of identifiability [2] and knowledge compilation [9]. The goal is to replace an intractable theory, or its set of models, with a tight upper and lower bound tractable language, thus allowing efficient query processing using the tractable approximations.

*This work was partially supported by NSF grant IRI-9157636, by Air Force Office of Scientific Research grant AFOSR 900136, by TOSHIBA of america and by Xerox grant.

In this paper, following [2] and in contrast to [9], we assume that the input theory is given by its set of models (or tuples) representing, perhaps, a set of observations and the task is to describe these observations using a tractable language. In [1, 2] it was shown that relations that can be described precisely by either *Horn theories* or *tree networks* can be identified in polynomial time. Otherwise, tight upper bound Horn theory or tree network approximations can be computed. In this paper we investigate empirically the effectiveness of such approximations on artificially generated relations as well as some real life data.

The effectiveness of an approximation should be measured with respect to a class of queries. There are two common types of queries on a theory φ: entailment queries (whether a formula holds in all models of φ), and membership queries (whether a given tuple is a model of φ). The first is common in automated reasoning while the second appears often in learning and classification tasks. Each query type dictates a different evaluation measure. We use, respectively, two measures: the fraction of clauses correctly entailed by the approximation, and the number of superfluous models in the approximation.

In our experiments it became immediately apparent that when the relations cannot be compiled into a language (as is often the case) the resulting tightest upper bound approximation is effective for certain entailment queries, but still yields numerous superfluous models.

To improve effectiveness relative to membership queries, we propose here to extend the model one step further by considering, as our target language, a *disjunction* of a fixed number of tractable theories. Specifically we address the following questions: Given a relation ρ and a class of theories Ω, can ρ be decomposed exactly into k subrelations, each represented by theories from Ω ? If not, does the approximation quality improve as the number of theories in the disjunction increases ? What are the complexity issues involved?

Our experiments show that for relations generated randomly and uniformly, the accuracy of an upper bound approximation improves significantly with the disjunction size. When we require the number of superfluous models to be small, decomposition is not effective because the number of theories required grows linearly with the size of the input relation. However, when the relation comes from a near-tractable source, the approach is useful.

The paper is organized as follows. Section 2 contains definitions and preliminaries, section 3 discusses the framework of k-disjunctive approximations and presents the

algorithms used, and section 4 presents our empirical results. Concluding remarks are given in section 5.

2 Definitions and Preliminaries

We denote propositional symbols, also called *variables*, by uppercase letters $P, Q, R, X, Y, Z, ...$, propositional literals (i.e., $P, \neg P$) by lowercase letters $p, q, r, x, y, z, ...$, and disjunctions of literals, or *clauses*, by $\alpha, \beta, ...$. A *formula* in conjunctive normal form (cnf) is a set of clauses $\varphi = \{\alpha_1, ..., \alpha_t\}$, implying their conjunction. The *models* of a formula φ, $M(\varphi)$, is the set of all satisfying truth assignments to all of the formula's symbols. A clause α is *entailed* by φ, written $\varphi \models \alpha$, iff α is true in all models of φ. A Horn formula is a cnf formula whose clauses all have at most one positive literal.

A *relation* associates a set of multivalued variables, also called attributes, with a set of tuples specifying their allowed combinations of values. A *constraint network* is a set of such relations, each defined on a subset of the variables. Taken together, this set represents a conjunction of constraints that restricts value assignments to comply with each and every constituent relation. The theory of relations has been studied extensively in the database literature [6].

Definition 1: (Relations and Networks)
Given a set of multivalued variables $X = \{X_1, ..., X_n\}$, each associated with a domain of discrete values $D_1, ..., D_n$ respectively, a *relation* (or, alternatively, a *constraint*) $\rho = \rho(X_1, ..., X_n)$ is any subset $\rho \subseteq D_1 \times D_2 \times ... \times D_n$. A *constraint network* N over X is a set $\rho_1, \ldots, \rho_t$ of relations each defined on a subset of variables $S_i \subseteq X$. Each relation ρ_i specifies the set of allowed assignments of the variables in S_i. A solution is an assignment of a value to each variable that satisfies all the constraints, and the network N represents the relation $rel(N)$ of all its solutions. If $rel(N) = \rho$ we say that N *describes* ρ. A constraint network in which all constraints involve pairs of variables ($|S_i| = 2$), is called a *binary constraint network*. A *constraint graph* associates each variable with a node and connects pair of variables that appear in the same constraint. *Tree networks* are binary networks whose constraint graph is a tree.

A cnf formula can be viewed as a special kind of constraint network, where the domains are bi-valued ($|D_i| = 2$) and each clause specifies a constraint on its propositional symbols. The set of models of the formula are the set of solutions of the corresponding constraint network. A bi-valued relation $\rho = \rho(x_1, ..., x_n)$ is *described* by a cnf formula $\varphi = \varphi(x_1, ..., x_n)$ iff $M(\varphi) = \rho$. We will use the term *theory* to denote either a network or a propositional formula.

Frequently, a relation cannot be precisely described by a theory from a given language, in which case we use approximations. We will examine primarily upper bound approximations.

Definition 2: (Upper bounds)
Given a class of theories, Ω, a theory $T \in \Omega$ is said to be an *upper bound* of ρ relative to Ω if $\rho \subseteq M(T)$. T is a tightest upper bound if $\rho \subseteq M(T)$ and there is no $T' \in \Omega$ such that $\rho \subseteq M(T') \subset M(T)$.

Clearly, if φ is a theory describing ρ ($M(\varphi) = \rho$), and T is an upper bound of φ, then if $T \models \gamma$ we can infer $\varphi \models \gamma$. Alternatively, if $t \notin M(T)$ then $t \notin \rho$ $(\rho \subseteq M(T))$.

Some languages admit a unique *tightest upper bound*. In this case, $U_\Omega(\rho)$ denotes the unique relation associated with this *tightest upper bound* expressed within this language. It is known that *Horn theories* allow a *unique* tightest upper bound [1, 2] while there may be many *tight upper bound tree networks* [2].

In [2, 3] it is shown that Horn theories and tree networks are identifiable, namely there is a polynomial algorithm that can decide whether any given relation can be described precisely by a Horn theory or a tree-network, and also finds the corresponding description whenever possible. Otherwise, the algorithm computes an upper bound. For Horn theories the algorithm generates the tightest Horn upper-bound, but it is no longer polynomial in the input relation. For tree-networks the algorithm is always polynomial but does not necessarily generate the tightest upper bound. For completeness sake, we present the algorithms for computing tight upper bounds for tree constraint networks and Horn theories.

2.1 Computing a tight tree network

The tree algorithm [2], finds a tree-network representation to a given relation, if such exists, otherwise it computes a tight upper bound.

Given an arbitrary relation, ρ, let $n(x_i)$ be the number of tuples in ρ for which $X_i = x_i$, and let $n(x_i, x_j)$ be the number of tuples for which $X_i = x_i \wedge X_j = x_j$. Let us define weights $w(X_i, X_j)$ as

$$w(X_i, X_j) = \frac{1}{|\rho|} \sum_{(x_i, x_j) \in \prod_{X_i X_j}(\rho)} n(x_i, x_j) log \frac{n(x_i, x_j)}{n(x_i) n(x_j)}$$

The constraint graph of the tree approximation is computed as the maximum weight spanning tree formed with the arc-weights $w(X_i, X_j)$. Once the structure of the tree is determined, the constraints of the network can be obtained by projecting ρ onto the pairs of connected variables in the tree.

2.2 Computing the tightest Horn upper bound

In [1, 2] it was shown that the models of a Horn theory are closed under intersection when intersection is defined as follows. Let $x = \{x_1, x_2, \ldots, x_n\}$ be a tuple where $x_i \in \{0, 1\}$. Then $true(x)$ is the set of variables assigned to 1 and $false(x)$ is the set of variables assigned to 0. The intersection $z = x \cap y$ is defined as $true(z) = true(x) \cap true(y)$ and $false(z) = false(x) \cup false(y)$. A bi-valued relation is said to be *closed under intersection* iff $\forall x, y \in \rho \quad x \cap y \in \rho$. The closure of ρ is the set of models of the tightest Horn theory bounding ρ.

We compute the set of models of the *tightest Horn upper bound* of a given relation by computing its intersection closure. The procedure is polynomial in the size of the output relation but not necessarily polynomial in the size of its input. Once the set of its models is computed, the

Horn theory can be extracted by algorithms presented in [1, 2].

3 Computing k-decompositions

We now extend the notion of identifiability to a disjunction of theories. We will assume (except when otherwise noted) throughout this paper that our languages admit a unique upper bound.

Definition 3: A relation ρ is k-decomposable relative to a class of theories Ω iff there exist a set of relations $Q = \{\rho_1, \ldots, \rho_k\}$ such that ρ_i is described in Ω and $\rho = \bigcup_{i=1}^{k} \rho_i$. A language, Ω, is k-identifiable if for every relation ρ, deciding if ρ is k-decomposable relative to Ω is polynomial.

Clearly, for any language that can describe a single tuple, every theory is k-decomposable for $k = |\rho|$. The interesting task is to find the smallest k for which a theory is *k-decomposable*. The following paragraphs provide the necessary and sufficient conditions for k-decomposability.

Definition 4: Let $U_\Omega(\rho)$ be the unique tightest upper bound of ρ relative to Ω. We define a graph $G_\Omega(\rho)$ as follows. Each *tuple*, $x \in \rho$, is mapped to a *node* and an *arc* between two nodes $x, y \in \rho$ exists iff $U_\Omega(\{x, y\}) \not\subseteq \rho$.

Theorem 1: *Given ρ and Ω,*

1. *If $G_\Omega(\rho)$ is not k-colorable then ρ is not k-decomposable.*
2. *If $G_\Omega(\rho)$ is k-colorable then ρ is k-decomposable iff there exists a k-coloring of $x_1 \ldots x_{|\rho|}$ (a value of $1 \ldots k$ assigned for every tuple in ρ) for which $\forall i \leq k$ the sets $\rho'_i = \{x \mid color(x) = i\}$ satisfy $U_\Omega(\rho'_i) \subseteq \rho$.*

Consequently, a lower bound on k is the size of each clique in G_Ω.

Theorem 1 suggests a brute force algorithm for computing a k-decomposition. Enumerate all k-colorings for $G_\Omega(\rho)$, and, for each coloring, check whether condition (2) is satisfied. If condition 2 can be tested in polynomial time (true for Horn theories), then the algorithm's complexity is dominated by the complexity of enumerating all k-colorings of a graph. Since finding even one coloring is NP-complete, the problem is clearly intractable. However, for the special case of $k = 2$, enumerating all possible colorings can be done in time linear in the number of colorings [4]. Moreover, for $k = 2$, it can be shown that every connected component of $G_\Omega(\rho)$ (bi-partite for $k = 2$) can be colored in at most two ways[1] and, therefore, $2^{\#components}$ possible colorings need to be checked.

Corollary 1: *Given a language Ω such that G_Ω can be computed in polynomial time, then 2-decomposability can be decided in time polynomial in $2^{\#components}$ of $G_\Omega(\rho)$.*

3.1 Approximated Decomposition

Because computing a k-decomposition is a difficult task, we examine polynomial approximation algorithms for two related formulation of this problem: (1) (minimization) given a theory φ and a language Ω, find the minimal k for which φ is *k-decomposable* relative to Ω. (2) (upper bound decomposition) given Ω and k, find a k-disjunctive upper-bound of φ relative to Ω that minimizes the number of superfluous models.

For the first task, we describe a greedy approximation algorithm. The algorithm can be viewed as a variant of the algorithm suggested by theorem 1. It starts from two arbitrary models $x, y \in \rho$, and computes $U_\Omega(\{x, y\})$. If $U_\Omega(\{x, y\}) \not\subseteq \rho$ it concludes that x and y must participate in different relations; otherwise, the algorithm adds $U_\Omega(\{x, y\})$ to ρ_i and deletes $U_\Omega(\{x, y\})$ from ρ. The algorithm continues with a third and fourth tuple, until all models in ρ are covered (see Figure 1).

Lemma 1: *Algorithm GreedyDecompose (Figure 1) terminates in $O(|\rho| \cdot k \cdot t_\rho)$ steps where k is the number of resulting theories and t_ρ is the number of steps required to test whether $U_\Omega(\rho) \subseteq \rho$.*

In the second task, the disjunction size k, is fixed in advance. The *partitioning* algorithm for that task, divides the input relation ρ into k equal partitions, $\rho_1, \ldots, \rho_k$, and outputs their tightest upper bounds, $U_\Omega(\rho_1), \ldots, U_\Omega(\rho_k)$. Clearly, $\rho \subseteq \cup_i U_\Omega(\rho_i) \subseteq U_\Omega(\rho)$. The complexity of partitioning is $O(k \cdot t_\rho)$ where t_ρ is the time required to compute $U_\Omega(\rho)$. Note that for Horn theories the greedy algorithm is always polynomial while the partitioning algorithm can be exponential. Note also that the partitioning algorithm has no control over the number of superfluous models.

To control the number of superfluous models we define ε approximations and show how such approximations can be computed by the greedy algorithm.

Definition 5: ((k, ε)-approximations) A relation ρ is *(k, ε)-decomposable* relative to Ω iff there exists a set of relations $Q = \{\rho_1, \rho_2 \ldots \rho_k\}$ such that ρ_i is described in Ω, and,

$$\rho \subseteq \bigcup_{i=1}^{k} \rho_i \qquad \text{and} \qquad \left| \bigcup_{i=1}^{k} \rho_i - \rho \right| \leq \varepsilon |\rho| \tag{1}$$

Q is called a (k, ε) upper bound.

A (k, ε) upper bound can be computed by *GreedyDecompose* if we allow only a bounded number of models in each subrelation ρ_i to fall outside the input relation, namely $|\rho_i - \rho| \leq \frac{\varepsilon}{k'} |\rho|$, and if the actual disjunction size generated (k) happens to be smaller than k'. This can be implemented by modifying line 10 of *GreedyDecompose* (Figure 1) to accommodate some models not in ρ (e.g. "**if** $|U_\Omega(\rho_i \cup \{x\}) - \rho| \leq \frac{\varepsilon}{k'} |\rho|$ **then**...").

4 Experiments

In this section we evaluate the quality of the approximation obtained. We use two measures: the number of superfluous models divided by the whole tuple space (2^n) and the fraction of clausal entailment queries answered correctly.

Notice that the unique tightest upper bound with respect a class Ω is guaranteed to correctly answer entailment

[1] We thank Dan Roth for this observation.

```
GreedyDecompose
1.  Input(ρ,  a class of theories Ω  and  a polynomial algorithm to compute U_Ω(ρ))
2.  Output(Q = {ρ_1, ..., ρ_k})          ; The disjunction of the relations - see Definition 3
3.  Begin
4.      Q ← {}                           ; Initialize an empty disjunction.
5.      ρ' ← ρ                           ; We use ρ' to preserve ρ for comparisons.
6.      while ρ' ≠ {} do
7.          choose arbitrary x ∈ ρ'
8.          flag ← false                 ; flag detects whether this tuple requires a new relation ρ_i
9.          if Q ≠ {} then
10.             foreach ρ_i ∈ Q do
11.                 if U_Ω(ρ_i ∪ {x}) ⊆ ρ then     ; If we can add the tuple to ρ_i, then
12.                     ρ_i ← U_Ω(ρ_i ∪ {x})       ; add it, and
13.                     ρ' ← ρ' − ρ_i              ; don't iterate on tuples already in ρ_i,
14.                     flag ← true                ; and signal that no new relation ρ_i is needed.
15.                 end-if
16.             end-for
17.         end-if
18.         if flag = false then         ; Here, we add to the disjunction Q
19.             Q ← Q ∪ {x}              ; a new relation which consists of
20.             ρ' ← ρ' − {x}            ; a single tuple x.
21.         end-if
22.     end-while
23. End.
```

Figure 1: The greedy algorithm for decomposing a relation.

queries of formulas expressed in Ω. In particular, a Horn tightest upper bound will infer correctly all Horn queries.

Observation 1: *Let $U_\Omega(\varphi)$ be the unique tightest upper bound of φ. For every $\alpha \in \Omega$, $U_\Omega(\varphi) \models \alpha$ iff $\varphi \models \alpha$.*

Proof: Clearly, if $U_\Omega(\varphi) \models \alpha$ then $\varphi \models \alpha$. If $\varphi \models \alpha$ then α is an upper bound of φ. Since $U_\Omega(\varphi)$ is the tightest upper bound, it also entails α. □

Consequently, it is meaningless to measure the effectiveness of the approximation with respect to queries from the bounding language since we are guaranteed correct answers.

We evaluate the effectiveness of the approximation on artificially generated relations and three real life databases: the KRK problem from the chess domain, the "politicians" relation that represents voting records of politicians, and the "breast-cancer" relation that represents medical records of patients.

4.1 Horn upper bound

Tables 1 and 2 summarize the results for random input relations. In Table 1, the input is a relation (whose number of attributes and models are given) and the output is the number of superfluous models in the tightest Horn upper bounding relation. Table 2 reports the fraction of entailment queries correctly answered as a function of the clauses size, for relations having 10 variables and 50 models (also reported in the 2nd row of Table 1). Additional details are provided in Figure 3(a) by the curve labeled "Single Partition". Note that most of the 3-literal clauses were *not* entailed by the theory nor by its upper bound.

Real Life Data: The "politicians" relation is defined over 16 bi-valued attributes and consists of 125 tuples. The tightest Horn upper bound consists of 1160 tuples.

As observed, although tightest Horn upper bounds exclude many non-models, they also contain numerous superfluous models and thus may be unacceptable for answering membership queries.

Table 1
membership queries

Num of variables	Num of models	Superfluous models
9	32	140
10	50	275
11	200	1052
15	50	1527

Table 2
entailment queries

Num of literals	Entailment accuracy
3	99%
5	82%
6	67%
7	60%

4.2 Disjunctive Horn approximations

We next show the improvement (over tightest Horn upper bound) obtained using disjunction of Horn theories. Given a constant k, the *partitioning algorithm* computes a k-disjunctive Horn theory. As described earlier, the algorithm partitions the input relation ρ into k disjoint subrelations of equal size, $\rho_1, \ldots, \rho_k$, and compute the Horn upper bound of each ρ_i yielding $U_\Omega(\rho_1), \ldots, U_\Omega(\rho_k)$. The results are summarized in Figures 2 and 3.

In Figure 2 we plot the size $|\cup_i U_\Omega(\rho_i)|$ (number of models) as a function of k, the disjunction size. The input relations have 9 and 11 attributes with 32 and 200 models respectively. We show, for instance, that when approximating with five theories, for 9 and 11 attributes the fraction of superfluous models (with respect to $2^9, 2^{11}$) was reduced from 29%,51% to 8%,33% respectively.

In Figure 3 we report the results obtained on relations having 10 attributes and 50 models. In this case, the tightest single upper bound Horn approximation had 325 models on average. Every point is obtained by testing all clauses of a fixed length and computing the fraction of correctly answered queries. This is averaged over 50 relations. We observe in Figure 3(a) that the approximation obtained using disjunction of nine Horn theories was significantly better than the tightest upper bound. In Figure 3(b) we report the accuracy as it improves when

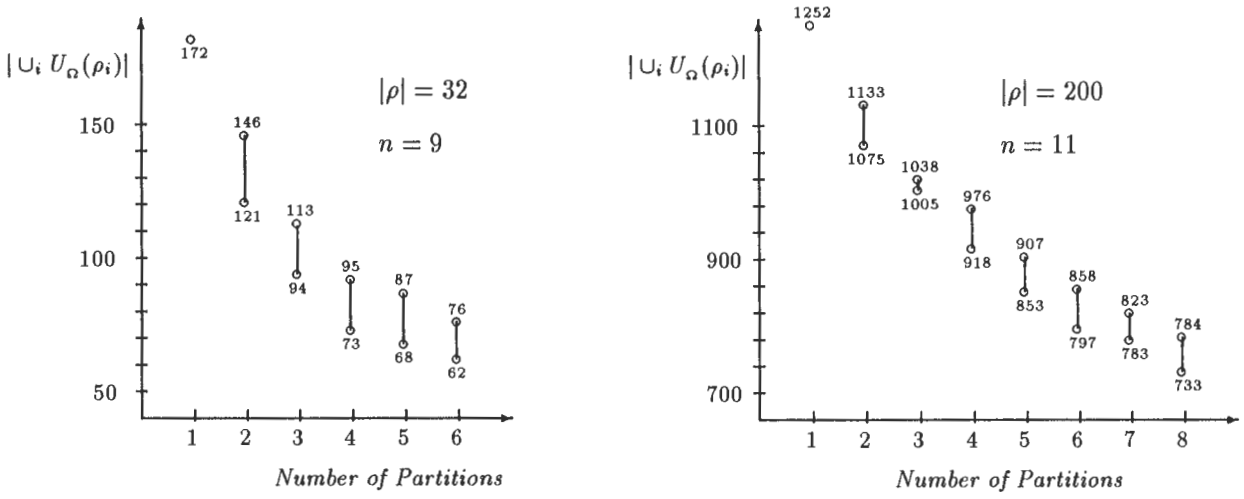
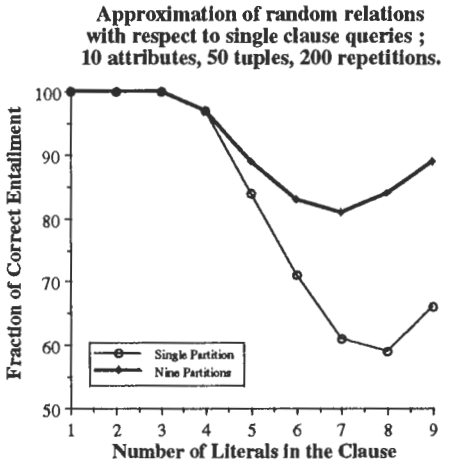
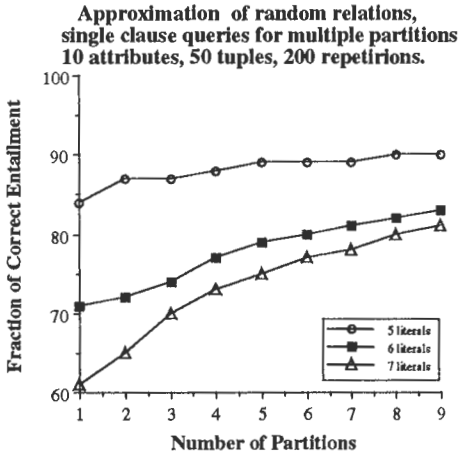


Figure 2: Decomposition by partitioning of random relations.



(a) effectiveness vs clause size,



(b) effectiveness vs number of partitions.

Figure 3

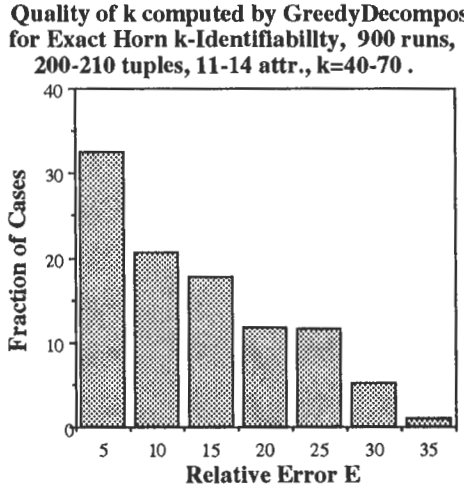


Figure 4: Stability of GreedyDecompose.

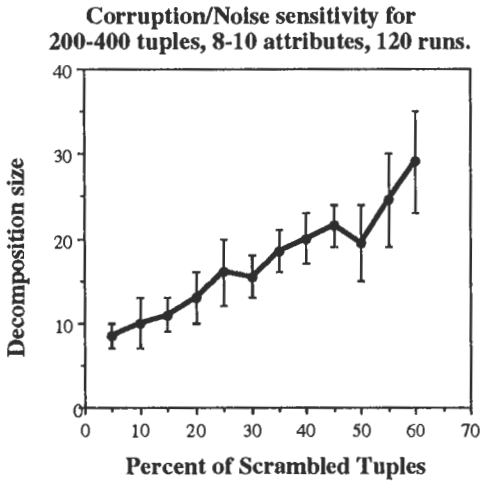


Figure 5: Near-Horn relations.

the number of theories in the disjunction increases. We plot results for clauses with 5,6,7 literals. Note that 50% is the lowest accuracy possible since guessing yields at least 50%. Figure 3(b) suggests that for small clause entailment, disjunctive upper bounds do not improve much over single Horn bounds, because for small clauses the single Horn upper bound is already quite effective. As the clause size increases, there are more non-Horn clauses and therefore single Horn upper bounds are less effective and consequently disjunctive bounds can be more cost-effective.

The same experiments were performed on the "politicians" relation and roughly the same results were obtained.

In the remainder of this section we report results of experiments using *GreedyDecompose* for computing both exact decomposition and approximate disjunctive upper bounds.

We next focus on the behavior *GreedyDecompose*. Since there exists an ordering of models for which the greedy algorithm yields an optimal decomposition size k, we evaluate its effectiveness by observing the variations of k as a function of the order by which models are processed. For every relation generated, *GreedyDecompose* was run 30 times, each time with a different random ordering. Q_0 denotes the smallest size decomposition out of the 30 computed, and $Q_1, \ldots, Q_{29}$ denote the other 29 decompositions. The stability, is measured by the *relative error* $E_i = \frac{|Q_i|-|Q_0|}{|Q_0|}$, namely the fraction of cases the size of the decomposition, $k_i = |Q_i|$, was larger than $k_0 = |Q_0|$. As shown in Figure 4, in about 32% of the cases the size of the decomposition computed was less than 5% larger than $k_0 = |Q_0|$ and the largest variation was 35%. Thus the algorithm is reasonably stable.

We next report results of experiments with exact decompositions. The experiments were performed on small bi-valued random relations, having about 30 models, and larger relations, having 300 models. As Table 3(a) shows, for relations with 30-34 models having 10 attributes, about 15 theories were required on the average (i.e., k=15). For larger relations with 300-325 models having 14 attributes, about 130 theories were needed. Clearly, since the algorithm is not optimal, we do not know whether smaller decompositions exist.

Tree networks (constraint networks whose constraint graph is a tree) were examined next. We observed in Table 3(b) that, as in the case of *Horn theories*, k might be arbitrary large. However, it does not increas as fast as the number of attributes.

To evaluate sensitivity to noise, we take a relation that can be described by a Horn theory and corrupt randomly selected models; the degree of noise introduced is measured by the percentage of models corrupted. A model is corrupted by flipping each of its bits with 0.5 probability. As shown in Figure 5, when only a few models were corrupted, the corrupted relation admits a relatively small k.

We next examine the effectiveness of bounded overflow on k. Namely, we compute a (k, ε) upper bound. To demonstrate the effect on k, we show (Figure 6(a)) the dependence of the *Horn* disjunct size, k, on the overflow fraction, defined as $\xi = \frac{|\cup_i U_\Omega(\rho_i)-\rho|}{|\rho|}$. As expected, we see that increasing overflow reduces the decomposition size.

Finally, we measure the trade-off between noise and overflow. We plotted the number of overflow models with respect to the number of corrupted models while holding k under 5. We observe that k, which was increased by corruption or noise, can be decreased by allowing overflow proportional to the degree of corruption (Figure 6(b)).

4.2.1 Real Life Data

We report results of experiments made with three real life databases taken from the machine learning repository at U.C. Irvine. We first examine Horn k-decompositions of a bi-valued relation that represents voting records of politicians. The relation can be represented exactly by 48 Horn theories. However, by allowing overflow, this number can be reduced (see Figure 7(a)).

The breast cancer relation represents records of symptoms and diagnosis (i.e. whether the patient had breast cancer or not). The task is to derive a theory that enables efficient processing queries that involve symptoms and diagnosis. The relation can be described exactly by 38 tree networks; however, by allowing overflow, this number can be reduced (see Figure 7(b)).

4.3 Learning with Horn upper bound

We next suggest that perhaps compilation methods that aim at providing a tractable and concise representation when all the data is available can be modified and used for learning when only part of the data is available.

The experiments are performed on the King Rook King (KRK) problem from the chess domain. The task is to learn a predicate that classifies board positions as either legal or illegal, given a small training set with positive and negative examples. Each example is a tuple with 6 attributes that specify the coordinates of the white king, the black rook and the black king. The multi-valued training relation $\rho_{training}$ is transformed into bi-valued relation $\rho'_{training}$ using a set of predicates provided by the expert, and $U_\Omega(\rho'_{training})$ is computed. To classify an unseen instance x, we map it to a bi-valued instance x' and check whether $x' \in U_\Omega(\rho'_{training})$. If $x' \in U_\Omega(\rho'_{training})$, then the classification can be determined accordingly. Otherwise, we guess the most frequent class.

We compare performance with a recent algorithm for learning prolog programs, called FOCL [7]. Figure 4 shows that $U_\Omega(\rho_{training})$ was able to correctly classify about 95% of the unseen examples when trained on 300 examples while FOCL was able to achieve better accuracy with only 200 examples. The curve labeled by "HORN known" shows the fraction of unseen examples found in the closure and that were correctly classified[2]. The curve labeled "HORN" shows the final accuracy

[2]Since the target concept in the KRK domain is not Horn, we can only approximate it.

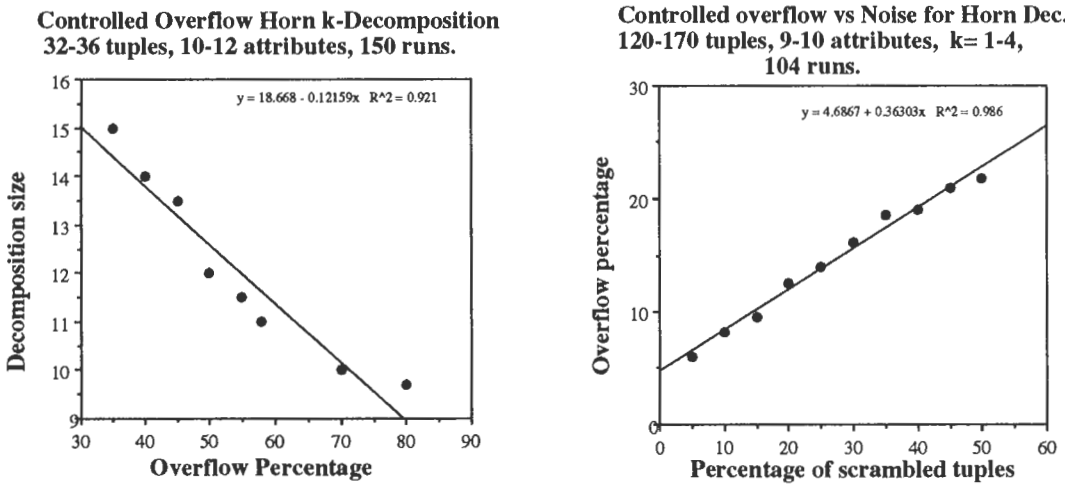
Table 3: Exact decomposability.

(a) k-Horn

Number of Attributes	*Number of Disjuncts*	*Min Num. Disjuncts*	*Max Num Disjuncts*
Horn $k-$decomposition, $30-34$ models, 250 runs			
6	5	4	6
7	8	6	10
8	11	7	14
9	14	9	16
10	15	9	17
Horn $k-$decomposition, $300-325$ models, 1040 runs			
11	70	68	72
12	90	84	98
13	113	105	122
14	133	126	140

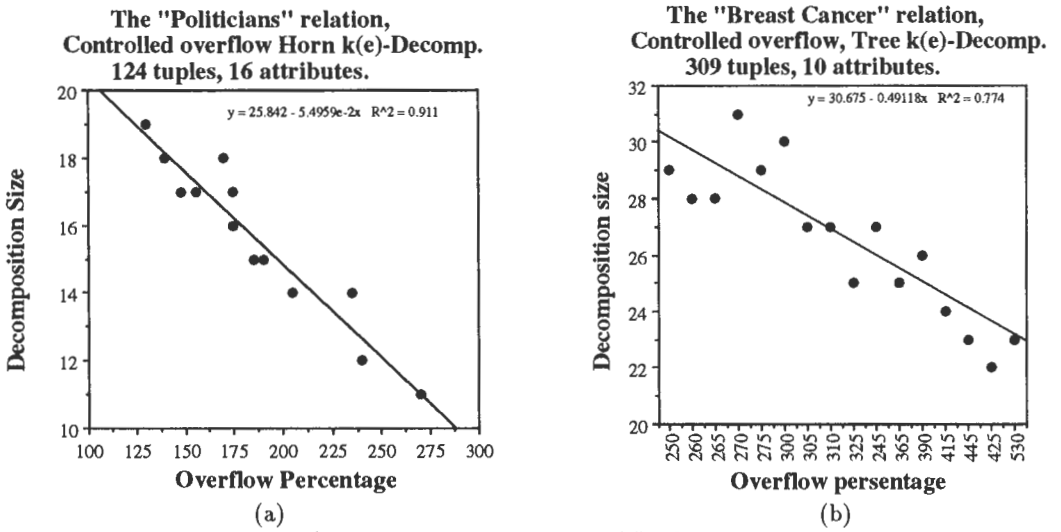
(b) k-Tree

Number of Attributes	*Number of Disjuncts*	*Min Num. Disjuncts*	*Max Num Disjuncts*
Tree $k-$decomposition, $32-36$ models, 125 runs			
8	9.4	8	11
9	11.0	9	13
10	11.5	10	13
11	12.0	11	13
12	13.1	12	14
Tree $k-$decomposition, $200-202$ models, 105 runs			
10	39	35	44
12	52	48	58
14	63	59	65
16	69	64	73



(a) (b)

Figure 6: Sensitivity to noise of Exact k-Horn decomposability.



(a) (b)

Figure 7: Decomposability of "real data"

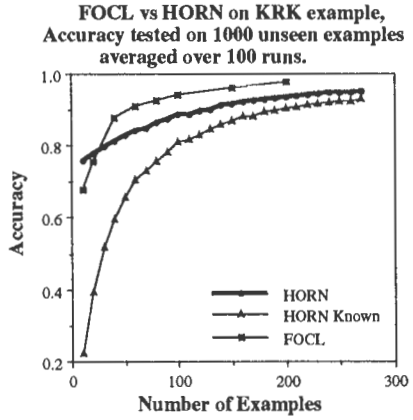


Figure 8: Learning with tightest Horn upper bound approximations.

achieved by guessing the most frequent class when the unseen instance is not in $U_\Omega(\rho'_{training})$. For more details on the use of single-upper bounds on learning see [8].

5 Conclusion

This paper builds upon prior investigations into the prospects of compiling empirical data into structures that allow efficient processing of queries [2]. Previous work had presented algorithms for describing or approximating data by Horn theories [1, 2] or tree constraint networks [2, 3]. The effectiveness of these approximation should be measured with respect to the class of queries to be asked. We focus on entailment queries, which are common in automated reasoning, and membership queries, which are common in classification tasks. Upon testing these approximations on artificially generated and real life data, it was immediately observed that, although effective for some entailment queries, they are ineffective for membership queries since they yield numerous supefluous models.

We therefore propose to improve on the single tightest upped bound approximation by approximating with a disjunction of theories. We define the *k-decomposition problem*: given an integer k and a relation ρ, determine whether ρ can be described by a *disjunction* of k tractable theories. The paper presents the necessary and sufficient conditions for a relation to be k-decomposable and identifies cases in which determining 2-decomposability is polynomial.

Because computing a k-decomposition is a difficult task, we examine polynomial approximation algorithms for two related formulation of this problem: (1) (minimization) given a theory φ and a language Ω, find the minimal k for which φ is *k-decomposable* relative to Ω. (2) (upper bound decomposition) given Ω and k, find a k-disjunctive upper-bound of φ relative to Ω that minimizes the number of superfluous models. We evaluate the effedctiveness of these approximations empirically with respect to both entailment and membership queries.

In our experiments we focus on the class of *tree constraint networks* and *Horn theories* and report results on artificially generated relations and on three real life cases. For the second task, we observe that the quality approximation obtained by upper bound decomposition improves as k increases. For the first task, when the data comes from a near-tractable source, or when the overflow is proportional to the level of noise, the approach is useful since k is small. However, when the input relation is not generated by a near-tractable source and we require very high quality approximations in which the number of superfluous models is bounded, decomposition is not effective since k grows almost linearly with the size of the input relation.

Fianlly we suggest that perhaps compilation methods, that aim at providing a tractable and concise representation when all the data is available, can be modified and used for learning when only part of the data is available. Experiments show that for the King Rook King problem the generalizing power of the tightest upped bound Horn approximation is comparable to that of recently developed learning algorithms. For more details on the use of single-upper bounds on learning see [8].

Acknowledgments

We would like to thank Dan Roth for valuable comments on an earlier version of this paper. The breast cancer database was obtained from the University Medical Center, Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the breast cancer data.

References

[1] Angluin, D., Frazier, M., Pitt, L., 1991. Learning conjunctions of Horn Clauses, *Machine Learning* 9:147-164.

[2] Dechter, R., Pearl, J., 1992. Structure Identification in Relational Data, *Artificial Intelligence* 58:237-270.

[3] Dechter, R., 1990. "Decomposing a relation into a tree of binary relations", *Journal of Computer and Systems Sciences*, special issue on the Theory of Relational Databases, Vol 41, 2-24 (1990)

[4] Dechter, R., Itai, A., 1992. Finding all solutions if you can find one. UCI Tech Rep. 92-61.

[5] Kautz, H., Kearns, M.J., Selman, B.S., 1993. Reasoning with Characteristic Models, *Proceedings of AAAI-93*, 34-39.

[6] Maier, D., 1983. The theory of Relational Databases Computer Science press, Rockville, MD, 1983.

[7] Pazzani, M.,, Kibler, D., 1992. The utility of knowledge in inductive learning, *Machine Learning* 9 (1992), 57-94

[8] Schwalb, E., Dechter, R., Pazzani, M, 1993. Using identifiability for learning Horn logic programs, UCI Tech-Rep. 94-13.

[9] Selman, B., Kautz, H., 1992. Knowledge compilation using Horn approximation *Proceedings of AAAI-91*, 904-909.

Are Vector Space Models Capable of Inductive Learning in a Symbolic Environment?*

Lev Goldfarb, John Abela, Virendra C. Bhavsar and Vithal N. Kamat
Faculty of Computer Science
University of New Brunswick
Fredericton, N.B., Canada E3B 5A3
Ph.(506)453-4566, Fax.(506)453-3566
E-mail:goldfarb, x45i, bhavsar, u095@unb.ca

"Mathematicians have abstracted the mathematical process away from the specific examples that were used to motivate their introduction and they study the concept of 'number', or 'shape', or 'distance', in the abstract. This is done by focusing attention upon the operation by which numbers are changed rather than upon the numbers themselves. Thus, a simple counting process like 1,2,3, ... is seen not as a list of particular numbers but as the result of carrying out a particular operation of change upon a number, thereby generating its successor. ... Once an abstract notion of number is present in the mind, and the essence of mathematics is seen to be not the numbers themselves but the collection of relationships that exist between them, then one has entered a new world".

John D. Barrow. *Pi in the Sky.*

Abstract

We outline a general framework for inductive learning. The mathematical foundations of this framework include *two basic components*: *set of operations* (on objects) and the corresponding *geometry* which is defined by means of the operations. According to this framework, to perform inductive learning in a symbolic environment the set of operations must be dynamically updated and this requires the geometric component to have the capability of changing the topology dynamically. For symbolic systems, as used in this framework, the geometric component has the ability of dynamically changing its topology whereas finite-dimensional numeric systems can essentially have only one (static) topology. This implies, in particular, that the vector space based models, e.g. artificial neural networks, cannot capture inductive generalization in a symbolic setting. The recently proposed *evolving transformation system* model is the inductive learning model within this framework.

Keywords: Inductive learning, inductive generalization, vector space models, artificial neural networks, symbolic models, evolving transformation system, learning topologies.

1 Introduction

Learning has become one of the most important research areas in artificial intelligence. Within learning, the area of inductive learning has always been of central importance. There are two major directions being pursued within inductive learning: numeric (vector space) and symbolic. Artificial neural networks (ANNs) are representative of the numeric models whereas the version-space model [Mitchell, 1982] is an example of a model that can be applied to symbolic representations. Currently, much effort has been directed towards applying ANNs to learning in symbolic environments.

The reexamination of the above two types of semi-formal models, which we undertake in this paper, has been prompted by the recently proposed model for inductive learning - the *Evolving Transformation System* model (*ETS*) [Goldfarb, 1990a; Goldfarb, 1992]. The ETS model emerged as a result of an effort to unify the numeric and symbolic models within one framework.

The main objectives of this paper are as follows. The central objective is to compare and contrast inductive learning in the numeric and the symbolic models. The second objective is to define a new framework for inductive learning. We finally present strong arguments that suggest that the numeric models are incapable of inductive generalization in symbolic environments.

Section 2 briefly discusses the problem of inductive learning and generalization. The characteristics of the vector space model are reviewed in Section 3. In Section 4 we present a brief description of the ETS model. Some limitations of the vector space model are presented in Section 5. Finally, Section 6 discusses the fundamental limitations of the vector space model when applied to learning in symbolic environments.

*This research was partially supported by NSERC grants OGP2686 and OGP0089.

2 Inductive Learning and Generalization.

We define inductive learning to be a process by means of which, given a finite positive training set C^+ from a possibly infinite class (or concept) C, and a finite set C^- from the complement of C, an agent is able to reach a state which allows it to form an idea about the class. This state enables the agent to recognize a new object as belonging to class C or not.

Inductive learning is what humans use to learn the idea (or concept) of, for instance, a cat. Having seen a finite number of cats, we acquire the ability to recognize and classify any animal as being a cat or not.

It is generally accepted that the inductive process is the only process by means of which an agent increases its semantic information [Johnson-Laird, 1988]. At the same time, we would like to stress that it is meaningless to consider inductive learning without generalization, where generalization is simply the state of the agent after the inductive learning process. Moreover, we believe that it is also meaningless to speak of recognition without the concept of inductive generalization.

It appears that even motor control skills are acquired through inductive learning. A boy catching a ball does not understand ballistics, yet he manages to learn this feat with ease.

Thus, unsupervised classification i.e. without the teacher, corresponds to the recognition stage, the stage which follows the inductive learning process. We believe that a fundamentally new mathematical model is required in order to understand the inductive learning process. None of the classical mathematical models is suitable for modeling the inductive learning process. The reason is that, in order to construct such a model, a new way of encoding (or capturing) a possibly infinite set C from a finite subset of training objects is required. Mathematicians have not addressed this question so far and therefore no new scheme to answer it exists within present mathematics. The issue of modeling the inductive learning process is addressed further in Section 6.

3 The Normed Vector Space Model

In order to understand the strengths and limitations of the vector space model (which is the basis of artificial neural networks (ANNs)) one has to consider the underlying mathematical model. The underlying mathematical model consists of two components: algebraic and geometric (topological). This point is often ignored. The algebraic model is that of a vector space, which is characterized by the set of vector space axioms. Since all ANNs use real vector spaces, we will restrict ourselves in this paper to the considerations of a vector space over the field $\Re$ of real numbers. The axioms of the vector space describe, in essence, the properties of the *two basic operations* defined in the vector space - multiplication of a vector by a scalar from the field and vector addition (see [Godement, 1968; Jordan, 1988]). It should be noted that the underlying algebraic structure alone is not sufficient for the traditional vector space based learning algorithms. All such learning algorithms require introduction of the second component of the model - the geometric (or topological) structure. Without such a structure such concepts as distance between the vectors, convergence, and therefore the necessary objective functions for optimization, cannot be introduced. All these concepts are an integral part of the inductive learning process in the vector space. For example, during training ANNs make use of an iterative gradient algorithm (such as the back propagation algorithm) to minimize the mean square error [Lipmann, 1987].

The introduction of the geometric component in the vector space model has profound consequences which are very often overlooked. There is essentially only one geometric structure (topology) in a vector space that is consistent with the underlying algebraic operations (see theorem 3.3H p. 127 [Taylor, 1987]). By 'consistent' we mean that the two basic algebraic operations are continuous with respect to the geometric structure ([Taylor, 1987], p. 81).

The implication of this fact (uniqueness of geometry) to ANN models is as follows: there is essentially only one norm in a finite-dimensional vector space and this is usually chosen to be the Euclidean norm (since all norms are equivalent, theorem 3.12A, p. 96, [Taylor, 1987]). In other words, all 'useful' distance functions are equivalent to the Euclidean distance function and no other geometric (metric) structure is consistent with the underlying algebraic structure. By 'useful' distance functions we mean a distance function that is consistent with the underlying algebraic structure of the vector space. If a distance function is not consistent with the underlying algebraic structure, then all the standard analytical techniques such as limits, differentiation, integration, etc. become non-applicable. The continuity of the algebraic operations ensures that the local geometry is preserved throughout the entire vector space, i.e. metric properties at a point v of the vector space V are the same as at any other point $w \in V$. The standard techniques in mathematical analysis have been developed under the above requirement of consistency between the algebraic and geometric structure. Removing the requirement will make life more difficult since then the homogenuity of the normed vector space will be broken.

Proposition: All ANNs use a finite-dimensional topological vector space model consisting of two components - algebraic and geometric.

Theorem: The geometric component is uniquely defined by the more fundamental (primary) algebraic component.

The uniqueness mentioned above means that even though one can choose a number of different distance functions (metrics) on the vector space which are consistent with the underlying algebraic structure, these are, in fact, all equivalent. Two metrics are said to be equivalent if they generate the same topology and therefore the same geometric structure as we have used above. In other words, two metrics are equivalent if the convergence of a sequence of points under one of them implies convergence in the other one.

4 The Evolving Transformation System (ETS) Model

A new mathematical model for inductive learning - *Evolving Transformation System* (*ETS*) has been proposed in [Goldfarb, 1990a]. As in the case with the normed vector space model, the ETS also consists of two components. In the case of ETS, the components are *symbolic* and geometric. While both models have a geometric component built on top of their underlying structures, in the ETS model the underlying structure, symbolic component, is completely different - here it is a symbolic component. We will discuss the differences between the two underlying structures - symbolic and algebraic, in the next section.

The *symbolic* component, a *transformation system* (*TS*), is defined as a triple $\mathbf{T} = (\mathbf{O}, \mathbf{S}, \mathbf{CR})$, where $\mathbf{O}$ is a set of homogeneously structured objects, $\mathbf{S} = \{S_i\}_{i=1}^m$ is a finite set of operations that can transform object $o_1 \in \mathbf{O}$ to another object $o_2 \in \mathbf{O}$, and $\mathbf{CR}$ is a small finite set of composition rules (or operators) which permit one to construct new operations from the existing operations.

The set $\mathbf{CR}$ of composition rules allows the system to evolve in time by changing (usually enlarging) the set $\mathbf{S}$ of operations, thus leading to the concept of an evolving transformation system (ETS).

The second component, geometric structure, is defined as follows:

$$\mathbf{D} = \{\Delta_\omega\}_{\omega \in \Omega}$$

where Ω is the $(m-1)$ dimensional simplex in $\Re^m$

$$\Omega = \{\omega = (w^1, w^2, \ldots, w^m) \mid w^i \geq 0, \sum_{i=1}^{m} w^i = 1\}$$

and each of the distance functions Δ_ω is defined as follows. Weight w^i is assigned to the operation S_i and

$$\Delta_\omega(o_1, o_2) = \min_{\mathbf{s}_j = (S_1^{(j)}, \ldots, S_k^{(j)})} \{\sum_{i=1}^{k} w_{(j)}^i\}$$

where $\mathbf{s}_j$ is a sequence of operations that transforms o_1 into o_2. In other words, the minimum is taken over all possible sequences of operations that can transform structured object o_1 into structured object o_2.

To compute the above distance the system must *use its set of operations in a cooperative and competitive manner*. Thus, all properties of the system resulting from this definition should be viewed as *emergent* properties.

Learning in a TS reduces to the following optimization problem:

$$\max_{\omega \in \Omega} f(\omega), \qquad f(\omega) = \frac{f_1(\omega)}{c + f_2(\omega)}$$

where $f_1(\omega)$ is the Δ_ω-distance *between* C^+ and C^-, $f_2(\omega)$ is the average Δ_ω-distance *within* C^+, and c is a small positive constant to prevent the overflow condition (when the values of $f_2(\omega)$ approach 0).

Let Ω_{max} be a subset of Ω consisting of all the (global) maximums of f on Ω. It is easy to see that, for the given concept C and given set S of operations, every weighting scheme $\omega^* \in \Omega_{max}$ generates the "best" metric configuration of the training examples: under Δ_{ω^*} positive examples form the most compact set relative to the negative examples. Thus we are justified in calling function f the *quality of the (learning) class perception* [Goldfarb, 1992]. It is not difficult to see that if $f_2(\omega^*) = 0$ and $f_2(\omega^*) \neq 0$, the set $\mathbf{S}$ of operations is sufficient to produce a *complete separation* of C^+ and C^-. This is often not the case, since then the learning agent has no need to acquire any new operations, or new "features". Hence the need to consider an evolving system (ETS).

The inductive learning process for the ETS proceeds by constructing a sequence of $\mathbf{S}_i$'s in such a way that for the corresponding transformation system $\mathbf{T}_i$, the minimum value of f_2 decreases (while making sure that the value of f_1 is not zero), i.e. the interdistances in C^+ gradually shrink (to zero, when no noise is present), while the distance between C^+ and C^- remains nonzero [Goldfarb and Nigam, 1994].

It is very important to note that when the set of operations $\mathbf{S}_i$ in the evolving transformation system (ETS) changes to $\mathbf{S}_{i+1}$ the corresponding geometric structure changes from $\mathbf{D}_i$ to $\mathbf{D}_{i+1}$, i.e.

$$\mathbf{D}_i = \{\Delta_{\omega_i}\}_{\omega_i \in \Omega_i}$$

becomes

$$\mathbf{D}_{i+1} = \{\Delta_{\omega_{i+1}}\}_{\omega_{i+1} \in \Omega_{i+1}}.$$

An important basic example of a TS is the string TS, where the set of objects $\mathbf{O}$ consists of strings over a finite alphabet, the set $\mathbf{S}$ of operations consists of single or multiple-letter insertion/deletion/substitution operations, $\mathbf{D}$ is a set of weighted Levenshtein (string-edit) distance functions [Kruskal and Sankoff, 1983], and $\mathbf{CR}$ consists of a small number($\leq$ 3) of rules that allow the formation of multiple-letter deletion/insertion/substitution operations.

There are fundamental differences between the algebraic structure of the vector space model (ANNs) and the underlying symbolic structure (TS) in the ETS. These stem from the fact that the concept of an algebraic operation is quite different from that of a symbolic operation. An algebraic operation (e.g. vector addition) is defined as a function that assigns for every pair of vectors another vector in the vector space. A symbolic operation (e.g. insertion of a single letter a) could be applied at any place in a given string, so that in this case the operation is multivalued.

The above difference in the underlying structures results in significant and critical differences in the corresponding induced geometric structure. In the ETS model the family of distances $\mathbf{D}_{i+1}$ has a member (distance function) which is not equivalent to any of the distances in all the previous families of distance functions $\mathbf{D}_0, \ldots, \mathbf{D}_i$, i.e. there exists $\Delta_{\omega_{i+1}} \in \mathbf{D}_{i+1}$ that generates a topology which is different from the topologies generated by any member $\Delta_{\omega_j} \in \mathbf{D}_j, 0 \leq j \leq i$ of the previous families [Goldfarb, 1993]. Thus instead of the single topology of a finite-dimensional vector space we now have an *infinite family of topologies associated*

with the symbolic system. This has a fundamental implication for the inductive learning model - ETS, as discussed in the following section. It should be noted that by enlarging a vector space of n-dimensions to $n+1$ dimensions, the topology of the n-dimensional space (now a subspace) does not change.

5 Some Limitations of the Vector Space Representation

One fundamental limitation of the vector space model, when used in any context, relates to the fact that any chosen input variables cannot be assumed to be commensurate. As discussed in [Goldfarb, 1985] the imposition of the Euclidean distance on the chosen set of measurable variables (features), as is done in ANN models, assumes that all these variables (features) are commensurable. However, as is well known from the theory of special relativity, even such well known variables as the three space coordinates and the time coordinate turn out to be non-commensurate (as realized by Minkowski and Einstein) and consequently required the introduction of Minkowski distance to arrive at a more appropriate mathematical model for space-time. In other words, the moral of this physical theory for us is: even if one assumes a vector space structure, one should not assume that the inner-product vector space generated by any variables is necessarily Euclidean.

The generality of the metric space, as compared to the Euclidean vector space, manifests itself, for example, in the following fact: consider a finite alphabet and the Levenshtein distance defined on a set of strings over the alphabet. Then, four randomly chosen strings cannot be represented isometrically, i.e. preserving the inter-string distances, in the Euclidean vector space of any dimension [Goldfarb, 1985]. In other words, there is no Euclidean vector space of any finite-dimension in which one can find four vectors that have the same inter-distance as the chosen four strings. This implies that the metric information, indispensable for capturing inductive generalization in the symbolic setting, cannot be represented in the finite-dimensional vector space.

Another fundamental limitation of the vector space model when applied to inductive learning in a symbolic environment is connected to *the necessity of ordering the input alphabet (symbols).* Consider biological sequence classification which is an important part of the Genome Project. The input space consists of strings over a finite alphabet. For example, for DNA, $\Sigma = \{g, a, c, u\}$, where each letter represents a DNA nucleotide (or base) [Creighton, 1993]. In order to map the symbolic data into the vector space one must order the input alphabet. This ordering is not related to the symbolic data and since there cannot be any basis for such an ordering, any chosen ordering is arbitrary. Furthermore, *an ordering introduces a topology* which is unique to the vector space [Section 3]. Once a particular ordering has been chosen, a topology for that particular ordering would be introduced. This is because the ordering of the real numbers is responsible for the unique topology of the input vector space. Since the ordering was arbitrary, the induced topology will not be relevant to the structure of the symbolic input, as captured by the symbolic operations of the TS, or equivalently as is present in the symbolic input (biological sequences). As seen in Section 4, this means that any symbolic operation does not have an appropriate interpretation in a vector space.

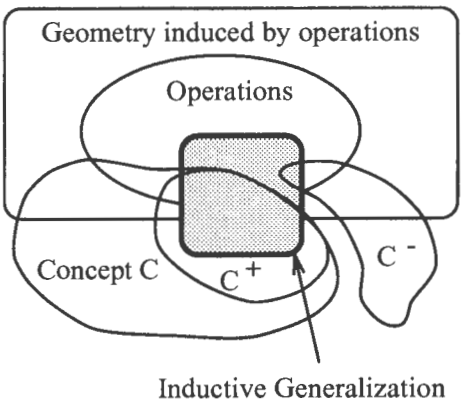


Figure 1: A schematic representation of a formal framework for inductive learning

Ordering of the real numbers (generated by the Peano relation on the natural numbers) is fundamentally linked to the intrinsic structure of $\Re$, while any ordering of the alphabet has nothing to do with the corresponding symbolic structure of the DNA bases.

To recapitulate, under any mapping of a biological sequence (string) into a vector space one must necessarily order the input alphabet. Contrary to the situation with real numbers, the ordering of the input alphabet must, as discussed, be arbitrary and this ordering of the symbols will not be related in any way to the symbolic structure of the input. In other words, while the Peano relation on the natural numbers is relevant (in fact, critical) to generating reals, its analog is absolutely irrelevant to a typical symbolic system as used in AI. This is because symbols usually represent non-numeric object features such as facial or geometric features and the Peano relation when imposed on an alphabet introduces a structure which is inappropriate and misleading. One should note that more appropriate structures/operations for the symbolic systems are those related to the operation of ETS, since insertion/deletion/substitution are the only natural operations for transforming one string into another.

6 A Framework for Inductive Learning and its Implications

Consider the mathematical framework for inductive learning of a single concept shown in Figure 1; the case for multiple concepts fits within the same framework. The main part of the framework is the basic mathematical model consisting of two components: a set of operations, and an induced geometry. For example, as discussed in Section 3, in the vector space model the two components are the algebraic component (viz. algebraic operations) and the geometric component (viz.

the norm on the vector space). On the other hand, for the symbolic model (see Section 4) we have the symbolic component (e.g. string TS) and the corresponding geometric component (e.g. weighted Levenshtein family of distance functions).

In our view, the geometric component in the above framework has often been overlooked in spite of its critical contribution to the inductive learning process. This component is necessary in order to define and capture the idea of inductive generalization. Informally, the geometry acts as a glue that holds the elements of a class together. This fact is reflected in the role of the similarity concept in the area of categorization in psychology [Bourne *et al.*, 1986].

As discussed in Section 2, inductive learning is a process by means of which given a finite positive training set C^+ from a possibly infinite class C, and a finite set of negative training examples C^- from the complement of C, an agent is able to construct an inductive generalization of C. As depicted in Fig. 1, we propose that inductive generalization must be expressed using *both* components of the mathematical model; the operations play the role of "features" in the generalization, while the corresponding distance function plays the role of the "glue that holds the elements of the class together". Formally, the *inductive generalization* is a triple $(\overline{C}^+, \mathrm{S}, \Delta_{\omega^*})$, where $\overline{C}^+$ is a reduced positive training set, S is the final set of operations and Δ_{ω^*} is the learned distance measure. The elements of $\overline{C}^+$ act as reference patterns for defining the class and consequently a new input pattern is always compared with these reference patterns using the Δ_{ω^*}. The set S of operations is necessary because the concept of distance can properly be defined only in terms of the operations: this is because the geometry must be consistent with the underlying set of operations. This form of inductive concept representation, or generalization, is in complete agreement with the most accepted theory of concept learning - Rosch's examplar theory. "Exemplar theory claims that concept-learning is accomplished by memorizing specific instances and by using some measure of instance similarity" [Bourne *et al.*, 1986].

The proposed framework has important implications even to inductive learning in a vector space. *If the vector space model is recast in the above framework* (see [Goldfarb, 1990a] and example 2 in [Goldfarb, 1990b]), then the set of operations is fixed and, as we have also seen in Section 3, the geometric component is uniquely determined by the set of operations. The most important implication of this fact relates to the form of the learned inductive generalization. According to the proposed framework, the *exact* inductive generalization will be an affine subspace of the vector space. For example, if the vector space is the vector space of functions, the inductive generalization is an affine subspace in the space of functions. It should be noted that since there is only one underlying geometry, this geometry does not change during learning. Consequently, no learning of the geometric component (corresponding to the training set) can occur. Therefore, to start with, the unique geometry of the vector space is simply imposed on the training set.

In fact, the proposed framework was motivated by the desire to allow the training set to generate the geometry appropriate for the inductive generalization of the class. As stated in Section 4, to arrive at inductive generalization in a symbolic setting, we must first learn the geometry corresponding to the training set and, thus, the geometric component begins to play a critical role.

To recapitulate, in the symbolic setting, the set of operations consists of substitution operations and the corresponding geometric component is defined by means of weighted operations. During inductive learning, an agent acquires the necessary new operations (composed from earlier operations) as well as the weights of all the operations [Goldfarb and Nigam, 1994].

We believe that the final set of learned operations represents a communicable and compact form of the inductive generalization and, furthermore, that the numeric component in any model does not represent a communicable part of the inductive generalization. We also strongly feel that communicability is related to the differences between the numeric and symbolic mathematical structures.

We also believe that the essential part of inductive generalization should be its communicability. The communicability refers to that between the various components of the agent as well as between different agents (for example, between a learning machine and a human agent). In the case of symbolic processing in vector space learning machines, the symbols are initially converted to vectors, and therefore the symbolic information (see Section 5) is not present during the learning process. Hence, the results of this learning *cannot contain any symbolic information.*

Thus, the vector space based models (e.g. ANNs) are absolutely incapable of inductive learning in symbolic environments. This is essentially due to the fact that these models construct, as a result of learning, polyhedral regions (in the input vector space) enclosing C^+. According to the proposed definition of inductive generalization such regions are not sufficient to "allow the agent to form an idea about the class". First, there are uncountably many *appropriate choices* for such regions (in a vector space over reals). Second, an agent cannot "form an idea about the class" on the basis of such regions. In contrast, the ETS model is capable of capturing the necessary inductive generalization in symbolic environments and this generalization is communicable within an agent as well as between agents.

7 Conclusion

We have outlined a framework for inductive learning. We have also explicated the fact that symbolic systems are mathematically and computationally quite different from numeric systems. Moreover, we have explained why symbolic information cannot be captured by numeric systems. It should be noted that the situation is no different in the case when a symbolic system is constructed on top of a numeric system.

The implication of the above to vector space models, and to ANNs in particular, is that they cannot perform

symbolic inductive generalization (the result of the inductive learning process).

As proposed in the evolving transformation system (ETS) model, the inductive learning process can capture inductive generalization corresponding to an infinite class by means of *both* the symbolic information (essentially the final set of operations constructed during the learning process) and numeric information (the weights attached to the operations).

In conclusion, we think that for any agent (artificial and, quite possibly, biological) engaged in inductive learning the symbolic component must play the central role. Otherwise, the agent simply will not be able to achieve inductive generalization.

References

[Bourne *et al.*, 1986] L. E. Bourne, R. L. Dominowski, E. F. Loftus, and A. F. Healy. *Cognitive Processes.* 2nd Ed., Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[Creighton, 1993] T. E. Creighton. *Proteins - Structures and Molecular Properties.* W.H. Freeman and Co., New York, NY, 1993.

[Godement, 1968] R. Godement. *Algebra.* Houghton Mifflin Company, Boston, MA, 1968.

[Goldfarb, 1985] L. Goldfarb. A New Approach to Pattern Recognition. In *Progress in Pattern Recognition 2*, eds. L. N. Kanal and A. Rosenfield, North-Holland, pages 241–402, 1985.

[Goldfarb, 1990a] L. Goldfarb. On the Foundation of Intelligent Processes - I. An Evolving Model for Pattern Recognition, *Pattern Recognition*, 23:596–616, 1990.

[Goldfarb, 1990b] L. Goldfarb. A Unified Metric Model for Pattern Learning. In *Proc. IASTED Int. Symp. on Machine Learning and Neural Networks*, eds. M.H. Hamza, New York, 10-11 Oct, pages 96–99, 1990.

[Goldfarb, 1992] L. Goldfarb. What is distance and why we need the Metric Model for Pattern Learning. *Pattern Recognition*, 25(4):431–438, 1992.

[Goldfarb, 1993] L. Goldfarb. On some mathematical properties of the ETS model. Technical Report TR93-079, Faculty of Computer Science, UNB, September 1993.

[Goldfarb and Nigam, 1994] L. Goldfarb and S. Nigam. The Unified Learning Paradigm - A Foundation for A.I. *Artificial Intelligence and Neural Networks: Steps Towards Principled Integration*, eds. V. Honovar and L. Uhr, to appear, Academic Press.

[Johnson-Laird, 1988] P. N. Johnson-Laird. *The Computer and the Mind - An Introduction to Cognitive Science.* Harvard University Press, Cambridge, MA, 1988.

[Jordan, 1988] M. I. Jordan. An Introduction to Linear Algebra in Parallel Distributed Processing In *Parallel Distributed Processing*, eds. D. E. Rumelhart and J. L. McClelland, pages 365–422, 1988.

[Kruskal and Sankoff, 1983] J. B. Kruskal and D. Sankoff. eds. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley Publishing Company Inc., Reading, MA, 1983.

[Lipmann, 1987] R. P. Lipmann. An Introduction to Computing with Neural Networks. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[Mitchell, 1982] T. M. Mitchell. Generalization as search, *Artificial Intelligence*, 18(2):203–236, 1982.

[Taylor, 1987] A. E. Taylor. *Introduction to Functional Analysis*, John Wiley and Sons Inc., New York, NY, 1987.

Are Vector Space Models Capable of Inductive Learning in a Symbolic Environment?*

Lev Goldfarb, John Abela, Virendra C. Bhavsar and Vithal N. Kamat
Faculty of Computer Science
University of New Brunswick
Fredericton, N.B., Canada E3B 5A3
Ph.(506)453-4566, Fax.(506)453-3566
E-mail:goldfarb, x45i, bhavsar, u095@unb.ca

"Mathematicians have abstracted the mathematical process away from the specific examples that were used to motivate their introduction and they study the concept of 'number', or 'shape', or 'distance', in the abstract. This is done by focusing attention upon the operation by which numbers are changed rather than upon the numbers themselves. Thus, a simple counting process like 1,2,3, ... is seen not as a list of particular numbers but as the result of carrying out a particular operation of change upon a number, thereby generating its successor. ... Once an abstract notion of number is present in the mind, and the essence of mathematics is seen to be not the numbers themselves but the collection of relationships that exist between them, then one has entered a new world".

John D. Barrow. *Pi in the Sky.*

Abstract

We outline a general framework for inductive learning. The mathematical foundations of this framework include *two basic components*: *set of operations* (on objects) and the corresponding *geometry* which is defined by means of the operations. According to this framework, to perform inductive learning in a symbolic environment the set of operations must be dynamically updated and this requires the geometric component to have the capability of changing the topology dynamically. For symbolic systems, as used in this framework, the geometric component has the ability of dynamically changing its topology whereas finite-dimensional numeric systems can essentially have only one (static) topology. This implies, in particular, that the vector space based models, e.g. artificial neural networks, cannot capture inductive generalization in a symbolic setting. The recently proposed *evolving transformation system* model is the inductive learning model within this framework.

Keywords: Inductive learning, inductive generalization, vector space models, artificial neural networks, symbolic models, evolving transformation system, learning topologies.

1 Introduction

Learning has become one of the most important research areas in artificial intelligence. Within learning, the area of inductive learning has always been of central importance. There are two major directions being pursued within inductive learning: numeric (vector space) and symbolic. Artificial neural networks (ANNs) are representative of the numeric models whereas the version-space model [Mitchell, 1982] is an example of a model that can be applied to symbolic representations. Currently, much effort has been directed towards applying ANNs to learning in symbolic environments.

The reexamination of the above two types of semi-formal models, which we undertake in this paper, has been prompted by the recently proposed model for inductive learning - the *Evolving Transformation System* model (*ETS*) [Goldfarb, 1990a; Goldfarb, 1992]. The ETS model emerged as a result of an effort to unify the numeric and symbolic models within one framework.

The main objectives of this paper are as follows. The central objective is to compare and contrast inductive learning in the numeric and the symbolic models. The second objective is to define a new framework for inductive learning. We finally present strong arguments that suggest that the numeric models are incapable of inductive generalization in symbolic environments.

Section 2 briefly discusses the problem of inductive learning and generalization. The characteristics of the vector space model are reviewed in Section 3. In Section 4 we present a brief description of the ETS model. Some limitations of the vector space model are presented in Section 5. Finally, Section 6 discusses the fundamental limitations of the vector space model when applied to learning in symbolic environments.

*This research was partially supported by NSERC grants OGP2686 and OGP0089.

2 Inductive Learning and Generalization.

We define inductive learning to be a process by means of which, given a finite positive training set C^+ from a possibly infinite class (or concept) C, and a finite set C^- from the complement of C, an agent is able to reach a state which allows it to form an idea about the class. This state enables the agent to recognize a new object as belonging to class C or not.

Inductive learning is what humans use to learn the idea (or concept) of, for instance, a cat. Having seen a finite number of cats, we acquire the ability to recognize and classify any animal as being a cat or not.

It is generally accepted that the inductive process is the only process by means of which an agent increases its semantic information [Johnson-Laird, 1988]. At the same time, we would like to stress that it is meaningless to consider inductive learning without generalization, where generalization is simply the state of the agent after the inductive learning process. Moreover, we believe that it is also meaningless to speak of recognition without the concept of inductive generalization.

It appears that even motor control skills are acquired through inductive learning. A boy catching a ball does not understand ballistics, yet he manages to learn this feat with ease.

Thus, unsupervised classification i.e. without the teacher, corresponds to the recognition stage, the stage which follows the inductive learning process. We believe that a fundamentally new mathematical model is required in order to understand the inductive learning process. None of the classical mathematical models is suitable for modeling the inductive learning process. The reason is that, in order to construct such a model, a new way of encoding (or capturing) a possibly infinite set C from a finite subset of training objects is required. Mathematicians have not addressed this question so far and therefore no new scheme to answer it exists within present mathematics. The issue of modeling the inductive learning process is addressed further in Section 6.

3 The Normed Vector Space Model

In order to understand the strengths and limitations of the vector space model (which is the basis of artificial neural networks (ANNs)) one has to consider the underlying mathematical model. The underlying mathematical model consists of two components: algebraic and geometric (topological). This point is often ignored. The algebraic model is that of a vector space, which is characterized by the set of vector space axioms. Since all ANNs use real vector spaces, we will restrict ourselves in this paper to the considerations of a vector space over the field $\Re$ of real numbers. The axioms of the vector space describe, in essence, the properties of the *two basic operations* defined in the vector space - multiplication of a vector by a scalar from the field and vector addition (see [Godement, 1968; Jordan, 1988]). It should be noted that the underlying algebraic structure alone is not sufficient for the traditional vector space based learning algorithms. All such learning algorithms require introduction of the second component of the model - the geometric (or topological) structure. Without such a structure such concepts as distance between the vectors, convergence, and therefore the necessary objective functions for optimization, cannot be introduced. All these concepts are an integral part of the inductive learning process in the vector space. For example, during training ANNs make use of an iterative gradient algorithm (such as the back propagation algorithm) to minimize the mean square error [Lipmann, 1987].

The introduction of the geometric component in the vector space model has profound consequences which are very often overlooked. There is essentially only one geometric structure (topology) in a vector space that is consistent with the underlying algebraic operations (see theorem 3.3H p. 127 [Taylor, 1987]). By 'consistent' we mean that the two basic algebraic operations are continuous with respect to the geometric structure ([Taylor, 1987], p. 81).

The implication of this fact (uniqueness of geometry) to ANN models is as follows: there is essentially only one norm in a finite-dimensional vector space and this is usually chosen to be the Euclidean norm (since all norms are equivalent, theorem 3.12A, p. 96, [Taylor, 1987]). In other words, all 'useful' distance functions are equivalent to the Euclidean distance function and no other geometric (metric) structure is consistent with the underlying algebraic structure. By 'useful' distance functions we mean a distance function that is consistent with the underlying algebraic structure of the vector space. If a distance function is not consistent with the underlying algebraic structure, then all the standard analytical techniques such as limits, differentiation, integration, etc. become non-applicable. The continuity of the algebraic operations ensures that the local geometry is preserved throughout the entire vector space, i.e. metric properties at a point v of the vector space V are the same as at any other point $w \in V$. The standard techniques in mathematical analysis have been developed under the above requirement of consistency between the algebraic and geometric structure. Removing the requirement will make life more difficult since then the homogenuity of the normed vector space will be broken.

Proposition: All ANNs use a finite-dimensional topological vector space model consisting of two components - algebraic and geometric.

Theorem: The geometric component is uniquely defined by the more fundamental (primary) algebraic component.

The uniqueness mentioned above means that even though one can choose a number of different distance functions (metrics) on the vector space which are consistent with the underlying algebraic structure, these are, in fact, all equivalent. Two metrics are said to be equivalent if they generate the same topology and therefore the same geometric structure as we have used above. In other words, two metrics are equivalent if the convergence of a sequence of points under one of them implies convergence in the other one.

4 The Evolving Transformation System (ETS) Model

A new mathematical model for inductive learning - *Evolving Transformation System* (*ETS*) has been proposed in [Goldfarb, 1990a]. As in the case with the normed vector space model, the ETS also consists of two components. In the case of ETS, the components are *symbolic* and geometric. While both models have a geometric component built on top of their underlying structures, in the ETS model the underlying structure, symbolic component, is completely different - here it is a symbolic component. We will discuss the differences between the two underlying structures - symbolic and algebraic, in the next section.

The *symbolic* component, a *transformation system* (*TS*), is defined as a triple $\mathbf{T} = (\mathbf{O}, \mathbf{S}, \mathbf{CR})$, where $\mathbf{O}$ is a set of homogeneously structured objects, $\mathbf{S} = \{S_i\}_{i=1}^m$ is a finite set of operations that can transform object $o_1 \in \mathbf{O}$ to another object $o_2 \in \mathbf{O}$, and $\mathbf{CR}$ is a small finite set of composition rules (or operators) which permit one to construct new operations from the existing operations.

The set $\mathbf{CR}$ of composition rules allows the system to evolve in time by changing (usually enlarging) the set $\mathbf{S}$ of operations, thus leading to the concept of an evolving transformation system (ETS).

The second component, geometric structure, is defined as follows:

$$\mathbf{D} = \{\Delta_\omega\}_{\omega \in \Omega}$$

where Ω is the $(m-1)$ dimensional simplex in $\Re^m$

$$\Omega = \{\omega = (w^1, w^2, \ldots, w^m) \mid w^i \geq 0, \sum_{i=1}^{m} w^i = 1\}$$

and each of the distance functions Δ_ω is defined as follows. Weight w^i is assigned to the operation S_i and

$$\Delta_\omega(o_1, o_2) = \min_{\mathbf{s}_j = (S_1^{(j)}, \ldots, S_k^{(j)})} \{\sum_{i=1}^{k} w_{(j)}^i\}$$

where $\mathbf{s}_j$ is a sequence of operations that transforms o_1 into o_2. In other words, the minimum is taken over all possible sequences of operations that can transform structured object o_1 into structured object o_2.

To compute the above distance the system must *use its set of operations in a cooperative and competitive manner*. Thus, all properties of the system resulting from this definition should be viewed as *emergent* properties.

Learning in a TS reduces to the following optimization problem:

$$\max_{\omega \in \Omega} f(\omega), \qquad f(\omega) = \frac{f_1(\omega)}{c + f_2(\omega)}$$

where $f_1(\omega)$ is the Δ_ω-distance *between* C^+ and C^-, $f_2(\omega)$ is the average Δ_ω-distance *within* C^+, and c is a small positive constant to prevent the overflow condition (when the values of $f_2(\omega)$ approach 0).

Let Ω_{max} be a subset of Ω consisting of all the (global) maximums of f on Ω. It is easy to see that, for the given concept C and given set S of operations, every weighting scheme $\omega^* \in \Omega_{max}$ generates the "best" metric configuration of the training examples: under Δ_{ω^*} positive examples form the most compact set relative to the negative examples. Thus we are justified in calling function f the *quality of the (learning) class perception* [Goldfarb, 1992]. It is not difficult to see that if $f_2(\omega^*) = 0$ and $f_2(\omega^*) \neq 0$, the set $\mathbf{S}$ of operations is sufficient to produce a *complete separation* of C^+ and C^-. This is often not the case, since then the learning agent has no need to acquire any new operations, or new "features". Hence the need to consider an evolving system (ETS).

The inductive learning process for the ETS proceeds by constructing a sequence of $\mathbf{S}_i$'s in such a way that for the corresponding transformation system $\mathbf{T}_i$, the minimum value of f_2 decreases (while making sure that the value of f_1 is not zero), i.e. the interdistances in C^+ gradually shrink (to zero, when no noise is present), while the distance between C^+ and C^- remains non-zero [Goldfarb and Nigam, 1994].

It is very important to note that when the set of operations $\mathbf{S}_i$ in the evolving transformation system (ETS) changes to $\mathbf{S}_{i+1}$ the corresponding geometric structure changes from $\mathbf{D}_i$ to $\mathbf{D}_{i+1}$, i.e.

$$\mathbf{D}_i = \{\Delta_{\omega_i}\}_{\omega_i \in \Omega_i}$$

becomes

$$\mathbf{D}_{i+1} = \{\Delta_{\omega_{i+1}}\}_{\omega_{i+1} \in \Omega_{i+1}}.$$

An important basic example of a TS is the string TS, where the set of objects $\mathbf{O}$ consists of strings over a finite alphabet, the set $\mathbf{S}$ of operations consists of single or multiple-letter insertion/deletion/substitution operations, $\mathbf{D}$ is a set of weighted Levenshtein (string-edit) distance functions [Kruskal and Sankoff, 1983], and $\mathbf{CR}$ consists of a small number($\leq$ 3) of rules that allow the formation of multiple-letter deletion/insertion/substitution operations.

There are fundamental differences between the algebraic structure of the vector space model (ANNs) and the underlying symbolic structure (TS) in the ETS. These stem from the fact that the concept of an algebraic operation is quite different from that of a symbolic operation. An algebraic operation (e.g. vector addition) is defined as a function that assigns for every pair of vectors another vector in the vector space. A symbolic operation (e.g. insertion of a single letter a) could be applied at any place in a given string, so that in this case the operation is multivalued.

The above difference in the underlying structures results in significant and critical differences in the corresponding induced geometric structure. In the ETS model the family of distances $\mathbf{D}_{i+1}$ has a member (distance function) which is not equivalent to any of the distances in all the previous families of distance functions $\mathbf{D}_0, \ldots, \mathbf{D}_i$, i.e. there exists $\Delta_{\omega_{i+1}} \in \mathbf{D}_{i+1}$ that generates a topology which is different from the topologies generated by any member $\Delta_{\omega_j} \in \mathbf{D}_j, 0 \leq j \leq i$ of the previous families [Goldfarb, 1993]. Thus instead of the single topology of a finite-dimensional vector space we now have an *infinite family of topologies associated*

with the symbolic system. This has a fundamental implication for the inductive learning model - ETS, as discussed in the following section. It should be noted that by enlarging a vector space of n-dimensions to $n+1$ dimensions, the topology of the n-dimensional space (now a subspace) does not change.

5 Some Limitations of the Vector Space Representation

One fundamental limitation of the vector space model, when used in any context, relates to the fact that any chosen input variables cannot be assumed to be commensurate. As discussed in [Goldfarb, 1985] the imposition of the Euclidean distance on the chosen set of measurable variables (features), as is done in ANN models, assumes that all these variables (features) are commensurable. However, as is well known from the theory of special relativity, even such well known variables as the three space coordinates and the time coordinate turn out to be non-commensurate (as realized by Minkowski and Einstein) and consequently required the introduction of Minkowski distance to arrive at a more appropriate mathematical model for space-time. In other words, the moral of this physical theory for us is: even if one assumes a vector space structure, one should not assume that the inner-product vector space generated by any variables is necessarily Euclidean.

The generality of the metric space, as compared to the Euclidean vector space, manifests itself, for example, in the following fact: consider a finite alphabet and the Levenshtein distance defined on a set of strings over the alphabet. Then, four randomly chosen strings cannot be represented isometrically, i.e. preserving the inter-string distances, in the Euclidean vector space of any dimension [Goldfarb, 1985]. In other words, there is no Euclidean vector space of any finite-dimension in which one can find four vectors that have the same inter-distance as the chosen four strings. This implies that the metric information, indispensable for capturing inductive generalization in the symbolic setting, cannot be represented in the finite-dimensional vector space.

Another fundamental limitation of the vector space model when applied to inductive learning in a symbolic environment is connected to *the necessity of ordering the input alphabet (symbols).* Consider biological sequence classification which is an important part of the Genome Project. The input space consists of strings over a finite alphabet. For example, for DNA, $\Sigma = \{g, a, c, u\}$, where each letter represents a DNA nucleotide (or base) [Creighton, 1993]. In order to map the symbolic data into the vector space one must order the input alphabet. This ordering is not related to the symbolic data and since there cannot be any basis for such an ordering, any chosen ordering is arbitrary. Furthermore, *an ordering introduces a topology* which is unique to the vector space [Section 3]. Once a particular ordering has been chosen, a topology for that particular ordering would be introduced. This is because the ordering of the real numbers is responsible for the unique topology of the input vector space. Since the ordering was arbitrary, the induced topology will not be relevant to the structure of the symbolic input, as captured by the symbolic operations of the TS, or equivalently as is present in the symbolic input (biological sequences). As seen in Section 4, this means that any symbolic operation does not have an appropriate interpretation in a vector space.

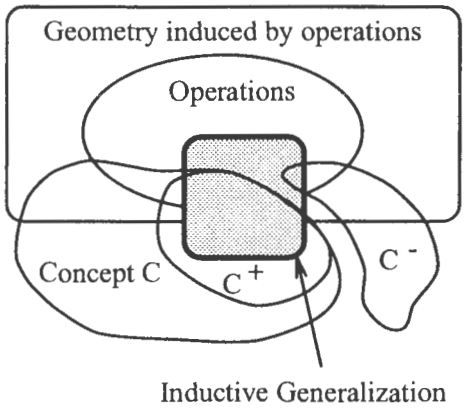


Figure 1: A schematic representation of a formal framework for inductive learning

Ordering of the real numbers (generated by the Peano relation on the natural numbers) is fundamentally linked to the intrinsic structure of $\Re$, while any ordering of the alphabet has nothing to do with the corresponding symbolic structure of the DNA bases.

To recapitulate, under any mapping of a biological sequence (string) into a vector space one must necessarily order the input alphabet. Contrary to the situation with real numbers, the ordering of the input alphabet must, as discussed, be arbitrary and this ordering of the symbols will not be related in any way to the symbolic structure of the input. In other words, while the Peano relation on the natural numbers is relevant (in fact, critical) to generating reals, its analog is absolutely irrelevant to a typical symbolic system as used in AI. This is because symbols usually represent non-numeric object features such as facial or geometric features and the Peano relation when imposed on an alphabet introduces a structure which is inappropriate and misleading. One should note that more appropriate structures/operations for the symbolic systems are those related to the operation of ETS, since insertion/deletion/substitution are the only natural operations for transforming one string into another.

6 A Framework for Inductive Learning and its Implications

Consider the mathematical framework for inductive learning of a single concept shown in Figure 1; the case for multiple concepts fits within the same framework. The main part of the framework is the basic mathematical model consisting of two components: a set of operations, and an induced geometry. For example, as discussed in Section 3, in the vector space model the two components are the algebraic component (viz. algebraic operations) and the geometric component (viz.

the norm on the vector space). On the other hand, for the symbolic model (see Section 4) we have the symbolic component (e.g. string TS) and the corresponding geometric component (e.g. weighted Levenshtein family of distance functions).

In our view, the geometric component in the above framework has often been overlooked in spite of its critical contribution to the inductive learning process. This component is necessary in order to define and capture the idea of inductive generalization. Informally, the geometry acts as a glue that holds the elements of a class together. This fact is reflected in the role of the similarity concept in the area of categorization in psychology [Bourne *et al.*, 1986].

As discussed in Section 2, inductive learning is a process by means of which given a finite positive training set C^+ from a possibly infinite class C, and a finite set of negative training examples C^- from the complement of C, an agent is able to construct an inductive generalization of C. As depicted in Fig. 1, we propose that inductive generalization must be expressed using *both* components of the mathematical model; the operations play the role of "features" in the generalization, while the corresponding distance function plays the role of the "glue that holds the elements of the class together". Formally, the *inductive generalization* is a triple $(\overline{C}^+, \mathbf{S}, \Delta_{\omega^*})$, where $\overline{C}^+$ is a reduced positive training set, $\mathbf{S}$ is the final set of operations and Δ_{ω^*} is the learned distance measure. The elements of $\overline{C}^+$ act as reference patterns for defining the class and consequently a new input pattern is always compared with these reference patterns using the Δ_{ω^*}. The set $\mathbf{S}$ of operations is necessary because the concept of distance can properly be defined only in terms of the operations: this is because the geometry must be consistent with the underlying set of operations. This form of inductive concept representation, or generalization, is in complete agreement with the most accepted theory of concept learning - Rosch's examplar theory."Exemplar theory claims that concept-learning is accomplished by memorizing specific instances and by using some measure of instance similarity" [Bourne *et al.*, 1986].

The proposed framework has important implications even to inductive learning in a vector space. *If the vector space model is recast in the above framework* (see [Goldfarb, 1990a] and example 2 in [Goldfarb, 1990b]), then the set of operations is fixed and, as we have also seen in Section 3, the geometric component is uniquely determined by the set of operations. The most important implication of this fact relates to the form of the learned inductive generalization. According to the proposed framework, the *exact* inductive generalization will be an affine subspace of the vector space. For example, if the vector space is the vector space of functions, the inductive generalization is an affine subspace in the space of functions. It should be noted that since there is only one underlying geometry, this geometry does not change during learning. Consequently, no learning of the geometric component (corresponding to the training set) can occur. Therefore, to start with, the unique geometry of the vector space is simply imposed on the training set.

In fact, the proposed framework was motivated by the desire to allow the training set to generate the geometry appropriate for the inductive generalization of the class. As stated in Section 4, to arrive at inductive generalization in a symbolic setting, we must first learn the geometry corresponding to the training set and, thus, the geometric component begins to play a critical role.

To recapitulate, in the symbolic setting, the set of operations consists of substitution operations and the corresponding geometric component is defined by means of weighted operations. During inductive learning, an agent acquires the necessary new operations (composed from earlier operations) as well as the weights of all the operations [Goldfarb and Nigam, 1994].

We believe that the final set of learned operations represents a communicable and compact form of the inductive generalization and, furthermore, that the numeric component in any model does not represent a communicable part of the inductive generalization. We also strongly feel that communicability is related to the differences between the numeric and symbolic mathematical structures.

We also believe that the essential part of inductive generalization should be its communicability. The communicability refers to that between the various components of the agent as well as between different agents (for example, between a learning machine and a human agent). In the case of symbolic processing in vector space learning machines, the symbols are initially converted to vectors, and therefore the symbolic information (see Section 5) is not present during the learning process. Hence, the results of this learning *cannot contain any symbolic information.*

Thus, the vector space based models (e.g. ANNs) are absolutely incapable of inductive learning in symbolic environments. This is essentially due to the fact that these models construct, as a result of learning, polyhedral regions (in the input vector space) enclosing C^+. According to the proposed definition of inductive generalization such regions are not sufficient to "allow the agent to form an idea about the class". First, there are uncountably many *appropriate choices* for such regions (in a vector space over reals). Second, an agent cannot "form an idea about the class" on the basis of such regions. In contrast, the ETS model is capable of capturing the necessary inductive generalization in symbolic environments and this generalization is communicable within an agent as well as between agents.

7 Conclusion

We have outlined a framework for inductive learning. We have also explicated the fact that symbolic systems are mathematically and computationally quite different from numeric systems. Moreover, we have explained why symbolic information cannot be captured by numeric systems. It should be noted that the situation is no different in the case when a symbolic system is constructed on top of a numeric system.

The implication of the above to vector space models, and to ANNs in particular, is that they cannot perform

symbolic inductive generalization (the result of the inductive learning process).

As proposed in the evolving transformation system (ETS) model, the inductive learning process can capture inductive generalization corresponding to an infinite class by means of *both* the symbolic information (essentially the final set of operations constructed during the learning process) and numeric information (the weights attached to the operations).

In conclusion, we think that for any agent (artificial and, quite possibly, biological) engaged in inductive learning the symbolic component must play the central role. Otherwise, the agent simply will not be able to achieve inductive generalization.

References

[Bourne *et al.*, 1986] L. E. Bourne, R. L. Dominowski, E. F. Loftus, and A. F. Healy. *Cognitive Processes.* 2nd Ed., Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[Creighton, 1993] T. E. Creighton. *Proteins - Structures and Molecular Properties.* W.H. Freeman and Co., New York, NY, 1993.

[Godement, 1968] R. Godement. *Algebra.* Houghton Mifflin Company, Boston, MA, 1968.

[Goldfarb, 1985] L. Goldfarb. A New Approach to Pattern Recognition. In *Progress in Pattern Recognition 2*, eds. L. N. Kanal and A. Rosenfield, North-Holland, pages 241–402, 1985.

[Goldfarb, 1990a] L. Goldfarb. On the Foundation of Intelligent Processes - I. An Evolving Model for Pattern Recognition, *Pattern Recognition*, 23:596–616, 1990.

[Goldfarb, 1990b] L. Goldfarb. A Unified Metric Model for Pattern Learning. In *Proc. IASTED Int. Symp. on Machine Learning and Neural Networks*, eds. M.H. Hamza, New York, 10-11 Oct, pages 96–99, 1990.

[Goldfarb, 1992] L. Goldfarb. What is distance and why we need the Metric Model for Pattern Learning. *Pattern Recognition*, 25(4):431–438, 1992.

[Goldfarb, 1993] L. Goldfarb. On some mathematical properties of the ETS model. Technical Report TR93-079, Faculty of Computer Science, UNB, September 1993.

[Goldfarb and Nigam, 1994] L. Goldfarb and S. Nigam. The Unified Learning Paradigm - A Foundation for A.I. *Artificial Intelligence and Neural Networks: Steps Towards Principled Integration*, eds. V. Honovar and L. Uhr, to appear, Academic Press.

[Johnson-Laird, 1988] P. N. Johnson-Laird. *The Computer and the Mind - An Introduction to Cognitive Science.* Harvard University Press, Cambridge, MA, 1988.

[Jordan, 1988] M. I. Jordan. An Introduction to Linear Algebra in Parallel Distributed Processing In *Parallel Distributed Processing*, eds. D. E. Rumelhart and J. L. McClelland, pages 365–422, 1988.

[Kruskal and Sankoff, 1983] J. B. Kruskal and D. Sankoff. eds. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley Publishing Company Inc., Reading, MA, 1983.

[Lipmann, 1987] R. P. Lipmann. An Introduction to Computing with Neural Networks. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[Mitchell, 1982] T. M. Mitchell. Generalization as search, *Artificial Intelligence*, 18(2):203–236, 1982.

[Taylor, 1987] A. E. Taylor. *Introduction to Functional Analysis*, John Wiley and Sons Inc., New York, NY, 1987.

The Problem of Small Disjuncts: its remedy in Decision Trees

Ting, Kai Ming
Basser Department of Computer Science
University of Sydney, NSW 2006, Australia
E-mail: kaiming@cs.su.oz.au

Abstract

For learning systems that describe a learned concept as a disjunction of conjunctions of conditions, small disjuncts are disjuncts which cover a small number of training instances and often entail high error rates. This paper investigates the problem of small disjuncts in decision trees and proposes a solution using a composite learner which consists of instance-based algorithms [Aha, 1990; Aha et al, 1991] and C4.5 [Quinlan, 1993].

Holte, Acker and Porter's [1989] findings on the problem of small disjuncts have motivated this investigation. Their findings are: (a) the choice of the learning system's bias (maximum generality) has been the main cause that creates the problem of small disjuncts, and (b) it is difficult to eliminate the error-prone small disjuncts without affecting the performance of large disjuncts. The proposed composite learner is an approach that uses the maximum specificity bias without having any impact on the performance of large disjuncts.

This paper also explores four definitions of small disjuncts. The composite learner and the definitions of small disjuncts are examined using a set of benchmark domains [Zheng, 1993].

1 Introduction

One of the current research areas in machine learning is integrating different kinds of representations into a single learning system. Early work concentrated on systems based on single representations such as decision trees [Breiman et al, 1984; Quinlan, 1986], rule-based systems [Michalski et al, 1986; Clark and Niblett, 1987], instance-based methods [Aha and Kibler, 1989; Aha, 1990; Aha et al, 1991] and neural networks [Rumelhart et al, 1986]. Numerous experiments (e.g. [Mooney et al, 1989; Weiss and Kapouleas, 1989]) have shown that no system based on a single representation can clearly excel in all domains. Characterizing the strong points of each representation and finding ways to integrate them have emerged as an important research direction. [Brodley, 1993] has integrated three models (univariate tests, linear discriminants and instance-based classifiers) in a decision tree structure. The MCS system produces comparable results to the best results of the systems using a single representation.

We approach the same problem from a rather different perspective. Beginning with a seemingly unrelated problem of small disjuncts [Holte et al, 1989], a composite learner that consists of decision trees and instance-based methods is proposed. It seeks to characterize capabilities of these differing representations insofar as they impact on the problem of small disjuncts.

We first discuss the problem of small disjuncts in the following section. A brief introduction of the composite learner is given in Section 3 and followed by a report of a series of experiments and their evaluations. Future work and conclusion are described in the last section.

2 The Problem of Small Disjuncts

The problem of small disjuncts was first introduced and explored by [Holte et al, 1989]. For learning systems that describe a learned concept as a disjunction of conjunctions of conditions, small disjuncts are disjuncts which cover a small number of training instances and often entail high error rates. On the other hand, large disjuncts cover a large proportion of the training instances and have low error rates. Improving the poor predictive accuracy of small disjuncts becomes the crux of the problem.

Holte and his colleagues used CN2 [Clark and Niblett, 1987], a learning system derived from AQ [Michalski et al, 1986] and ID3 [Quinlan, 1986], to explore the problem of small disjuncts. They make the following observations:

i The choice of the learning system's bias (maximum generality) is the main cause, if not the only one, that creates the problem of small disjuncts. Changing the learning system's bias, from maximum generality to maximum specificity for small disjuncts, successfully improves their predictive accuracy.

ii It is difficult to eliminate the error-prone small disjuncts without affecting the performance of large disjuncts. When the accuracy of small

disjuncts is improved by more specific rules using maximum specificity bias, these rules cover less instances than the original rules. The instances covered by previous maximum generality small disjunct rules but not by the maximum specificity rules would have to be covered by large disjuncts or default rule. This has an adverse effect on large disjuncts and default rule that reduces their accuracies. As a result, the rules induced from two different bias has a total accuracy either comparable to or worse than the rules induced from the maximum generality bias only.

The experiment also raises another important issue, namely the definition of small disjuncts. Holte and his colleagues define small disjuncts using absolute number of covered instances. For example, small disjuncts can be defined as those disjuncts that cover five training instances or less. This poses a problem as the definition would be expected to change when the number of training instances changes; and it would also be different for different domains. We explore this particular problem in Section 4.2.

3 The Composite Learner

Holte and his colleagues' exposition and experiments have prompted the use of another representation in solving the problem of small disjuncts. If maximum specificity bias is the way to overcome the problem of small disjuncts, then instance-based learning methods [Aha and Kibler, 1989; Aha, 1990; Aha et al, 1991] would be the best remedy. Instance-based learning (IBL) algorithms do not learn rules in the form of a decision tree, a set of rules or a network. Instead, IBL stores the training instances and predicts the class of the stored instance that is nearest (according to some distance metric) to the test instance. Thus, the specific instances are used in classification rather than generalized rules.

We summarize our hypothesis as follows:

Instance-based methods have higher predictive accuracies than generalized rules in small disjuncts.

A composite learner that consists of C4.5 [Quinlan, 1993] and IB1 [Aha, 1990] is used to test the above hypothesis. The two learning systems are trained independently during the training process. In classification, the decision tree induced by C4.5 is used to decide whether C4.5 or IB1 shall be employed in actual classification. The composite learner will use IB1 for classification if the test instance belongs to a small disjunct, and use C4.5 otherwise. Thus, the same decision tree may produce different decisions depending on the definition of small disjuncts.

The composite learner 'kills two birds with one stone'. First, the instance-based method is the furthest one can go with maximum specificity bias. Second, the composite learner has completely overcome the difficulty of eliminating the error-prone small disjuncts without affecting the performance of large disjuncts. Simply replacing poor performance small disjuncts with instance-based methods leaves large disjuncts intact.

The first thing we have to do now is to verify the hypothesis. The experiment and its results are presented in Section 4.1.

4 Experiments

The experiments are conducted using twelve benchmark domains [Zheng, 1993] derived from the UCI Repository of Machine Learning database. Each experiment with a specific definition of small disjuncts is conducted over 50 runs with randomly selecting 90% of the instances for training and using the remaining 10% as test data, except in the monks-2 domain where a separate set of training data (169) and testing data (432) are given. All the results shown in the following sections are using the pruned trees of C4.5, unless otherwise stated; and the results will be presented as the difference with respect to the performance of C4.5.

The characteristics of the experimental domains and the performance of C4.5 are given in Table 1. All error rates are calculated on the test data.

Domain	*C4.5*	*#Ex*	*#Cl*	*#AT*
lymphography	22.1%	148	4	9B+9N
breast(bcw)	5.4%	699	2	9C
promoter	22.6%	106	2	57N
soybean	8.1%	683	19	16B+19N
monks-2	35.0%	169-432	2	2B+4N
nettalk	19.7%	5438	52	7N
diabetes	29.3%	768	2	8C
hypothyroid	0.8%	3163	2	18B+7C
hepatitis	23.5%	155	2	13B+6C
LED7	28.6%	200	10	7B
LED24	37.9%	200	10	24B
waveform	30.8%	300	3	40C

C4.5: Error rate of C4.5, #Ex: Number of examples,
#Cl: Number of classes,
#AT: Number of attributes and types.
B: Binary, N: Nominal, C: Continuous.

Table 1: Details of experimental domains and performance of C4.5

4.1 Verifying the Hypothesis

For the first experiment, the simplest and straightforward definition of small disjuncts is used, i.e. disjuncts that cover less than or equal to a fixed number of the training instances. We examine a few such definitions by varying this number from 1 to 29; for each small disjunct definition, 50 trials are conducted with the composite learner.

Figure 1 and Figure 2 present the experimental results for the promoter and LED7 domains respectively. Four graphs are plotted for each domain with respect to the twenty nine small disjunct definitions. The first graph shows the difference in error rate between C4.5 and IB1 on small disjuncts; a positive difference indicates that IB1 is more accurate than C4.5. The second graph shows the percentage of the test data that belongs to small disjuncts. The difference in error rate between C4.5 and IB1, if used separately, is shown in the third graph. The

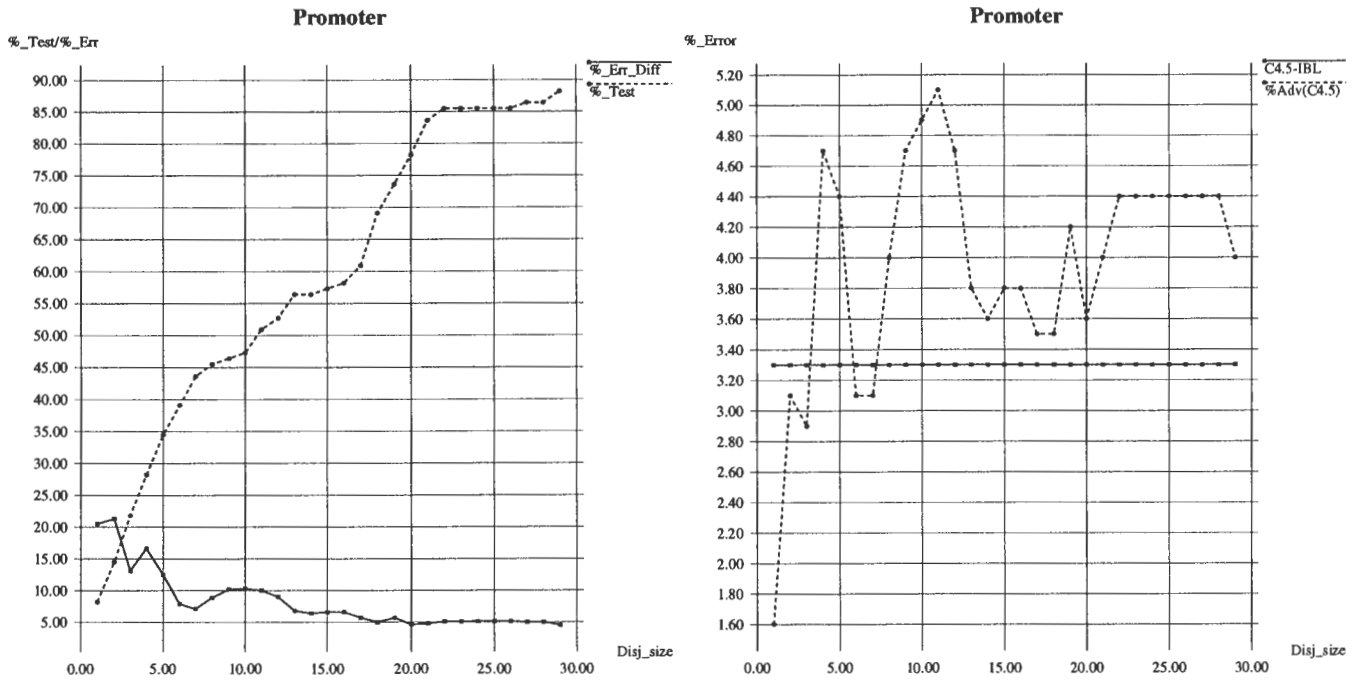


Figure 1: Promoter domain. Two graphs on the left show the difference in error rate between C4.5 and IB1 on small disjuncts, and the percentage of test data that belongs to small disjuncts. The performance difference between C4.5 and IB1 is shown by the straight line graph on the right. The remaining graph illustrates the performance difference between C4.5 and the composite learner.

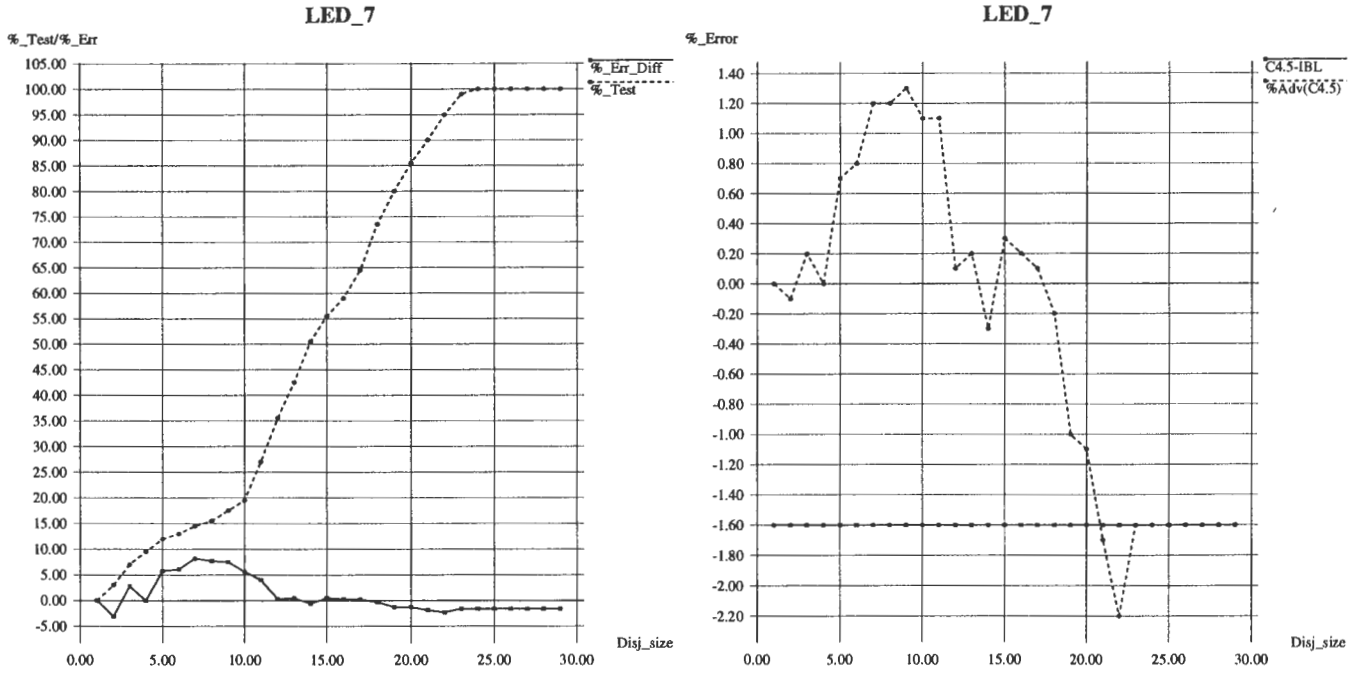


Figure 2: LED7 domain

Domain	*IB1>C4.5*	*R=4%*	*E=35%*	*R=4% or E=35%*	*T=30%*	*T=30% or E=20%*	*DE*	*Largest Improv*
lymphography	N(+0.4)	-1.6	-1.9	-2.0	-2.7	-1.9	-2.1	-3.3
bcw	Y(-0.9)	-1.3	-0.4	-1.3	-1.3	-1.3	-1.3	-1.5
promoter	Y(-3.3)	-2.9	-4.2	-4.2	-4.2	-5.6	-6.9	-6.9
soybean	N(+0.4)	+0.8	-0.3	+0.8	+0.2	+0.2	-0.3	-0.6
monks-2	Y(-5.4)	-3.0	-5.1	-5.1	-3.0	-5.4	-4.6	-6.0
nettalk	N(+6.4)	+6.4	-0.1	+6.4	+1.9	+1.3	+0.5	-0.7
diabetes	N(+0.5)	-2.8	-2.1	-2.9	-3.0	-2.9	-2.6	-3.5
hypothyroid	N(+1.9)	+0.5	0.0	+0.5	+1.5	+1.5	+1.5	0.0
hepatitis	Y(-4.2)	-0.5	-0.8	-1.0	-1.6	-2.4	-2.2	-4.6
LED7	N(+1.6)	-1.2	+1.4	+1.4	-0.6	+2.5	+1.5	-1.5
LED24	N(+31.5)	+0.7	+8.0	+10.0	+4.4	+21.4	+9.8	0.0
waveform	N(+5.7)	-0.7	-1.3	-0.9	-0.8	-1.3	-0.8	-1.9

Table 2: Experimental results of composite learner

last graph presents the performance difference of using the composite learner as compared to C4.5.

The results of these two domains clearly demonstrate that the instance-based method outperforms C4.5 in small disjuncts. In the LED7 domain, even though IB1 alone is worse than C4.5 by 1.6%, the composite learner in which IB1 is used to represent small disjuncts achieves a maximum overall advantage of 1.3% over C4.5. Both results show that the composite learner can be a better predictor than either C4.5 or IB1.

The above experiment supports the hypothesis. However, it is not clear what is the optimum size limit for small disjuncts in one domain. It is also anticipated that the absolute size limit will differ from one domain to another and when the training data size changes. The next experiment is designed to investigate a more appropriate small disjunct definition.

4.2 The Definition of Small Disjuncts

Four measures have been considered for the definition of small disjuncts, which based on:

i Absolute disjunct size.

ii Relative disjunct size.

iii Percentage of training data coverage.

iv Disjunct error rate.
   a. fixed error rate.
   b. decision tree's estimated error rate.

The second measure attempts to solve the problem of using an absolute disjunct size measure when the training data size changes. In this case, a fixed percentage of training data size is used to define a small disjunct. The third measure requires that the total coverage of all small disjuncts shall not be more than a fixed percentage of the total training data. The use of disjunct error rate to define small disjuncts requires a re-orientation of the meaning of "small" disjuncts. To be precise, it means "poor performance" disjuncts rather than small disjuncts and is not determined by size alone. As result of the last measure, there are two situations where some small coverage disjuncts are not being regarded as small disjuncts. First, some small disjuncts have low estimated error rates, and second, disjuncts which have the same size, possess differing estimated accuracies. While some disjuncts are being regarded as small disjuncts, others are being treated as large disjuncts; though all of them have the same coverage. These situations do not arise for the first three measures.

The four corresponding small disjunct definitions based on these measures are:

i *the disjunct size is less than or equal to a fixed number(F).*

ii *the disjunct size is less than or equal to a fixed percentage(R) of the training data size.*

iii *the total small disjuncts' coverage is less than or equal to a fixed percentage(T) of the training data size.*

iv a. *the disjunct error rate is more than or equal to a fixed error rate(E).*
   b. *the disjunct error rate is more than or equal to decision tree's estimated error rate(DE).*

The experimental results of the composite learner using the above small disjunct definitions are tabulated in Table 2. Note that the results stated are the difference with reference to the error rate of C4.5 listed in Table 1. For example, the second column shows that IB1 has error rate 0.4% more than that of C4.5 in the lymphography domain, and likewise for the other domains. Note that the definitions listed in the table are determined from a series of experiments by varying the fixed number in a way similar to the experiments in Section 4.1. A particular definition is chosen based on the performance in all twelve domains. For example, E=35% is selected as it enables the composite learner to produce the best results across all twelve domains. The best result (with respect to all experiments) for each individual domain is listed in the last column of Table 2.

There are a few reasons why some domains do not conform to the hypothesis. First, IB1 is known to have difficulty with domains that contain irrelevant attributes [Aha, 1990; Aha et al, 1991] such as the LED24 and

Domain	*IBL > C4.5*	*% Difference.*	*T=30%*	*DE*	*DE(unpruned)*
lymphography	N(+0.4)	+1.8	-1.9	-2.1	-1.1
bcw @	Y(-0.8)	-14.8	-1.4	-1.7	-1.6
promoter	Y(-3.3)	-14.6	-5.6	-6.9	-7.7
soybean	N(+0.4)	+4.9	+0.2	-0.3	-0.5
monks-2	Y(-5.4)	-15.4	-5.4	-4.6	-5.4
nettalk	N(+6.4)	+32.5	+1.3	+0.5	+0.6
diabetes @	N(+1.0)	+3.4	-2.7	-1.9	-1.9
hypothyroid	N(+1.9)	+237.5	+1.5	+1.5	+1.7
hepatitis	Y(-4.2)	-17.9	-2.4	-2.2	-4.1
LED7 #	Y(-0.9)	-3.1	-2.8	-1.6	-1.8
LED24 @	N(+0.3)	+0.8	-5.7	-3.5	-3.8
waveform @	N(+1.5)	+4.9	-3.2	-3.0	-3.6

#: IB3, @: IB4, (unpruned): Unpruned trees.

% Difference: Percentage difference in error rate wrt that of C4.5.

Table 3: Results of the composite learner with selection rule

waveform domains[1]. Noise could be another factor. In view of these factors, we have attempted the same experiments with IB3 and IB4[2] [Aha, 1990; Aha et al, 1991] acting as the small disjunct classifier in the composite learner. This experiment prompted the use of rule to select a small disjunct classifier from the three algorithms. We report this investigation in the following section.

Additionally, we also tried to narrow down the number of small disjunct definitions to three, namely, T=30%, T=30% or E=20%, and DE. Experiments are repeated with these definitions using two thirds of the data as training and one third as test (results not shown in this paper). Similar trends are observed in all domains. This shows that the definitions are robust across different data sizes and domains.

4.3 Composite Learner with Selection Rule

The experimental results using IB1, IB3 or IB4 as the small disjunct classifier show that no single algorithm can perform well in all these benchmark domains. One can perform well in some domains where the others fail and vice versa. However, it seems to exhibit some relationships between the performance of different types of IBL and the domain that has specific attribute type. An attempt to construct a rule that depicts this relationship has been made and it is shown in Figure 3.

The experimental results obtained by applying this rule are listed in Table 3. As in Table 2, the figures are the differences with reference to the error rates of C4.5. Two small disjunct definitions have been used; they are T=30% (IB1 uses T=30% or E=20% definition instead, as shown in the rule) and DE, and their results are listed in the fourth and fifth columns of Table 3. Using the unpruned trees of C4.5 with DE definition has also been tried and the results are shown in the last column.

With the T=30% small disjunct definition, the composite learner outperforms C4.5 in nine domains and is marginally worse in the other three domains. Out of the nine domains, the composite learner performs better than both IBL and C4.5 in seven domains, and performs equally well and worse than IBL in one domain respectively.

```
Define small disjunct: T=30%

If real-valued domain or known to have irrelevant
attributes
    then use IB4 as small disjunct classifier
else if binary domain
    then use IB3 as small disjunct classifier
else
    use IB1 (T=30% or E=20%) as small disjunct
    classifier
```

Figure 3: Selection rule for the composite learner

Using the estimated error rate of the decision tree as the small disjunct definition, the composite learner outperforms C4.5 in ten domains (the soybean domain has only marginal gain) and is marginally worse in the other two domains. The composite learner performs better than both IBL and C4.5 in eight domains, and performs worse than IBL in two domains. Comparing this definition with T=30%, where the composite learner performs better than C4.5 in the same nine domains, the definition T=30% has better results in six domains and worse in three domains.

Generally, the composite learner that uses unpruned trees is marginally more accurate than that using pruned trees. With the DE(unpruned) definition, the composite learner achieves the best results in the promoter, monks-2, soybean, hepatitis and waveform domains. This result

[1]IB1 uses a separate metric for nominal and real-valued attributes. The simple metric used for nominal attributes can produce poor performance. See [Cost and Salzberg, 1993] for a better metric for nominal attributes.

[2]IB3 is a noise-tolerant version and IB4 is built on top of IB3 with an extra capability of dealing with irrelevant attributes.

may suggest that C4.5 could have overpruned in some of these domains.

The composite learner has performed better than C4.5 in one way or another in every domain, except for the hypothyroid and nettalk domains; and with DE(unpruned) definition, it performs better than IBL in all domains except in the monks-2 and hepatitis domains, where equal performance is achieved.

The overall results seem to favour using the DE small disjunct definition and unpruned trees.

The third column of Table 3 shows the performance difference with respect to that of C4.5. It indicates that there is no advantage in combining IBL and C4.5 if the difference in performance is more than 30%. In order to obtain any actual gain in prediction accuracy by combining two learning algorithms, the difference in performance must not be large. Because we split the description space into two regions and by using only one learning algorithm in each region, we expect the algorithm is superior in its own region. This can only be achieved if the overall performance of the two algorithms do not differ substantially. If they do, the superior algorithm is most likely to outperform the other one in both regions of the description space; thus, render no advantage in combining the two algorithms.

5 Discussion

Although the preliminary results support the hypothesis, the real success of the composite learner depends heavily on the definition of small disjuncts. The definition based on T=30% represents "small" disjuncts, and the definition based on E=35% represents "poor performance" disjuncts. The disjunction of these two definitions, i.e. T=30% or E=20% covers both types of disjuncts. A still better "poor performance" disjunct representation seems to be the definition based on the decision tree's estimated error rate. Though these definitions are superior to the definition based on the absolute disjunct size, there is no definition which is a clear winner across all the benchmark domains.

The selection rule, shown in Figure 3, is by no means the best rule for the composite learner. Because it only makes use of one kind of information, i.e. domain attribute type, the rule is bound to fail for some new domains. Nevertheless, it successfully shows that the composite learner can be a better learner than its constituent parts.

6 Related Work

[Quinlan, 1991] demonstrates that small disjuncts associated with classes of different relative frequencies are likely to have different accuracies. Based on this finding, he gives improved estimates for the accuracy of small disjuncts by using an adaptation of the Bayes-Laplace formula.

Work by [Danyluk and Provost, 1993] has exposed the fact that small disjuncts are due to exceptions and rare cases as well as noise. However, there is no simple method to differentiate between exceptions and noise.

MCS [Brodley, 1993] uses a hand-crafted rule to guide the construction of different models in the nodes of a decision tree. This work has motivated the use of a selection rule for the composite learner. The major difference between the two systems is that MCS produces a multi-model tree and each model is trained on part of the total training data; whereas the two algorithms in the composite learner are trained independently on the total training data and work cooperatively according to the definition of small disjuncts. Both apply a rule for model/algorithm selection; MCS selects among three models (univariate tests, linear discriminants and instance-based classifiers) whereas the composite learner selects among three types of instance-based methods.

7 Future Work and Conclusion

The key idea of this work is to use instance-based methods to solve the problem of small disjuncts in decision trees. We have successfully shown that the composite learner that consists of C4.5 and one of the instance-based algorithms (IB1, IB3 or IB4) is a promising approach. We would expect this will also work well in rule-based systems such as C4.5rules and intend to verify this in the near future.

Our hypothesis is based on Holte et al's [1989] findings that the maximum specificity bias should be used for small disjuncts. Though it has been supported by the empirical results, there is no reason to believe that it is the only bias one should apply. This opens another area for further investigation. Other systems such as neural nets might be used instead of instance-based algorithms.

Another important issue that we have explored is the definition of small disjuncts. Three definitions are found to be robust across domains and varying data sizes. However, the investigation is by no means exhaustive and there is still room for future work.

Acknowledgements

This research was partially supported by an Australia Research Council grant (to J.R.Quinlan) and by a research agreement with Digital Equipment Corporation. This author is partially supported by Equity and Merit Scholarship Scheme. Numerous discussions with J.R. Quinlan, Mike Cameron-Jones, Zijian Zheng, Ivan Bratko and Alen Varšek have provided much insight into this problem. They have also helped to improve the clarity of the previous version of the paper (which contained wrong results regarding unpruned trees due to a mistake in the experiment). Thanks to J.R. Quinlan and David W. Aha for providing C4.5 and IBL.

References

[Aha and Kibler, 1989] David W. Aha and D. Kibler. Noise-Tolerant Instance-Based Learning Algorithms, in *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 794-799, Morgan Kaufmann.

[Aha, 1990] David W. Aha. *A Study of Instance-Based Algorithms for Supervised Learning Tasks*, PhD The-

sis, Department of Information and Computer Science, University of California, Irvine, Technical Report 90-42.

[Aha et al, 1991] David W. Aha, D. Kibler and M.K. Albert. Instance-Based Learning Algorithms, *Machine Learning, 6*, pages 37-66.

[Breiman et al, 1984] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone (1984), *Classification And Regression Trees*, Belmont, CA: Wadsworth.

[Brodley, 1993] Carla E. Brodley. Addressing the Selective Superiority Problem: Automatic Algorithm/Model Class Selection, in *Proceedings of the Tenth International Conference on Machine Learning*, pages 17-24, Morgan Kaufmann.

[Clark and Niblett, 1987] Peter Clark and T. Niblett. Induction in Noisy Domains, In I. Bratko and N. Lavrač, editors, *Progress in Machine Learning*, pages 11-30. Sigma Press.

[Cost and Salzberg, 1993] Scott Cost and S. Salzberg. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features, *Machine Learning, 10*, pages 57-78.

[Danyluk and Provost, 1993] A.P. Danyluk and F.J. Provost, Small Disjuncts in Action: Learning to Diagnose Errors in the Local Loop of the Telephone Network, in *Proceedings of the Tenth International Conference on Machine Learning*, pages 81-88, Morgan Kaufmann.

[Holte et al, 1989] Robert C. Holte, L.E. Acker and B.W. Porter. Concept Learning and the Problem of Small Disjuncts, in *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pages 813-818, Morgan Kaufmann.

[Michalski et al, 1986] Ryszard S. Michalski, I. Mozetič, J. Hong and N. Lavrač. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains, in *Proceeding of the Fifth National Conference on Artificial Intelligence*, pages 1041-1045.

[Mooney et al, 1989] Ray Mooney, J. Shavlik, G. Towell and A. Gove. An Empirical Comparison of Symbolic and Connectionist Learning Algorithms, in *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pages 775-780, Morgan Kaufmann.

[Quinlan, 1986] John Ross Quinlan. Induction of Decision Trees, *Machine Learning, 1*, pages 81-106.

[Quinlan, 1991] John Ross Quinlan. Improved Estimated for the Accuracy of Small Disjuncts, *Machine Learning, 6*, pages 93-98.

[Quinlan, 1993] John Ross Quinlan. *C4.5: Program for machine learning*, Morgan Kaufmann.

[Rumelhart et al, 1986] D.E. Rumelhart, G.E. Hinton and R.J. Williams. Learning Internal Representations by Error Propagation, in *Parallel Distributed Processing, Vol.1,* Eds. D.E. Rumelhart & J.L. McClelland, MIT Press, MA.

[Weiss and Kapouleas, 1989] Sholom M. Weiss and I. Kapouleas. An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods, in *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pages 781-787, Morgan Kaufmann.

[Zheng, 1993] Zijian Zheng. A Benchmark for Classifier Learning, in *Proceedings of the Sixth Australian Joint Conference on Artificial Intelligence*, pages 281-286, World Scientific.

Learning Default Concepts

Dale Schuurmans
Department of Computer Science
University of Toronto, Toronto, ON M5S 1A4
dale@cs.toronto.edu

Russell Greiner
Siemens Corporate Research
Princeton, NJ 08540
greiner@learning.siemens.com

Abstract

Classical concepts, based on necessary and sufficient defining conditions, cannot classify logically insufficient object descriptions. Many reasoning systems avoid this limitation by using "default concepts" to classify incompletely described objects. This paper addresses the task of learning such *default concepts* from observational data. We first model the underlying performance task — classifying incomplete examples — as a probabilistic process that passes random test examples through a "blocker" that can hide object attributes from the classifier. We then address the task of learning accurate default concepts from random training examples. After surveying the learning techniques that have been proposed for this task in the machine learning and knowledge representation literatures, and investigating their relative merits, we present a more data-efficient learning technique, developed from well-known statistical principles. Finally, we extend Valiant's PAC-learning framework to this context and obtain a number of useful learnability results.

1 Introduction

Many reasoning tasks involve "classification" [Cla85] — *i.e.*, determining whether a particular object belongs to a specified class, given a description of that object. For example, a diagnosis process must determine whether a patient, with a specified set of symptoms, has a particular disease; a chess player must determine whether a particular move is appropriate given a board configuration; and a planner must determine whether to apply a particular action, given the perceived state. Many classifiers are based on *classical concept definitions* (ccd s), which specify necessary and sufficient conditions for concept membership. While these systems can work effectively when given *completely specified* objects (*e.g.*, a complete description of the patient's symptoms, etc.), they may be unable to categorically classify objects that are only *partially* described.

Unfortunately, we may still have to provide a classification for such partially-described domain objects. For example, as doctors seldom have access to every potentially relevant fact about a patient, they usually cannot rule out all but the one true disease. The patient is usually better off if the doctor makes a credulous assessment and suggests some treatment based on what is known, rather than skeptically withholding judgement.

Notice that the doctor's diagnosis can *change* if he receives further information about the patient. As this type of *nonmonotonic* classification behavior cannot be described in terms of necessary and sufficient conditions, it cannot be encoded as a ccd. There are, however, formalisms designed to classify partial object descriptions. *Default concept definitions* (dcd s) are a natural generalization of ccds, which avoid this limitation by using *default* classification rules [Rei87]. These classifiers play an important role in many expert systems [Cla85, PBH90].

Of course these dcds must somehow be acquired for such applications. As it is often quite difficult to explicitly extract the knowledge of domain experts, it makes sense to use machine learning techniques to automatically acquire the appropriate default concept based on existing "solved" cases; *cf.*, [PBH90]. Unfortunately, the task of learning default concept definitions has received relatively little attention, especially when compared to the vast literature on the subject of learning to classify *complete* object descriptions. To date, only a few empirical studies have been published [PBH90, Qui89, BFOS84], and the problem has yet to receive an adequate theoretical treatment in the machine learning literature; *cf.*, [Riv87, p.245]. This means there is no supporting theory that specifies when proposed techniques can be expected to perform well, or even why they work when they do.

We attempt to fill this void by studying the problem of learning accurate default concepts from examples within a precise mathematical framework. As preliminaries, Section 2 first defines the formal structure of default concepts and the associated object level classification task, and Section 3 introduces a probabilistic testing model that incorporates "attribute blocking". Section 4 then considers the problem of *learning* accurate dcds from random training examples: It considers learning under a relatively benign (resp., completely general) blocking model, introduces many of the existing learning techniques discussed in the literature, and considers an alternative procedure (relatively unknown

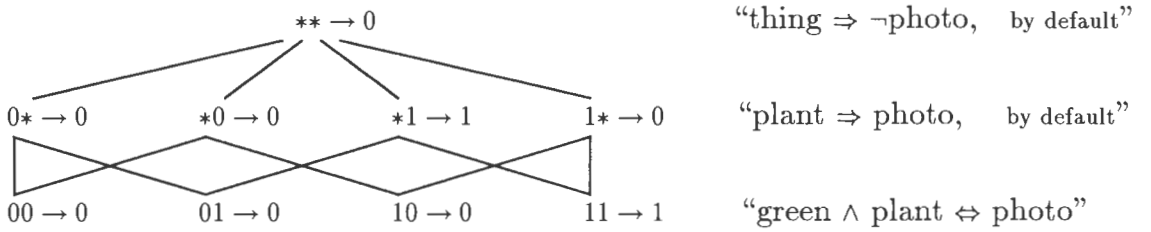


Figure 1: Structure of a complete default concept definition

in machine learning community) that is based on well-known ideas from theoretical statistics. It also extends Valiant's PAC-learning framework to the present case: assessing the effects of prior knowledge on learning efficiency, and determining the difficulty of learning under different conditions.[1]

We first close this introduction by tying this research to existing work: Notice first that, while there is a voluminous literature on default and nonmonotonic reasoning [Rei87], and even a recent trend towards probabilistic interpretations of default logics [Pea88, Bac90], the issue of *learning* defaults has scarcely been raised. Second, to avoid possible confusions, it is worth explicitly distinguishing our "missing attribute" framework from two other models of learning from the learnability community: A system that learns with attribute noise [SV88] does not know which attribute values have been corrupted; by contrast, we know explicitly which values are missing. Also, a probabilistic concept [KS90] is a mapping $c_i : X_n \mapsto [0,1]$ from the space of *complete* object descriptions X_n to probability values; such mappings do not directly handle missing attribute values.

2 Default Concepts

Following standard practice, we consider a set of domain objects $X_n = \{0,1\}^n$, where each object is identified by a vector of boolean attributes $\bar{x} = \langle x_1, \ldots, x_n \rangle$. A (complete) test example is specified by a pair $\langle \bar{x}, c \rangle$, consisting of a domain object $\bar{x}$ and its true classification c. In standard classification models, this domain object $\bar{x}$ would be passed "as is" to the classifier before testing its classification against the correct class c. Here, however we assume the classifier only sees a "degraded version" of $\bar{x}$ in which certain attribute values have been replaced by the "unknown" value $*$; see Figure 2. We model this degradation using a (stochastic) *blocking process* $\beta : X_n \times \{0,1\} \rightarrow \{0,1,*\}^n$ that may "hide" some of the attribute values: replacing certain values with $*$, but otherwise leaving $\bar{x}$ intact. Thus, $x_i = 0$ can be mapped to $x_i^* \in \{0,*\}$, and $x_i = 1$ to $x_i^* \in \{1,*\}$. We let $X_n^* = \{0,1,*\}^n$ denote the set of possible object *descriptions*. A *test example* $\langle \bar{x}^*, c \rangle$ is a (partial) description $\bar{x}^*$ of some domain object $\bar{x}$, along with $\bar{x}$'s true classification $c \in \{0,1\}$. The space of possible examples is denoted $X_n^* \times \{0,1\}$.

A *classical concept definition* (ccd) is a subset of X_n, which we represent by its indicator function $c : X_n \rightarrow \{0,1\}$; thus $c(\bar{x}) = 1$ iff $\bar{x}$ belongs to the concept. A *default concept definition* (dcd) $d : X_n^* \rightarrow \{0,1\}$, on the other hand, takes a *description* $\bar{x}^*$ as its input and returns $d(\bar{x}^*) = 1$ if the object described by $\bar{x}^*$ belongs to the concept *by default*, and returns 0 otherwise. Given a test example $\langle \bar{x}^*, c \rangle$, a dcd d makes a *correct* classification if $d(\bar{x}^*) = c$, otherwise it makes an *error*.

We can represent a dcd d as a collection of *rules* of the form $\bar{x}^* \rightarrow c$ where $c \in \{0,1\}$ and $\bar{x}^* \rightarrow c \in d$ means $d(\bar{x}^*) = c$. By insisting that for every object description $\bar{x}^* \in X_n^*$ either $\bar{x}^* \rightarrow 1 \in d$ or $\bar{x}^* \rightarrow 0 \in d$ but not both, we are in effect only considering *complete* dcds that categorically classify every possible object description, even $\bar{x}^* = \langle *, *, \ldots, * \rangle$.[2] To illustrate, consider the example of a dcd on two attributes shown in Figure 1, where the first attribute is "green", the second "plant", and the class is "photosynthetic".[3] Notice this collection of rules specifies *nonmonotonic* classification behavior, as its assessment of concept membership can change as more attributes are specified. For example, even though non-green-plants $\subset$ plants $\subset$ things, the predicted photosynthesis properties are 0, 1, 0, respectively. Such a classifier cannot be specified by a classical concept.[4]

There are many unexpected similarities between dcds and existing nonmonotonic knowledge representation formalisms. For example, Reiter [Rei87] considers commonsense concepts like "bird", "chair", and "game" and notes that they do not have classical definitions in terms of necessary and sufficient conditions. He argues that these concepts can be better characterized by specifying "default" necessary and sufficient conditions, and shows that this idea is similar to Minsky's concept of *frames* [Min75]: frame selectors can be viewed as "default" sufficient conditions for the frame concept, and

[1] Unfortunately, space constraints preclude presenting proofs of the results stated in this abstract; see [Sch94].

[2] Thus there are 2^{3^n} distinct dcds possible on n boolean attributes. Only some of these have "reasonable" structures, see Lemma 1 below.

[3] Each node in the graph represents a rule; *e.g.*, "$*1 \rightarrow 1$" encodes the rule that plants, of unspecified color, are accepted in the photosynthetic class. An arc descending from node n_1 to n_2 means the antecedent of n_1's rule is "more general" than n_2's antecedent, in that any object that matches n_2's antecedent will also match n_1's.

[4] Notice the blocking process β introduces only a restricted form of ambiguity: β may produce descriptions corresponding to disjunctions like $0* \equiv 00 \vee 01$, but cannot produce a description corresponding to $01 \vee 10$ (this is reminiscent of [BE89]) — *i.e.*, it cannot express the claim that an object is "either a non-green plant or a green non-plant". This will restrict the type of "reference classes" we must consider when learning dcds; see Footnote 7 below.

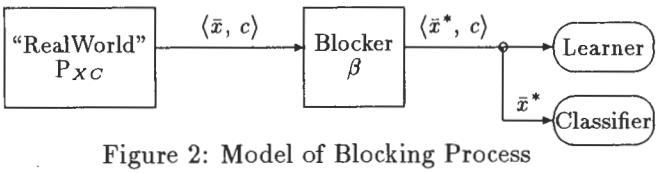


Figure 2: Model of Blocking Process

frame instantiations can be viewed as "default" necessary conditions. These notions of non-classical concepts appear quite similar to the account of dcds developed here. Our acceptance conditions (rules of the form $\bar{x}^* \rightarrow 1$) correspond to Reiter's "default" sufficient conditions (a.k.a. frame selectors). However our rejection conditions (rules of the form $\bar{x}^* \rightarrow 0$) and Reiter's "default" necessary conditions (frame instantiations) are *contrapositives*, and do not serve precisely the same function [Gin87]. Still, the similarities are striking given the far different motivations behind these formalizations.

3 Model: Random Test Examples

We assume there is a "natural" source of random test examples against which we can evaluate the accuracy of any classifier. In particular, we assume there is a distribution P_{XC} over the space of domain objects and concept labels $X_n \times \{0,1\}$, called the *domain distribution*, from which random labelled objects are independently drawn. Before presentation, these labelled objects $\langle \bar{x}, c \rangle$ are first passed through a *blocking process* β to yield test examples $\langle \bar{x}^*, c \rangle$; see Figure 2. Thus, the domain distribution P_{XC} and the blocking process β induce a distribution P_{X^*C} over the space of possible examples, called the *example distribution*. The *accuracy* of a dcd d, written $P_{X^*C}(d)$, is defined as the probability that d correctly classifies a random test example. Note that in general a classifier's accuracy depends on both the domain distribution *and* the blocking process. We say that any example distribution P_{X^*C} for which d is optimal *satisfies* d.

Lemma 1 *For any example distribution P_{X^*C}, the optimally accurate dcd d includes the rule $\bar{x}^* \rightarrow c \in d$ whenever $P_{X^*C}\{\langle \bar{x}^*, c \rangle\} > P_{X^*C}\{\langle \bar{x}^*, \neg c \rangle\}$. Furthermore, for any dcd d, there is an example distribution P_{X^*C} which makes d non-trivially[5] optimal.*

We can therefore interpret any dcd d as asserting a collection of inequalities about the underlying example distribution. Notice the meaning of a rule $\bar{x}^* \rightarrow c$ depends not only on the (objective) distribution of domain objects in the world P_{XC}, but also on the (subjective) blocking process β, which specifies *how* information is received by the classifier. There are a number of reasonable assumptions one could make about β, but we restrict our attention to just two: *independent blocking* and *arbitrary blocking*.

3.1 Independent blocking

The *independent blocking* model, β_I, hides each object attribute x_i with a fixed probability p_i that is independent of x_i's value and those of the other attributes x_j, $j \neq i$. In this model, it turns out the optimally accurate dcd is determined strictly by the domain distribution P_{XC}, regardless of the specific blocking rates $\langle p_1, p_2, \ldots, p_n \rangle$.

Lemma 2 *Under β_I, for any domain distribution P_{XC}, the optimally accurate dcd d makes maximum conditional likelihood (mcl) classifications under P_{XC}, given the observed attributes of an object (cf., [DH73]).*

Thus, the structure of an optimal dcd d is determined solely by the domain distribution, and we can interpret d as a collection of assertions about the domain distribution P_{XC} directly: $\bar{x}^* \rightarrow c \in d$ asserts that $P_{X^*C}\{\langle \bar{x}^*, c \rangle\} \geq P_{X^*C}\{\langle \bar{x}^*, \neg c \rangle\}$. However, not all of the possible 2^{3^n} dcds consistently specify mcl classifications in this manner — only (and all) the ones consistent with the following "consistent inheritance axiom."

Definition 1 (Consistent Inheritance) *A dcd d is* inheritance consistent *iff*

$$\left.\begin{array}{l} \langle x_1^* \ldots 0 \ldots x_n^* \rangle \rightarrow c \in d \\ \langle x_1^* \ldots 1 \ldots x_n^* \rangle \rightarrow c \in d \end{array}\right\} \Rightarrow \langle x_1^* \ldots * \ldots x_n^* \rangle \rightarrow c \in d.$$

Theorem 1 *Under β_I, d is inheritance consistent $\Longleftrightarrow$ d is satisfiable by some domain distribution P_{XC}.*

Existing default logics based on ϵ-semantics (*e.g.*, [Pea89]) all satisfy the consistent inheritance axiom and so tacitly assume independent blocking β_I. Here the meaning of a rule $\bar{x}^* \rightarrow c$ can be given a "majority" semantics under β_I akin to that of [Bac90].

3.2 Arbitrary blocking

While β_I is a simple and convenient model, it does not capture every practical situation; in particular, it cannot deal with circumstances where our knowledge of an attribute is *correlated* with its value; *e.g.*, ex-inmates are unlikely to answer the question "have you ever been in prison?". The *arbitrary blocking* model, β_A, can hide object attributes x_i according to an arbitrary probability distribution that can be conditioned on the entire object $\bar{x}$ and its classification c, allowing this model to incorporate correlations between hidden attributes and their values, other attributes, or even concept membership.

Under β_A the structure of an optimal dcd does not depend solely on the domain distribution P_{XC}, but also on the nature of the blocking process β. This means that making mcl classifications according to P_{XC} may no longer be optimal. In fact,

Lemma 3 *Under β_A, making mcl classifications according to P_{XC} can yield error rates arbitrarily close to $1/2$, even when the optimum dcd has error rate 0.*

Of course, other classifiers, which can exploit correlations between missing attributes and object classifications, can do much better in these situations.

[5] Here we are ruling out the "pure noise" case where $P_{X^*C}\{\langle \bar{x}^*, c \rangle\} = P_{X^*C}\{\langle \bar{x}^*, \neg c \rangle\} = 1/2$ for each $\bar{x}^* \in X_n^*$; here every dcd is (trivially) optimal.

4 Learning Accurate Default Concepts

We now consider the task of *learning* an accurate dcd from random training examples. We assume the learner L receives a sequence of random training examples drawn from a *training distribution*, from which it must produce a dcd, which is then tested on random test examples drawn from a *test distribution*. The learner's goal is to produce an accurate dcd with as few training examples as possible. We can consider a number of distinct learning problems, based on our assumptions about the form of training examples and the type of blocking process. Here, we focus the two types of blocking introduced in Section 3, and on the following two types of training examples.

The *incomplete* training example model, χ_I, assumes training examples are generated by the *same* example distribution that generates test examples. This is a natural model for many practical settings where we do not have access to complete object descriptions, even for training examples. One benefit of training on partial examples is that learner is exposed to the natural blocking process operating in the domain.

The *complete* training example model χ_C, on the other hand, assumes training examples are generated by the same *domain* distribution P_{XC} underlying the process that generates incomplete test examples. Here, however, some teacher has "filled in" the proper value of each attribute of each training example. Even though our goal is to learn classification rules that classify incomplete examples, we can still consider learning from *complete* examples. This situation that can easily arise in practical situations; *e.g.*, a medical student may be trained to diagnose the presence of a particular disease given fairly complete descriptions of all relevant patient data, and yet as a doctor, be expected to produce diagnoses without the benefit (and cost) of performing every available diagnostic test. Furthermore, we intuitively expect an advantage in training on complete examples as they appear to provide more information than incomplete examples. We will see below that this intuition is only sometimes correct.

4.1 Learning under Independent-Blocking

We first consider learning under the independent blocking model β_I. This is the simplest and arguably most natural blocking model, where the fact that an attribute is missing provides no information about the underlying values or object classifications. Lemma 2 showed that under β_I, the structure of the optimal d_{opt} depends solely on the domain distribution P_{XC}, regardless of the blocking probabilities $\langle p_1, \ldots, p_n \rangle$. In particular, d_{opt}'s classifications depend on the most probable class (under P_{XC}) given the *observed* (non-$*$) attributes of a description $\bar{x}^*$; *i.e.*, if $\mathrm{P}_{C|X}(c \mid obs(\bar{x}^*)) > \mathrm{P}_{C|X}(\neg c \mid obs(\bar{x}^*))$ then $\bar{x}^* \rightarrow c \in d_{opt}$. Hence, under β_I, learning an accurate dcd requires only determining whether $\mathrm{P}_{C|X}(c = 1 \mid obs(\bar{x}^*)) > \mathrm{P}_{C|X}(c = 0 \mid obs(\bar{x}^*))$ for each object description $\bar{x}^*$, based on observing a sequence of training examples.

4.1.1 Estimating Most Likely Classifications

Complete training examples: Here the learner is given a sequence of random training examples $\langle \langle \bar{x}_1, c_1 \rangle, \ldots, \langle \bar{x}_m, c_m \rangle \rangle$ (drawn independently from the domain distribution P_{XC} — the same domain distribution that will be used to generate pre-blocked test examples), from which it must decide whether to use the classification rule $\bar{x}^* \rightarrow 1$ or $\bar{x}^* \rightarrow 0$ for each description $\bar{x}^*$. Here, it seems reasonably obvious that this decision should be based on the *observed* classification frequencies among all training examples $\bar{x}$ that *match* a description $\bar{x}^*$, as specified by the following learning strategy.

MLC (Maximum Likelihood (Complete)) For description $\bar{x}^*$, predict the most frequent class among all training examples whose domain object *matches* $\bar{x}^*$.

This simple strategy turns out to have the following rather remarkable optimality property.

Theorem 2 *For any learner* L[6] *that produces the* optimal *rule* $\bar{x}^* \rightarrow c$ *for some* $\bar{x}^*$ *with higher probability than* MLC, *given some* P_{XC} *and sample size* m, *there is another domain distribution* P'_{XC} *for which* L *produces a dcd* d *with accuracy* $< 1/2$ *with probability* $> 1/2$.

Thus, no learner can outperform MLC on *any* non–pure-noise domain distribution (*i.e.*, where $\mathrm{P}_{X*C}\{\langle \bar{x}^*, c \rangle\} \neq 1/2$ for some $\bar{x}^*$), and object description.

Incomplete training examples: Here the learner is given a sequence of random training examples $\langle \langle \bar{x}^*_1, c_1 \rangle, \ldots, \langle \bar{x}^*_m, c_m \rangle \rangle$ (drawn independently from the same example distribution P_{X*C} used to generate test examples), from which it must decide whether to use classification rule $\bar{x}^* \rightarrow 1$ or $\bar{x}^* \rightarrow 0$ for description $\bar{x}^*$. As before, the optimal classification rules are determined by the underlying domain distribution P_{XC}, and so the general idea is to gain as much information as possible about P_{XC} from the random training examples; the difficulty here is that many of the training object attributes will be blocked. The challenge, therefore, is to extract as much information as possible from the object attributes that are actually observed.

A number of techniques have been proposed in the machine learning literature for determining the most likely classification of a description from a collection of incomplete training examples. Surprisingly, none of these techniques appear to make the most efficient use of the available training data. This leads us to investigate a simple statistical principle, relatively unused in machine learning, that appears to be far more efficient for this purpose. We first briefly survey the existing proposals and point out the intuitive source of inefficiency in each.

The first technique ignores the fact that training descriptions are independently blocked versions of complete descriptions, and simply gathers separate statistics for each description $\bar{x}^*$; effectively treating "$*$" as a third attribute value.

[6] Given the benign assumption that L's guesses for a description $\bar{x}^*$ are conditionally independent of the training labels of domain objects $\bar{x}$ that do *not* match $\bar{x}^*$.

THV (Three-valued) [Qui89] For description $\bar{x}^*$, predict the most frequent classification among training examples of the form $\langle \bar{x}^*, c \rangle$.

THV clearly does not make the most effective use of the available training data, given that attributes are blocked independently of their values. In particular, it ignores more specific training patterns that might match the description x^*, which is ineffective as these patterns can provide additional information about the prevalence of a particular classification among objects matching $\bar{x}^*$. The next refinement is a technique that takes just this information into account.

LEM (Local error minimization) For description $\bar{x}^*$, predict the most frequent class among *all* more specific training patterns that *match* $\bar{x}^*$.

By considering more *specific* training patterns, LEM makes more efficient use of the training data than THV. However, it turns out that even LEM does not fully exploit all of the relevant information that can be gleaned from the training examples. In fact, there are situations where we ought to incorporate statistics from *more general* descriptions than $\bar{x}^*$. To illustrate this, imagine a simple setting where domain objects are described by a single bit, so any dcd for this domain will consist of three rules: $\{ \langle 0 \rangle \rightarrow c_0,\ \langle 1 \rangle \rightarrow c_1,\ \langle * \rangle \rightarrow c_* \}$ where each each $c_i \in \{0, 1\}$. Now, imagine a collection of training examples where

$$\#\langle\langle 0\rangle, 0\rangle = 2 \quad \#\langle\langle 1\rangle, 0\rangle = 2 \quad \#\langle\langle *\rangle, 0\rangle = 0$$
$$\#\langle\langle 0\rangle, 1\rangle = 1 \quad \#\langle\langle 1\rangle, 1\rangle = 1 \quad \#\langle\langle *\rangle, 1\rangle = 14.$$

Here, since $\#\langle\langle 0\rangle, 0\rangle > \#\langle\langle 0\rangle, 1\rangle$, it appears $\langle 0\rangle \rightarrow 0$ would be the optimal rule for $\langle 0\rangle$; similarly $\#\langle\langle 1\rangle, 0\rangle > \#\langle\langle 1\rangle, 1\rangle$ suggests $\langle 1\rangle \rightarrow 0$. Notice, however, that all 14 of the $\langle * \rangle$ observations belong to class 1, and *each of these must have actually been a domain object with attribute value* $\langle 0\rangle$ *or* $\langle 1\rangle$. So there is overwhelming evidence that at least one of the two attribute values (if not both) should be classified 1 rather than 0. This is a clear case where the statistics from a more general description should override those of the more specific.

A learning technique that attempts to do just this has been proposed in the philosophy of statistics literature — namely Kyburg's proposals for choosing the best reference class on which to base statistical judgements.

REF (Reference class) [Kyb83, Kyb91] For description $\bar{x}^*$, first select a "reference-class" description $\bar{x}^*_r$ (either $\bar{x}^*$ itself, or possibly a more general description), then predict the most likely classification given all training descriptions that *match* the reference class description $\bar{x}^*_r$.

The idea is to select a sufficiently general description $\bar{x}^*_r$ so that our choice of classification rule $\bar{x}^* \rightarrow c$ for $\bar{x}^*$ is based on "adequate" statistics. Kyburg suggests the following reference class selection procedure: For each incomplete description $\bar{x}^*$, compute a 90% (say) confidence interval about the probability of observing classification c given all training descriptions that match $\bar{x}^*$. Then employ a conflict resolution strategy (which trades-off interval bias and width) to decide whether to adopt, for this $\bar{x}^*$, the classification associated with successively more general reference classes [Kyb91].[7]

Although the REF strategy can override the predictions from specific descriptions with those from more general descriptions, it is not clear that it does so in the best conceivable way. The strategy is fundamentally *ad hoc* (in particular by incorporating an arbitrary parameter in the confidence intervals), and is not based on any real principles beyond "intuition" to adjudicate between candidate reference class descriptions. Furthermore, there is no empirical data to support the efficacy of this approach.

It is often stated that the crux of this type of statistical reasoning is the problem of "choosing the right reference class" [Bac90, BGHK92]. However, this premise might actually be leading us away from the most effective learning approaches here. Fundamentally, our goal should be to preserve *all* available statistical information, rather than throwing away statistics from one class in favor of those from another. The best approach should involve *combining* all of the available statistics in a principled way. Here we note that a well-known idea from theoretical statistics is applicable: namely, first determine the *maximum likelihood* distribution that accounts for *all* the data, then perform inferences according to this distribution [LR87]. This approach yields an effective method for determining the most likely classifications given incomplete training examples.

MLI (Maximum Likelihood (Incomplete)) [LR87] First, determine the domain distribution P^{max}_{XC} that maximizes the likelihood of the observed training examples. Then, for description $\bar{x}^*$, predict the most probable classification according to P^{max}_{XC}, given $\bar{x}^*$'s *observed* attributes.

Notice that this approach never "throws away" an observation; instead, it seeks the best model that accounts for all of them. The statistics for *all* relevant descriptions, both more general and more specific than $\bar{x}^*$, are combined in a principled way to yield a classification.

Based on the preceding discussion it seems intuitive that MLI should be more efficient than the other learning strategies, *i.e.*, we expect that MLI should produce more accurate classification rules, given fewer training examples. Although an optimality result akin to Theorem 2 has not yet been proven, it is fairly easy to demonstrate the superior efficiency of MLI empirically.

To support this point, consider the results of the following simulation study: Each of the four techniques was implemented and tested in the simple domain where domain objects are described by a single bit (as before). We then tested the techniques on random domain distributions and blocking rates, and recorded the accuracies

[7] Philosophical discussions often mention the difficulty in choosing the candidate reference classes to participate in any conflict resolution procedure (*cf.*, [Bac90, Chapter 5]). Kyburg simply adopts the reference classes considered here, and ignores other "disjunctive" classes (*cf.*, Section 2) by fiat. However, there is a principled argument behind ignoring disjunctive classes, based on the observation that they do not correspond to any possible "partial states of knowledge" one can have about a domain object, *cf.*, Section 2.

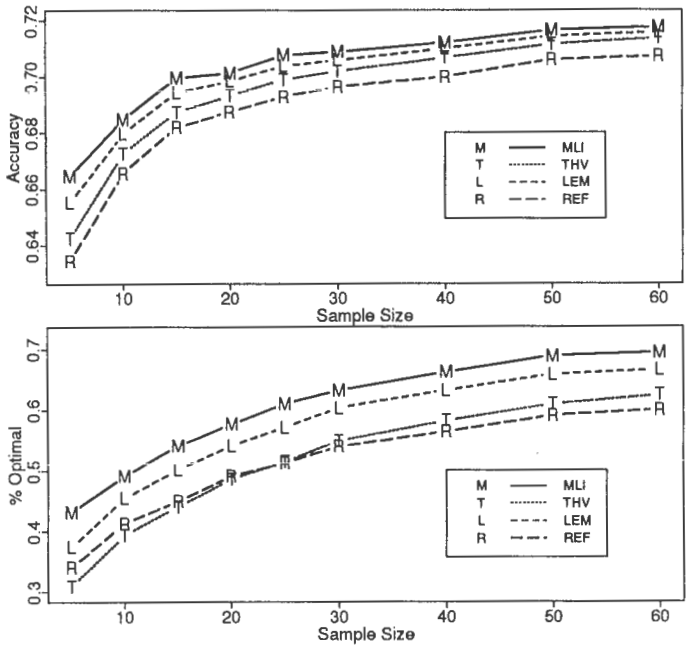


Figure 3: Percent Optimal, Accuracy vs Training Size

of the classification rules produced by each strategy. The graphs in Figure 3 plot the average accuracy obtained by each learner (resp., how often each learner returned the optimal dcd), as a function of training sample size; averaged over 10,000 trials. It is clear that, for a given number of training examples, MLI both attains the highest average accuracy levels, and also identifies the optimal dcd with the highest probability, *cf.*, Theorem 2.

4.1.2 Scaling Up

As efficient as the previous estimation techniques appear to be (particularly MLC and MLI), they cannot be applied "as is" to any real learning task. The problem, of course, is that these estimation techniques simply do not *scale up*. This is because determining the appropriate classifications for arbitrary object descriptions $\bar{x}^*$ can, in general, involve the simultaneous estimation of an exponential number of parameters (in n). For example, there are $\binom{n}{\lceil n/2 \rceil}$ descriptions containing $\lceil n/2 \rceil$ $*$'s, and most of the observations for one such pattern does not match any of the others. None of the estimation techniques generalize between these patterns.

This is a well-known issue in machine learning: to achieve reasonable performance with reasonable amounts of data, we will eventually have to introduce some form of prior knowledge to constrain our learning systems. This points to the necessity of *bias*. In any successful application, the learning system must be constrained to search a restricted space of appropriate classifiers, which here are dcds.[8]

Following the methodology pioneered by Valiant [Val84], we consider how learning performance *scales* as a function of prior knowledge. Here we *quantify* bias by its measurable effects on the quality of learning that can be guaranteed. *A la* Valiant, we consider prior domain knowledge that can be expressed by a restricted *set* of dcds $\mathcal{D}$, which is known to include the *optimal* dcd. The difficulty of learning a set of dcds $\mathcal{D}$ is then measured by the number of training examples needed to reliably guarantee a near optimal hypothesis, in the worst case over all possible example distributions satisfying some dcd $d \in \mathcal{D}$.

Definition 2 (PACO-learning) [9] *A learner L* PACO-learns *a class of dcds $\mathcal{D}$ under β_I blocking given $m(\epsilon, \delta)$ χ-type training examples ($\chi \in \{\chi_C, \chi_I\}$), if $\forall \epsilon > 0, \forall \delta > 0$, and $\forall$ domain distributions $\mathrm{P}_{\chi C}$ consistent with some $d_{opt} \in \mathcal{D}$, L outputs a dcd $d_L \in \mathcal{D}$ whose accuracy is within ϵ of this d_{opt}, with probability at least $1 - \delta$.*

To investigate scaling, we consider parameterized classes $\mathcal{D}_n$ defined on n attributes for $n = 1, 2, \ldots$.

Definition 3 (Feasible-learnability) *A parameterized class of dcds $\mathcal{D}_n$, $n = 1, 2, \ldots$ is said to be* feasibly-learnable *if there exists a polynomial function $poly(\cdots)$ and a learner L that* PACO*-learns each $\mathcal{D}_1, \mathcal{D}_2, \ldots$ with sample size $m(\epsilon, \delta) = poly(\frac{1}{\epsilon}, \frac{1}{\delta}, n)$.*

Intuitively, we expect the difficulty of learning a set of dcds $\mathcal{D}$ to depend on the "complexity" of $\mathcal{D}$, *i.e.*, more complex $\mathcal{D}$s are harder to learn. The question is: what precise complexity measure (effectively measuring the "amount" of prior knowledge encoded by $\mathcal{D}$) actually determines the difficulty of PACO-learning a default concept class $\mathcal{D}$? It turns out the appropriate complexity measures can be based on the notion of the *Vapnik-Chervonenkis dimension* of a set of dcds $\mathcal{D}$, written $\mathrm{VCdim}(\mathcal{D})$.[10]

Learning performance also clearly depends on the precise learning model under consideration (*e.g.*, β_I blocking and either χ_C or χ_I training examples). For β_I blocking and complete training examples, we have been able to identify precise conditions on the complexity of $\mathcal{D}_n$ (as a function of n) that determine whether $\mathcal{D}_n$ is feasibly learnable.

Lemma 4 *Under β_I blocking, $\mathcal{D}_n$ is feasibly-learnable from complete training examples $\Longleftrightarrow \forall s \subset \{1, \ldots, n\}$, $\mathrm{VCdim}(\mathcal{D}_n^s) = poly(n)$ (where $\mathcal{D}_n^s$ is the set of ccds induced by $\mathcal{D}_n$ on attribute subset s).*

In the case of learning from incomplete training examples, a much stronger condition can be shown to be sufficient for feasible-learning.

Lemma 5 *Under β_I blocking, $\mathrm{VCdim}(\mathcal{D}_n) = poly(n)$ $\Longrightarrow$ $\mathcal{D}_n$ is feasibly-learnable from incomplete examples.*

[8] THV and MLI are particularly well suited to incorporating background knowledge; as demonstrated for THV in many decision-tree applications [Qui89, BFOS84], and for MLI by applications of the EM algorithm to parameterized domain distributions [LR87].

[9] For "Probably Approximately Class Optimal". Our goal differs slightly from standard PAC-learning, as we are forced to seek near-*optimal* rather than near-*perfect* classifiers, since with blocking *no* classifier can attain perfect accuracy in general. Notice also that we are only addressing the *sample* complexity of learning, *not* computational complexity.

[10] This is the same measure used when learning ccds. See [BEHW89] for a precise definition of VCdim and its application to determining the difficulty of learning sets of ccds.

Combining these lemmas yields the intuitive result that learning from complete training examples is easier than learning from incomplete examples:

Corollary 1 *$\mathcal{D}_n$ is feasibly-learnable under (β_I,χ_I) $\Longrightarrow$ $\mathcal{D}_n$ is feasibly-learnable under (β_I,χ_C).*

However, the converse (*i.e.*, is there a class $\mathcal{D}_n$ that is is feasibly-learnable from complete but not incomplete training examples) remains an open question.

4.2 Learning under Arbitrary-Blocking

We now consider the arbitrary blocking model β_A. In this model, the fact that an attribute is missing from an object description can be correlated in various ways with the attribute values and the object's classification. In effect, *no* reliable information can be obtained about the value of missing attributes under β_A. Here, learning an accurate dcd amounts to determining whether $P_{C|X^*}(c = 1 \mid \bar{x}^*) > P_{C|X^*}(c = 0 \mid \bar{x}^*)$ for each object description $\bar{x}^*$, given training examples.

4.2.1 Estimating Most Likely Classifications

As in Section 4.1.1, we can consider the problem of estimating the most likely classification of a description $\bar{x}^*$ from both complete and incomplete training examples. The relative merits of the various learning techniques discussed in Section 4.1.1 change dramatically under these alternative learning conditions.

Complete training examples: Notice that *complete* training examples provide no information about the blocking process that will be applied to future test examples. By observing complete training examples, the learner can only estimate properties of the *domain* distribution P_{XC}, and not the *test example* distribution P_{X^*C} (generated by a blocking process over P_{XC}). Therefore it is fundamentally impossible to estimate whether $P_{C|X^*}(c = 1 \mid \bar{x}^*) > P_{C|X^*}(c = 0 \mid \bar{x}^*)$ for arbitrary blocking processes just by observing complete training examples. Lemma 3 exploits this fact to show that even given *exact* knowledge of the domain distribution P_{XC}, any classification rule produced by a learner can still have an arbitrarily high error rate on incomplete test examples for some example distribution P_{X^*C}. Therefore *no* learning strategy can reliably estimate the proper classification of an incomplete test description $\bar{x}^*$ from complete training examples.

Incomplete training examples: Given *incomplete* training examples, however, the learner is directly exposed to the natural blocking processes operating in the domain. Under these conditions it is possible to estimate whether $P_{C|X^*}(c = 1 \mid \bar{x}^*) > P_{C|X^*}(c = 0 \mid \bar{x}^*)$ for a description $\bar{x}^*$, simply by applying the THV strategy of determining whether $\#\langle \bar{x}^*, 1\rangle > \#\langle \bar{x}^*, 0\rangle$.

The various learning techniques discussed in Subsection 4.1.1 have different relative merits under the different learning conditions: We saw in Subsection 4.1.1 that LEM and MLI were more efficient than THV under β_I blocking. In general, maximum likelihood estimation (MLC, MLI) appears to be the superior technique for estimating the most probable classifications under β_I, regardless of whether complete or incomplete training examples are available. However, since these techniques base their judgements directly on estimated properties of the *domain* distribution P_{XC}, Lemma 3 shows that their classifications can have arbitrarily high error rates under β_A. In contrast, THV is the *only* provably effective technique for learning under β_A, given incomplete training examples, and so clearly dominates in this case.

These theoretical observations can actually help explain some of the results obtained by recent empirical studies: Quinlan [Qui89] compared applications of the LEM and THV techniques (along with some other ad hoc approaches) to decision-tree learning, and found that no single technique dominated the others over the set of test problem he considered. The preceding theoretical results, however, clearly demonstrate that the relative effectiveness of particular learning strategies strongly depends on the nature of the *blocking* process involved; an observation that can be applied in practice. For example, if blocking is known to be (more or less) independent (β_I), then MLI should outperform the other techniques, however, if blocking were known to be strongly correlated (β_A), then THV should dominate.

4.2.2 Scaling Up

As in Subsection 4.1.2, we can determine what constraints on prior knowledge (expressed as a parameterized class of dcds $\mathcal{D}_n$) are sufficient to permit efficient learning, as we scale up in n.

Lemma 6 *Under β_A, $\mathcal{D}_n$ is feasibly-learnable from incomplete examples* $\Longleftrightarrow$ $\mathrm{VCdim}(\mathcal{D}_n) = poly(n)$.

Notice that although complete training examples actually make learning *easier* under β_I, they make learning *impossible* under β_A. This is because complete examples provide information only about instance distribution, but supply *no* information about the blocking process that will be applied to future test examples. While this is not a problem under β_I (where the optimal classifications are determined strictly by the instance distribution P_{XC}), this issue is fatal under β_A; *cf.*, Lemma 3.

Lemma 7 No *non-trivial set $\mathcal{D}$ of default concepts is* PACO*-learnable under (β_A,χ_C).*

As expected, the feasible learnability of a parameterized class of dcds $\mathcal{D}_n$ depends on the specific conditions in which learning takes place. Here we compare the relative difficulty of learning under the various conditions.

Lemma 8 *$\mathcal{D}_n$ is feasible-learnability under (β_A,χ_I) $\Longrightarrow$ $\mathcal{D}_n$ is feasible-learnability under (β_I,χ_I) $\Longrightarrow$ $\mathcal{D}_n$ is feasible-learnability under (β_I,χ_C).*

The first inclusion is strict, as

Lemma 9 *There are parameterized classes $\mathcal{D}_n$ which are feasibly-learnable under (β_I,χ_I), but not feasibly-learnable under (β_A,χ_I).*

Hence, learning under (β_I,χ_I) is fundamentally easier than learning under (β_A,χ_I), as it can require exponentially fewer training examples in some cases.

5 Conclusions

This work constitutes a start on the general problem of acquiring default knowledge from empirical observations. Of course, much remains to be done. One of the more immediate concerns is to develop an efficient implementation of the MLI strategy for useful forms of bias. We are also beginning to examine many extensions to better cope with practical problems. For example, many application domains like medical diagnosis have the property that missing attribute values actually give *useful* information — namely that the missing attributes are *irrelevant* to the classification, given the known attribute values [PBH90]. Notice that β_I is overly restrictive and β_A is too underconstrained to adequately model such tasks; [GHR94] provides an initial analysis of this situation. We are currently investigating other intermediate blocking models that can more accurately model such domains and (we hope) lead to better empirical learning performance.

Other interesting research directions involve alternative generalizations of standard classification learning: This work has assumed that default definitions categorically classify every description, no matter how incomplete. An interesting direction is to consider *partial* default definitions that sometimes say "I don't know" *à la* [RS88]. Such classifiers could prove useful in domains where the consequences of an incorrect classification sometimes outweigh those of remaining silent.

Another interesting extension is to consider *active* classifiers. That is, we have assumed that classifiers *passively* observe test examples and play no role in determining which attributes are observed. It would be interesting to consider learning classifiers that *actively* decide which attributes to test, and when there is sufficient information to posit an accurate prediction (*i.e.*, learning to *diagnose*). This raises the issue of how best to trade off the number of tests required against the accuracy of the classifier.

Contributions: We formulated and studied the problem of learning "default concepts" (dcds), which can then be used to classify incomplete object descriptions. After formally defining the structure and function of dcds, we modelled the classification (performance) task as a random example generator that passes examples through a "blocking process" that hides object attributes from the classifier. We then addressed the task of learning dcds from random examples — first discussing many of the standard techniques for this problem, and then explaining why MLI is more effective than many standard learning techniques under the (β_I,χ_I) model. We also extended Valiant's PAC-learning framework to the problem of learning dcds: assessing the effects of prior knowledge on learning efficiency, and determining the difficulty of learning under different conditions. By providing a theoretical understanding of many empirical observations in the literature, we hope that our results will lead to the development of more effective learning procedures for practical problems that involve missing data.

References

[Bac90] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* MIT Press, 1990.

[BE89] A. Borgida and D. Etherington. Hierarchical knowledge bases and efficient disjunctive reasoning. In *KR-89*, 1989.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *J. of ACM*, 36(4), 1989.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[BGHK92] F. Bacchus, A. Grove, J. Halpern, and D. Koller. From statistics to beliefs. In *AAAI-92*, 1992.

[Cla85] W. Clanccy. Heuristic classification. *Artificial Intelligence*, 27, 1985.

[DH73] R. O. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[GHR94] R. Greiner, T. Hancock, and R. B. Rao. Knowing what doesn't matter. Technical report, Siemens Corporate Research, 1994.

[Gin87] M. Ginsberg, editor. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, Los Altos, 1987.

[KS90] M. J. Kearns and R. E. Shapire. Efficient distribution-free learning of probabilistic concepts. In *FOCS-90*, 1990.

[Kyb83] H. Kyburg. The reference class. *Philosophy of Science*, 50, 1983.

[Kyb91] H. Kyburg. Evidential probability. In *IJCAI-91*, 1991.

[LR87] J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data.* Wiley, 1987.

[Min75] M. Minsky. A framework for representing knowledge. In *The Psychology of Computer Vision.* McGraw-Hill, 1975.

[PBH90] B. Porter, R. Bareiss, and R. Holte. Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence*, 45, 1990.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[Pea89] J. Pearl. Probabilistic semantics for nonmonotonic reasoning: A survey. In *KR-89*, 1989.

[Qui89] J. R. Quinlan. Unknown attribute values in induction. In *ML-89*, 1989.

[Rei87] R. Reiter. Nonmonotonic reasoning. *Annual Review of Computer Science*, 1987.

[Riv87] R. Rivest. Learning decision lists. *Machine Learning*, 2(3), 1987.

[RS88] R. Rivest and R. Sloan. Learning complicated concepts reliably and usefully. In *AAAI-88*, 1988.

[Sch94] D. Schuurmans. *Efficient, Accurate, and Reliable Machine Learning.* PhD thesis, Univ. of Toronto, Dept. Computer Science (forthcoming)

[SV88] G. Shackelford and D. Volper. Learning k-DNF with noise in the attributes. In *COLT-88*, 1988.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), 1984.

Unsupervised Learning of Planning Knowledge

Bertrand Pelletier
Druide Informatique
4333 Ste-Catherine Ouest, Bur. 270
Westmount, Québec
H3Z 1P9
Canada
bertrand@binkley.cs.mcgill.ca

Stan Matwin
Ottawa Machine Learning Group
Department of Computer Science
University of Ottawa
Ottawa, Ontario
K1N 6N5
Canada
stan@csi.uottawa.ca

Abstract

This paper describes an approach to the problem of learning from observing experts' behaviour. The architecture of LEADER (LEArning DEsign Rules), an implemented system to realize this form of unsupervised and non-obtrusive learning, is discussed. The approach exploits techniques and representations from machine learning and planning to address the problem. Starting from an incomplete theory of goals and plans used by the experts to solve problems, and using several instances of experts behaviours not fully explained by this theory, missing knowledge is learned to augment the theory, making it adequate to explain a larger part of the behaviours. The learning methods were applied to two domains – entity-relationship model design and office tasks – where favorable empirical results have been obtained and are described.

1 Introduction

Learning by observing an expert at work is a natural way to acquire expertise. This type of learning is most convenient in situations where the expert does not want to be disturbed and the apprentice cannot validate immediately his or her (or even its) assumptions and findings through actual experience. These situations are the ones that motivate the realization of the present learning system.

When addressing problems related to the automation of knowledge acquisition, one may list four classes of problems [Levi, Perschbacher et al., 1988]:

- construction of an initial knowledge base
- improvement of the existing knowledge
- automatic adaptation of the system to the style and to the level of expertise of the human user
- elaboration of principles and techniques for building intelligently behaving systems.

Each of these classes contains numerous difficult problems. The second class of problems, the improvement of the information used by a knowledge-based system, is known as the *knowledge base refinement problem*, a special case of the *theory revision problem* [Ginsberg, 1989] dealing with pathologies of the domain theory such as incorrectness, inconsistency, incompleteness and inefficiency.

LEADER (LEArning DEsign Rules), the learning system we propose here is to acquire new plans used by some agent (typically a human expert) solving problems in a given domain. Its learning capabilities rely on the following assumptions:

- the agent's behaviour is driven by goals
- the agent creates and executes hierarchical plans to achieve desired goals
- the system learns without interacting with the agent.

The initial knowledge base is assumed to be correct and consistent; the learning system focuses primarily on the problems of incompleteness and inefficiency: acquiring new plans to perform tasks. However, because acquired plans can be overgeneralized, correctness and consistency are also of concern.

Some existing learning systems address similar problems to the ones involved in LEADER. Explanation-based learning (EBL) systems [DeJong and Mooney, 1986] share large similarities with LEADER as LEADER uses EBL as a component. However, in contrast with these systems where learning occurs when training instances are fully explained (by a domain theory assumed to be complete), LEADER learns only when the domain theory fails to explain the instances: it learns the missing elements.

Because the learning system acquires its knowledge by observing only the agent's interactions (i.e. the sequence of actions performed by the agent), there are no disturbances of the agent during the interaction with the given domain. This contrast with Learning Apprentice Systems (LAS). [Mitchell and Mahadevan, 1990], such as ARMS [Segre, 1988], a system for robot control that requires that the teacher guides the robot arm through a series of movements that achieve a given goal, CLERK [Campbell, 1990], a system that learns tasks accomplished by an electronic-mail system user by recording the task and the corresponding sequence of actions taught by the user, or LEAP [Mitchell

and Mabadevan, 1990], a system designed to acquire new knowledge in VLSI by observing design steps.

The organization of the paper is as follows: Section 2 presents the architecture of the system and introduces concepts needed for its description and analysis. Section 3 presents the system's kernel: its learning component. An evaluation of the system is described in Section 4, and Section 5 presents our conclusion.

2 Architecture and basic concepts

The learning system is showed Figure 1; it contains four components, the *Observer*, the *Executor*, the *Analyzer* and the *Learner*. The last three components form LEADER (the broken line box).

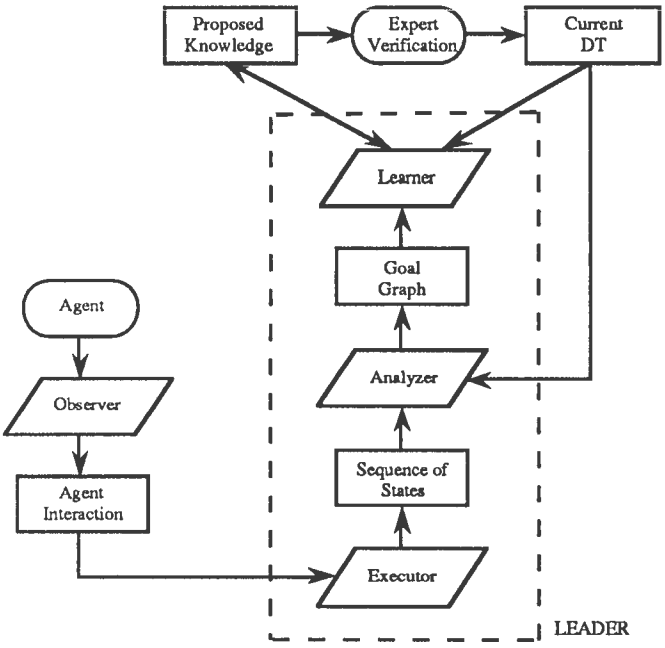


Figure 1. Architecture of the learning system. Ovals represent external sources of information, parallelograms represent processes, rectangles represent data used or produced by processes, and arrows represent data flow.

Here is how the process goes. First, the *Observer* watches the agent interaction, with no disturbance, and produces the ordered sequence of actions corresponding to that interaction. LEADER does not perform planning; it observes the user. Consequently, some of the problems of planning (e.f. conflicting goals) never arise. Then, the *Action Executor* simulates in order each of the actions of the provided interaction and produces the sequence of corresponding states. Next, the *Analyzer* uses the actual domain theory (the one that is to be refined) to build the *goal graph*, a structure showing how each of the goals achieved by the agent is decomposed into subgoals and is ultimately realized by individual actions. Finally, the *Learner*, the component where all the learning is performed, uses the actual domain theory (and possibly also the enhanced theory) to analyze the goal graph for generating proposals to explain parts of the interaction that were left that were left unexplained by the actual theory. After the system has been functioning for a while, a human expert in the domain examines the accumulated proposed knowledge to verify it and add the filtered knowledge to the actual domain theory.

Below, we introduce the basic structures and notions on which we have founded the design of LEADER.

The basic representational components of our approach are: the goals (the specification of the properties the agent wants to achieve), the interaction (the sequence of actions executed), the hierarchical plan (the hierarchy of agent's goals and subgoals, along with the corresponding plans and sub-plans), the states resulting from the execution of the actions, and so forth.

To represent the states, the basic operators and the hierarchical plans created, the approach uses a variation of the TWEAK formalism [Chapman, 1987]. In TWEAK, each operator is represented by a precondition (defining, as for STRIPS [Fikes and Nilsson, 1971], its condition of applicability) and a postcondition (defining the strongest condition that is true in the state resulting from the operator application). The TWEAK formalism leads to efficient planning ([Chapman, 1987; Kambhampati and Hendler, 1990]), but its main advantage is the STRIPS assumption voiding the specification of frame axioms, i.e., the requirement that the DEL and ADD lists of an operator specify everything about the initiating state that is altered by the execution of the operator. For complex domains, such an enumeration can be tedious, or even impossible.

One way to overcome the problem consists of separating the conditions into two classes, the primitive (or primary) and the inferential (or secondary) literals. As [Waldinger, 1977] pointed out, using inferential literals makes easier the description of operators, allows more efficient updates of the states, and makes possible the introduction of new relationships between literals without modifying the operators' descriptions. We thus augmented the TWEAK representation with DEL and ADD lists (while keeping the postcondition component), and we allow the presence of inferential literals: it allows the use of an axiomatic theory where axioms are implication formulas in the disjunctive normal form.

As Figure 2 shows, to each action a_i of the interaction corresponds an (ordered sequence of) states S_i. A goal G is expressed by a pair of conditions $<P_0,Q_0>$ specifying what holds before and after its achievement, respectively.

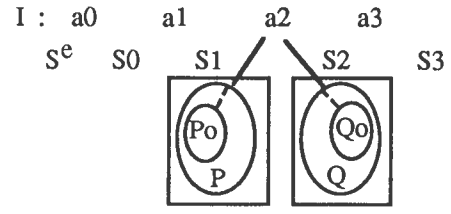


Figure 2. An interaction with its corresponding states. S^e is the empty state and a_0 represents the action that creates S_0, the state in which the agent begins the interaction. The figure also illustrates how the goal $<P_0,Q_0>$ is achieved by the execution of the action a_2 whose precondition and

of the action a_2 whose precondition and postcondition are P and Q, respectively.

A hierarchical plan <P,H,Q> achieves a goal G = $<P_0,Q_0>$ if P implies P_0 and Q implies Q_0., where H is a hierarchical plan (i.e. a sequence of subgoals) that achieves the goal G. In such a case, the hierarchical plan is called a *goal rule* and is represented by G <= <P,H,Q>.

These definitions are "conservative" in the sense of preferring having an eventually too strong precondition or too weak postcondition rather than having an incorrect description. One can look at data-flow analysis as an analogy, where useful properties of a sequence of statements are computed, and where to be "conservative" means to prefer missing optimization opportunities to guarantee no changes in what the program computes.

3 Learner: the learning component of LEADER

An interaction is either totally or partially explained by the current set of goal rules. We focus on the second case, when learning can occur to explain the unexplained parts. For instance, new goals and new hierarchical plans can be inferred. The following outlines our approach to this task:

- identify a goal for which the actions contribute to the realization
- identify the hierarchical plan realizing this goal
- identify the precondition and postcondition of this hierarchical plan.

3.1 Identifying a candidate goal

A given interaction may achieve a large number of goals. Mechanisms must be used to cut down the number of goals and goal rules proposed by the Learner; these filtering mechanisms must ensure that (most) good candidates are not thrown away.

The easiest way to identify good candidate goals is by having the goal provided by the agent as a component of the training instance. This is the approach taken in typical learning systems such as [DeJong and Mooney, 1986; Segre, 1988; Mitchell and Mabadevan, 1990; Hammond, 1989].

The main limitation of this approach is precisely that the goal must be provided: without this focus of interest, learning (if possible) is restricted. An alternate way to identify goals is by having a predetermined list of "meaningful" goals to look for, a list usually provided by experts when the domain theory is created. The major problem with this approach is that the list is static: the searching method never looks for goals that are not in the list, and thus never learns new goals.

A more useful and challenging approach consists in allowing the learning system to create new goals when those present in the domain theory are inadequate to explain the agent's behaviour. For instance, a promising method for identifying new candidate goals might rely on looking at consequences and prerequisites of unexplained actions or partially realized tasks. Indeed, failures to explain an interaction occur either because the hierarchical plan used by the agent is unknown, or because the goal is unknown (or both). So, focussing on unexplained parts is a good heuristic approach to identify new goals.

The approach taken in LEADER is neither the creation of new goals nor a search through a list of provided goals: it is an hybrid of the two. As described below, although a list of goals in provided by the domain theory, LEADER does not directly process it: LEADER first looks for goal rules that are partially realized[1] to get clues about good candidate goals. Then, it uses these candidate goals to propose new goal rules.

The general method described next for identifying goals also provides information on what part of the interaction achieves them. There are several differences between the approach presented here, and the traditional techniques such as EBL and backward chaining inference. Firstly, in our representation we have in fact two domain theories: a theory of goals decomposed into subgoals and/or actions, and the theory of inferential literals. Secondly, the theory of goals is assumed to be incomplete here, and the incompleteness is handled with heuristics; there is no incompleteness allowed in EBL. Thirdly, EBL determines the weakest precondition for satisfying a given goal, while our method determines both the precondition of a goal and its postcondition after the execution of the interaction. The method (illustrated by Figure 3) is the following:

Given:
- a domain theory
- an interaction I
- the goal graph corresponding to I

Build:
- a good candidate goal G' = <P',Q'>
- a sub-interaction I' of I realizing G'

Method:

1. Process each goal rule of the domain theory to find one (say, G <= <P,H,Q>, corresponding to the goal G = $<P_0,Q_0>$) that is *partially satisfied*[1] by I.
2. Determine a sub-interaction I" corresponding to this goal rule. Let I" be the sub-interaction of I delimited by the first and the last of the action of I contributing to the realization of G. Let S_P and S_Q be the states occurring before the first action and after the last action, respectively.
3. Determine if the goal and the precondition and postcondition of the goal rule are achieved in the boundaries states, i.e. compute the boolean values $P(S_P)$, $P_0(S_P)$, $Q(S_Q)$ and $Q_0(S_Q)$.
4. Determine G' and I'. According to the four values computed in Step 3, and using the partial realization of the goal rule, apply heuristics H1 or H2 described below to propose the goal G', along with a sub-interaction I' of I (delimited by the states $S_{P'}$ and $S_{Q'}$) that achieves G'.

[1] A goal is partially satisfied, or partailly realized, if some of its subgoals are satisfied, and some are not. The ratio of satisfied goals to the total number of goals is the measure of partial satisfiability.

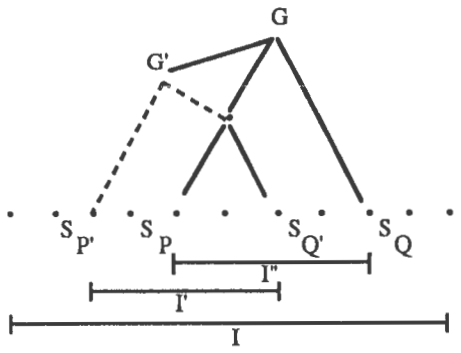


Figure 3. Illustration of the general algorithm.

To complete the refinement of Step 4 of the above generic algorithm, we now present heuristics H1 and H2 used by LEADER for proposing a candidate goal G'. The heuristic methods can be summarily described as follows (Figures 4 and 5):

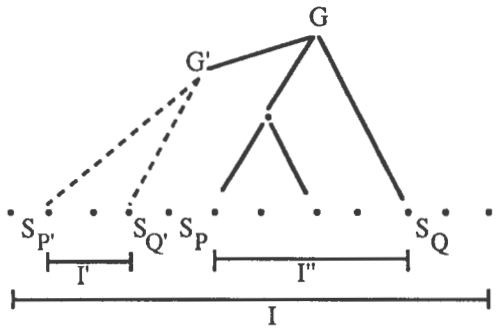


Figure 4. Illustration of Learning Heuristic H1.

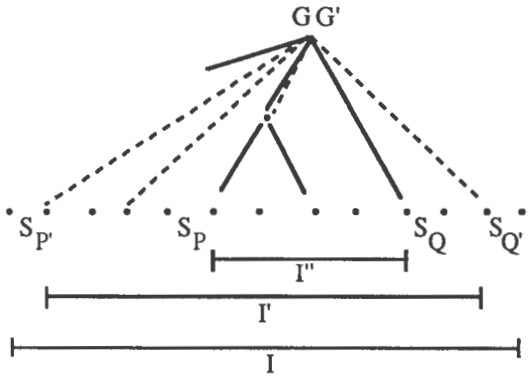


Figure 5. Illustration of Learning Heuristic H2.

Learning Heuristic H1 :

learning a new plan for a goal when the corresponding goal rule is partially satisfied.

This method can be applied when the hierarchical plan H' of a goal rule G' <= <P',H',Q'> is only partially realized, although the corresponding goal G = <P_0,Q_0> is realized. The method is the following:

- starting from S_P (resp. S_Q), the state corresponding to the leftmost (resp. rightmost) direct action appearing in the partially realized goal rule, go to the left (resp. to the right) to find a state $S_{P'}$ where P_0 is satisfied (resp. state $S_{Q'}$ where Q_0 is satisfied).
- the result is a goal G' = G = <P_0,Q_0> with the boundaries ($S_{P'}$ and $S_{Q'}$) of an interaction I' that achieves it.

Learning Heuristic H2:

learning a new hierarchical plan for the first task of a goal rule.

This method can be applied when the only missing task of a partially realized goal rule is the first one (leftmost). The assumption is the following: if the agent executed all but one of the tasks of a known goal rule, the subgoal of the missing task was probably achieved (because required in the known goal rule) in a way different as than expected. So, there might be a learning opportunity for acquiring a new hierarchical plan to achieve the subgoal in question. The method is as follows:

- identify the subgoal G' = <P',Q'> corresponding to the missing subtask
- starting from the state corresponding to the first (leftmost) direct action appearing in the partially realized goal rule, go backward (to the left) to find a state $S_{Q'}$ where Q' is satisfied (to cut down the search time while processing the states backward, a maximal search distance is specified)
- starting from the state preceding $S_{Q'}$, go backward again to find a state $S_{P'}$ where P' is satisfied.
- the result is a goal G' = <P',Q'> with the boundaries ($S_{P'}$ and $S_{Q'}$) of an interaction I' that achieves it.

Now that a candidate goal G' has been identified along with an interaction I' that achieves it, the next task is to extract from I the components relevant to G. This is the topic of the following section.

3.2 Identifying the hierarchical plan realizing a goal

This section briefly describes the second step of the process used to learn goal rules from unexplained actions, that is to say, the identification of the hierarchical plan involved in the realization of a goal. The identification first determines the relevant actions, then the relevant high level tasks, and finally the general (un-instantiated) hierarchical plan.

Rather than identifying relevant actions through a blind method such as the *loose-ends heuristic* used by cognitive scientists [Lewis, 1988] ("if an action cannot be explained, and if an expected command is not realized, then assume that the unexplained action is linked to the unrealized command"), LEADER employs a method more similar to goal regression: starting with the actions that directly contributed to achieve Q_0, it finds the actions whose postcondition contributed to achieve the precondition of the former actions, and so on, up to the action executed in the state $S_{P'}$.

The previous step results in an interaction I' (delimited by the first and last of these contributing actions) achieving G. However, I' might not be the most appropriate

hierarchical (readable and informative) because it is flat: the high level tasks are expanded in I' and are lost. To recover these high level goals, the goal graph is searched for trees whose leaves are actions of I'. These actions are then replaced with the root of these trees to produce a more structured hierarchical plan H'. This process is repeated on H' until no more actions can be replaced with high level tasks. The resulting H' is an instantiated hierarchical plan realizing the goal G.

The last step is the generalization of H'. This is done simply by applying the inverse of the instantiation used to the interaction. Note that a more powerful generalization technique might be used; however, the technique used must ensure that the hierarchical plan obtained still achieves the goal G.

3.3 Identifying a candidate goal

Now that the hierarchical plan is identified, its precondition and its postcondition are computed. The two problems can be formalized as follows:

- given a hierarchical plan, determine what must have held before its execution to allow its execution
- given a condition and a hierarchical plan executed over this condition, determine what holds after the execution.

The first problem is known as the *projection problem* [Elkan, 1989]; the second one is a version of the *retrojection problem* [ibid] (the determination of what must have held before the execution of a hierarchical plan, given what holds after it).

An efficient solution (in time and space) for both problems in the propositional case (in the absence of inferential literals) is described in [Pelletier and Matwin, 1992]. The idea is to reduce the computation of the conditions to operations on sets, and then to use bit vectors for representing sets. Solution for the general case of inferential literals can be found in [Pelletier, 1993].

4 Experimental verification

In previous sections, we have described general methods to acquire knowledge, in an unsupervised manner, to enhance the domain theory of a knowledge-based system.
This section illustrates the applicability of ideas described in previous sections. Two following hypotheses are verified experimentally:

- the learning system can be used to increase significantly the *relative coverage of a domain theory*, that is to say, the percentage of actions (over the interaction) explained by the domain theory. An action is *explained* if it is a *direct action*, i.e., it is a leaf of a goal tree (the action is part of a goal rule used to explain the interaction).

- the learning system can propose interesting enhancements for domain theories of practical domains.

Whereas the first hypothesis, the coverage, refers to a quantitative measure of LEADER, the second hypothesis is a qualitative evaluation of the potential of the learning methods.

To test the hypotheses, we have implemented a version of LEADER in Prolog and have conducted an experiment in two different domains: the execution of office tasks (to verify the first hypothesis), and the design of entity-relationship models (to verify the second one). The experiments are described in the next two sub-sections.

4.1 Generality of the domain theory

In order to evaluate LEADER's potential to increase the relative coverage of a domain theory, the protocol shown below was used. The main components of this protocol are elaborated in subsequent sections.

1. Identify an appropriate domain and its domain theory T (basic goals, literals, commands and hierarchical plans).
2. Define a training interaction $I_{training}$ for the domain identified in Step 1.
3. Run LEADER over $I_{training}$ for producing a new domain theory $T_{training}$.
4. Define new (testing) sets of interactions $I_{testing,k}$, preferably independent ones.
5. Compute the relative coverage of the domain theory T (i.e., before learning) over each of the interaction $I_{testing,k}$. Do the same with the domain theory $T_{training}$ (i.e., after learning).
6. Compute the relative coverage of the domain theory $T_{training}$ (i.e., after learning) over each of the interaction $I_{testing,k}$ and compare the coverage before and after learning.

The first step in the application of this protocol consists in selecting an appropriate domain. Because experts are often difficult (and costly) to find, and to automate the generation of training and testing instances, a tempting approach is to choose an artificial domain. To explore this avenue, we looked at the artificial domains described in [Kambhampati and Jengchin, 1993]. However, their lack of meaningfulness, their poor use of hierarchical goals and their constraints (or absence of constraints) on what and how actions can be grouped to achieve goals made them inappropriate for our needs. After considering suitability of some other artificial domains (such as video games), we have developed the Office Tasks domain. It describes the tasks, goals and actions arising in a typical office involving persons, information and physical tools. In this domain, a given number of persons have information that they want to exchange. This information takes the form of letters, or of any verbal information transmittable by phone (such as phone and fax numbers). For instance, to satisfy the goal *'transmit information'*, communication must be established (using a fax machine or a phone), the information must be transmitted, and the connection must be broken .

We selected this domain because it is interesting and meaningful, it has hierarchical goals and plans, there are many different ways to achieve some goals, and finally it is easy to judge the appropriateness of the knowledge produced by the learning methods.

The experiments were conducted over two different initial states: one involving 3 persons, 3 fax machines, 3

phones and 3 letters, and a more complex one involving 6 persons, 6 fax machines, 6 phones and 6 letters. For each initial state, experiments with interactions of length 50 to 300 were performed. For the more complex initial state, experiments were also performed with interaction's length of 600 and 900. For each different length and initial state, 1 training interaction and 10 testing interactions were randomly[2] generated, the knowledge produced by the training interaction was added to the initial domain theory and used on the testing sets. The statistics were collected, and the entire operation was repeated 20 times, each time with a new training interaction.

In addition to the coverage measurement, we were also interested in the representativeness of the interaction' actions (in comparison with the total number of possible actions, for a given initial state). Thus, the *variety* parameter, defined as the ratio of distinct actions appearing inside an interaction over the total number of possible actions for the given initial state, was also measured.

In order to examine the variety, we have recorded, for each initial state, the proportion of distinct actions present in the interactions (distinct/possible). Our experiments [Pelletier, 1993] show that the proportion of distinct actions increases (to converge toward 100%) with the interaction length, indicating an increase of representativeness of the actions.

The *relative gain of coverage* is presented in Figure 6. The graph shows the results for the two initial states. The observed gain varies from 5% to 13%. The second initial state provides a higher relative gain, indicating a better opportunity for learning.

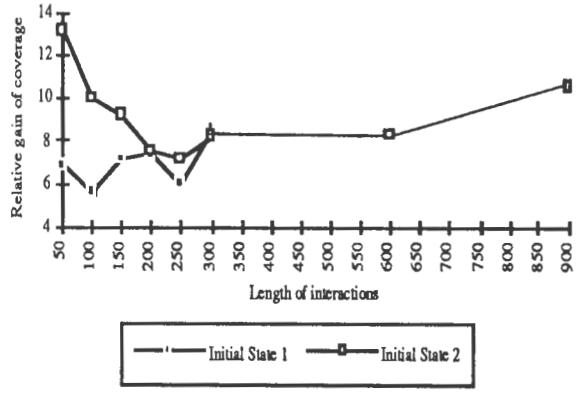


Figure 6. Relative gain of coverage for the two initial states.

As the results concerning the relative gain of coverage are statistically significant, the first hypothesis is verified: learning resulted in quantitatively important generalization of the domain knowledge, and a variety of available and observed actions was exploited during learning.

[2] With the biais that each interaction was executable over the initial state, and that none contained absurd actions that nobody executes in the real life (such as "P says something to P", "P phones P").

4.2 Application to a practical domain

This section presents an experimentation conducted to evaluate the second hypothesis, i.e., LEADER's capability for producing interesting goal rules. LEADER was used to enhanced Modeller, an assistant to a database designer (a complete description is provided in [Tauzovich, 1990]). As described below, the intended use of LEADER in this domain is to enhance the design rules of one of the Modeller's components: the expert system responsible of assisting the designer in the creation of entity-relationship (ER) models.

LEADER was applied at the conceptual design level: the enhancement of expert system's rules for helping the design process, checking for potential design flaws, etc. This level offers the best potential for learning because the result of the interaction with the user depends largely on her expertise (in contrast, most of the tasks of the two other expert system components can be automated).

To illustrate the application of LEADER to the conceptual design expert of the Modeller, consider the following high-level goal:

gen_entity(A,B,G) = "create a more general entity by creating a new entity and a dependency relationship".

The definition of the goal along with the associated part of the domain theory is provided in Figure 8 (in particular, the definition of a dependency relationship that is either a *role* or a *characteristic* relationship). Notice that in this initial domain theory, the only way known to create a dependency relationship is to create a *role* link, although a characteristic link is also a dependency relationship. An example of an instantiation of the first goal is pictured in Figure 7: to generalize the entity *teacher* by creating the more general entity *dept_employee* and by transferring the old link *related* between *teacher* and *department* to the new entity.

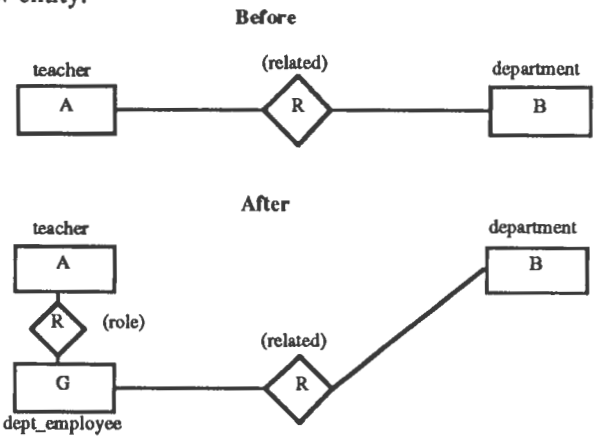


Figure 7. Graphical representation of the goal *gen_entity(A,B,G)* shown in Figure 8.

```
/* Definition of two goals */
gen_entity(A,B,G) = < P0, Q0 >, where
   P0 = entity(A) ∧ entity(B) ∧ ¬entity(G)
              ∧ relation(A, B, R)
   Q0 = entity(A) ∧ entity(B) ∧ entity(G)
              ∧ ¬related(A, B) ∧ relation(G, B, R)
              ∧ depend_on(A, B)

make_dependent_on(A,G) =
   < entity(A) ∧ entity(G) ∧ ¬depend_on(A,G),
     entity(A) ∧ entity(G) ∧  depend_on(A,G) >

/* Generalization of a relationship by creating a role
   relationship */
gen_entity(A,B,G) <=
   < entity(A) ∧ entity(B) ∧ ¬entity(G)
             ∧ relation(A, B, R),
   ( com(addE(G)),
     make_dependent_on(A,G),
     com(addR(G, B, R)),
     com(delR(A, B))  ),
     entity(A) ∧ entity(B) ∧ entity(G) ∧¬related(A, B)
      ∧ relation(G, B, R) ∧ depend_on(A, G) >

/* Creation of a dependency relationship by creating a
   role relationship */
make_dependent_on(A,G) <=
   < entity(A) ∧ entity(G) ∧ ¬related(A,G),
         ( com(addR(A, G, role)) ),
      entity(A) ∧ entity(G) ∧ relation(A,G,role) >
/* Some inferential literals */
related(Ent1, Ent2) <= relation(Ent1, Ent2, _).
depend_on(Ent1,Ent2) <= relation(Ent1,Ent2,role).
depend_on(Ent1,Ent2) <= relation(Ent1,Ent2,char).
```

Figure 8. Part of the initial domain theory for Modeller. The figure shows two goals (gen_entity(A,B,G) and make_dependent_on(A,G)), a goal rule for achieving each of them, and the definition of some inferential literals.

Then, LEADER was ran over an interaction (pictured in Figure 9) exhibiting a new hierarchical plan for achieving the goal *gen_entity(A,B,G)*: the generalization of an entity by creating a *characteristic* relationship. Note that the interaction also contained irrelevant actions to the goal, such as the creation of the entity *university*.

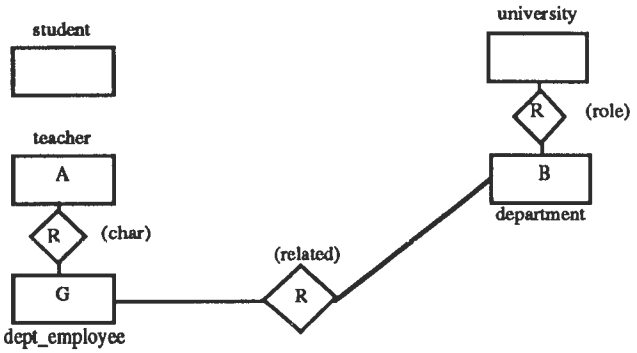


Figure 9. A new way of achieving the goal G = $<P_0,Q_0>$.

Using Learning Heuristic H1 (learning a new hierarchical plan for a goal when the corresponding goal rule is partially satisfied – Section 3.1), LEADER identified that the goal G = *gen_entity(A,B,G)* was achieved by the partially realized corresponding goal rules. Then, a new goal rule for G was proposed (this time by creating a characteristic relationship), as shown in Figure 10. This rule is something a designer may use to obtain a more suitable model than the model of Figure 7. Consequently, it should be explainable by the domain theory. The initial domain theory did not contain this rule, which has been learned by LEADER. LEADER can therefore, as was postulated in Section 4, learn useful rules. The same has also occurred in the office tasks domain, where, e.g., LEADER has learned that instead of redialing the phone number in order to communicate with same agent as previously, it is enough to press the 'repeat' button.

```
gen_entity(A,B,G) <=
  <entity(A) ∧ entity(B) ∧ ¬entity(G)
        ∧ ¬related(G,B) ∧ ¬related(A,G)
        ∧ relation(A, B, R),
    ( com(addE(G)),
      com(addR(A,G,char)),
      com(addR(G, B, R)),
      com(delR(A, B))  ),

     entity(A) ∧ entity(B) ∧ entity(G) ∧ ¬related(A, B)
        ∧ relation(G, B, R)
        ∧ relation(A, G, char) >
```

Figure 10. A goal rule produced by LEADER.

To summarize, this section has illustrated the application of LEADER on the domain of databases design to show how LEADER can enhance an initial domain theory about entity-relationship modelling by producing interesting and useful rules.

5 Conclusion

We have presented an approach to learning useful extensions to an existing body of domain knowledge from unobtrusive observation of an expert performing tasks in that domain. Our approach combines techniques from learning and planning to produce a more general theory of the domain than the one given at the outset. The learning component is

based on two heuristics that we have proposed. We have shown experimentally that: 1) the heuristics perform as expected, increasing the explanatory power of the domain theory, and that 2) the heuristics produce knowledge in form of goal rules that are useful in a real-life application in which LEADER learns rules of conceptual database design.

The work presented here puts forward a number of additional questions which could be investigated in the theoretical and experimental framework that we have developed. Does the approach extend to other deficiencies of the domain theory, e.g. inconsistency? How could a satisfactory theory be learned from its very scant initial version, or even from scratch? What sequences of interactions during training would result in faster learning? Is the number of rules learned by LEADER and presented for user's approval reasonable? Does it converge to a level determined by the quality of the initial theory? What steps could be taken to automatically pre-validate the rules produced by LEADER (see [Pelletier, 1993] for a discussion of the last question).

Acknowledgements

The work described here has been supported by the Natural Sciences and Engineering Research Council of Canada, the Government of Ontario (URIF and OTF Programs), the Department of Systems and Computer Engineering of Carleton University, the Département d'informatique de l'Université du Québec à Hull, Cognos Inc, Canada Centre for Remote Sensing, and Druide Informatique Inc.

References

Bruce Campbell. Office Clerk II, Final year project report, University of Calgary.

[Chapman, 1987] David Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333-377, 1987.

[DeJong and Mooney, 1986] G.F. DeJong and R. Mooney. Explanation-Based Learning: An Alternative View. *Machine Learning*, 1:145-176, 1986.

[Elkan, 1989] Charles Elkan. Incremental, Approximative Planning, Rapport KRR-TR-89-12, Department of Computer Science, University of Toronto, November 1989.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, (2):198-208, 1971.

[Ginsberg, 1989] Allen Ginsberg. Knowledge Base Refinement and Theory Revision. In Proceedings *of the Sixth International Workshop on Machine Learning*, pages 260-265, Ithaca, N.Y., 1989. Morgan Kaufmann.

[Hammond, 1989] Kristian J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task.* Perspectives in Artificial Intelligence, vol. 1. Academic Press Inc., 1989.

[Kambhampati and Hendler, 1990] Subbarao Kambhampati and James A. Hendler. A Validation Structure Based Theory of Plan Modification and Reuse. *Artificial Intelligence*, 1990.

[Kambhampati and Jengchin, 1993] Subbarao Kambhampati and Chen Jengchin. Relative Utility of EBG Based Plan Reuse in Partial Ordering vs Total Ordering Planning. In Proceedings *of the AAAI-93*, 1993.

[Levi, Perschbacher et al., 1988] Keith R. Levi, David L. Perschbacher and Valerie L. Shalin. Learning Plans and Information Requirements for Pilot Aiding. In Proceedings *of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1988.

[Lewis, 1988] Clayton Lewis. Why and How to Learn Why: Analysis-based Generalization of Procedures. *Cognitive Science*, 12:221-256, 1988.

[Mitchell and Mabadevan, 1990] Tom M. Mitchell and Sridbar Mabadevan. LEAP: A Learning Apprentice for VLSI Design. In *Machine Learning*, vol. III, pages 271-289, 1990.

[Pelletier, 1993] Bertrand Pelletier. Unsupervised Learning From a Goal-Driven Agent. Ph. D. Thesis, Department of Systems and Computer Engineering, Carleton University, 1993.

[Pelletier and Matwin, 1992] Bertrand Pelletier and Stan Matwin. Building Macros in Deterministic and Non-Deterministic Domains. In Proceedings *of the Ninth Canadian Conference on Artificial Intelligence*, pages 15-21, Vancouver, B.C., 1992. Morgan Kaufmann.

[Segre, 1988] Alberto Segre. *Machine Learning of Robot Assembly Plans*. Kluwer Academic Publishers, 1988.

[Tauzovich, 1990] Branka Tauzovich. An Expert System for Conceptual Data Modelling. In *Entity-Relationship Approach to Database Design and Querying*, pages 205-220. Elsevier Science Publisher, B.V. [North-Holland], 1990.

[Waldinger, 1977] Richard J. Waldinger. Achieving Several Goals Simultaneously. In *Machine Intelligence 8: Machine Representations of Knowledge*, pages 94-136. Ellis Horwood, Chichester, UK, 1977.

IDENTIFYING THE TRIGGER FEATURES FOR HIDDEN UNITS IN A PDP MODEL OF THE EARLY VISUAL PATHWAY

Michael R.W. Dawson, Stefan C. Kremer and Timothy N. Gannon

Biological Computation Project[1]
Department of Psychology & Department of Computing Science
University of Alberta
Edmonton, Alberta

ABSTRACT

Traditionally, PDP networks have been interpreted by examining patterns of connection weights. We propose an alternative pattern-based method, in which the trigger feature -- the stimulus that produces an optimal response in a processing unit -- is identified. This trigger feature can be identified for any size of network without an iterative search through a pattern space. We demonstrate the viability of this method by using it to interpret the structure of a PDP network trained to emulate certain aspects of early form perception. When the output units of this network are trained to respond like complex cortical cells, our trigger feature rule reveals that a significant number of hidden units develop trigger features that are characteristic of simple cortical cells.

THE NEURON DOCTRINE

For over half a century, neuroscientists have attempted to understand the visual system by mapping its receptive fields. This work has shown that (i) many neurons in the visual pathway have a retinal receptive field that defines a particular "trigger feature" to which these neurons respond optimally, and (ii) the more central the neuron, the more complex and abstract the trigger feature is likely to be (see Kandel, Schwartz & Jessel, 1991, Chaps. 28 - 30).

These results led Barlow (1972) to propose a *neuron doctrine* for perceptual psychology. The primary implication of the neuron doctrine is that in order to intepret the role of specific cells in the visual system, one must find the stimulus pattern which best matches a cell's receptive field, and which thus produces maximum cell activity.

Unfortunately, this interpretive strategy is extremely difficult to apply to complex biological networks, because an extremely large pattern space must be explored to find a cell's trigger feature. As a result, biological systems are usually studied by trial and error (e.g., Hubel & Wiesel, 1962, p. 145), and this has led to considerable debate about whether particular features are optimal (e.g., De Valois, Albrecht & Thorell, 1978).

THE TRIGGER FEATURE RULE

In contrast, Barlow's (1972) interpretive strategy is extremely easy to apply to PDP networks. If a network uses monotonic activation functions, then the pattern of connections fanning in to any network processor defines a unique trigger feature. This trigger feature can be identified for any size of network without an iterative search through a pattern space.

Assume that a processing unit uses a monotonic activation function like the logistic, and computes its net input by summing the weighted signals fanning in from the connections that define its receptive field. The monotonicity of the activation function implies that the processor's maximum activity will be generated when it receives the maximum net input possible. *This occurs when the highest possible signal value is transmitted through each connection that has a positive weight, and the lowest possible signal value is transmitted through each connection that has a negative weight.*

For example, consider a hidden unit connected to a set of input units that can adopt activation values that can range from 0 to 1. The trigger feature for this hidden unit is simply the activation pattern that transmits a 1 through every positive connection and a 0 through every negative connection fanning in to it.

AN EXAMPLE APPLICATION

Typically, PDP networks are interpreted by applying techniques that analyse patterns of connectivity (for a re-

[1]This research was supported by NSERC Operating Grant A2038 and NSERC Equipment Grant 46584 awarded to the first author, by an NSERC Postgraduate Scholarship awarded to the second author, and by an NSERC Undergraduate Research Assistanship awarded to the third author. Correspondence can be addressed to Dr. Michael Dawson, Biological Computation Project, Department of Psychology, University of Alberta, Edmonton, Alberta, CANADA T6G 2E9. E-mail: mike@psych.ualberta.ca

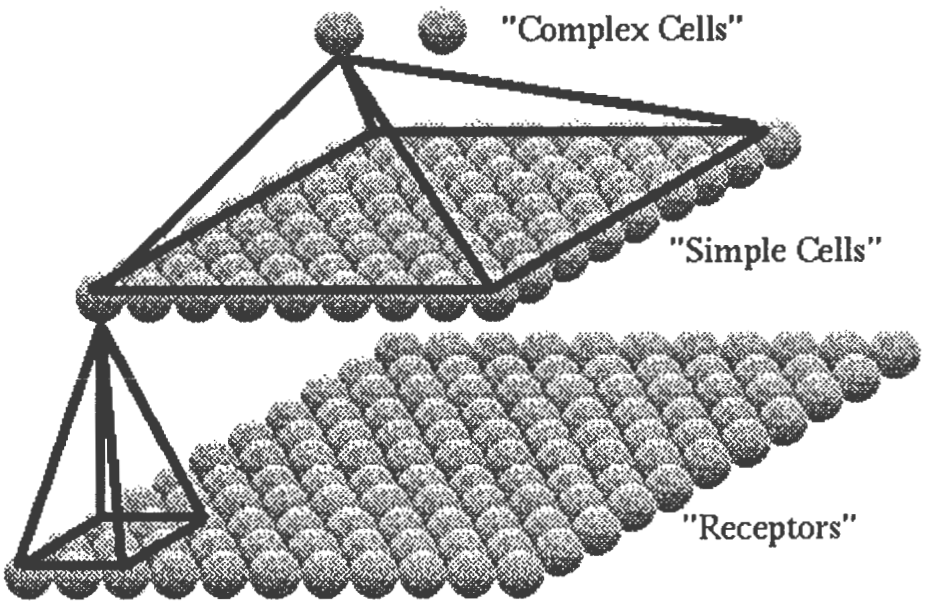


Figure 1
An illustration of the experimental network trained in both studies. In this network, each output unit is connected to every hidden unit ("simple cell"), but each hidden unit is only connected to a 3X3 array of input units ("receptors"). In each study a comparison network was also used. It was identical to the one illustrated above with the exception that each hidden unit was connected to every input unit.

view, see Hanson & Burr, 1990). The ease of applying the neuron doctrine suggests an alternative, pattern-based approach to interpretation: identifying the trigger feature for each network processor. However, the success of this approach depends on two additional factors. First, while one might be able to quickly determine a processor's trigger feature, the maximum response of this processor might still be extremely weak, and the processor may not play a major role within the network (cf. Mozer & Smolensky, 1989). Second, Barlow's (1972) neuron doctrine must apply to the network being interpreted. If processing units do not function as feature detectors, as would possibly be the case for a network trained to approximate a function, then identifying trigger features would be unlikely to aid network interpretation.

On the one hand, several observations suggest that one domain in which both of these limitations may be avoided is the one for which Barlow (1972) originally proposed the neuron doctrine: vision. First, in spite of the increased attention being paid to distributed representations by neuroscientists (e.g., Georgopoulos et al., 1989; Richmond & Optican, 1992), many modern researchers still attempt to understand biological visual systems by identifying trigger features for individual cells (for reviews of this type of work, see Kandel, Schwartz & Jessell, 1991, Chap. 30; Maunsell & Newsome, 1987; Rolls, 1990). Second, the most successful attempts to relate the structure of trained PDP networks to that of biological systems have compared the receptive fields of individual processing elements (e.g., Zipser & Anderson, 1988).

On the other hand, there are no guarantees that a PDP network trained to process visual stimuli will be governed by the neuron principle. For example, Moorhead, Haig and Clement (1989) trained a network consisting of a 5X5 array of input units, 2 or 3 hidden units, and 4 or 8 output units. The input units represented oriented bars and edges after they had been passed through centre-surround filters. The output units were trained to behave like simple cells, detecting particular orientations at specific input locations. Moorhead et al hypothesized that the hidden units would develop centre-surround receptive fields because they were assumed to be analogous to cells in the lateral geniculate nucleus. However, spotmapping of the hidden unit receptive fields failed to reveal this type of organization. Moorhead et al. concluded that there were substantial shortcomings in the use of artifical neural networks to model the visual pathway.

To test the usefulness of our rule for identifying trigger features, we adopted an approach analogous to that of Moorhead, Haig, and Clement (1989), and trained a PDP network to perform as a highly simplified model of early visual processing. However, in contrast to Moorhead et al.'s decision to use a very small number of hidden units connected to every input unit, we adopted a more biologically plausible design in which a larger number of hidden units existed, but each of these received input from a very small number of processors.

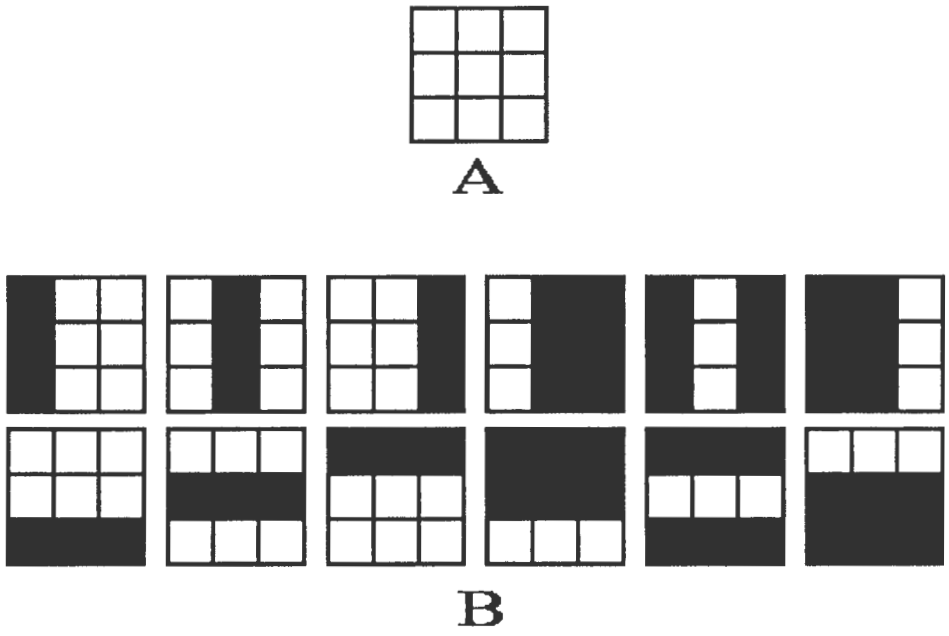


Figure 2
(A) A 3X3 array representing the receptive field of a hidden unit in one of the experimental networks. The trigger rule being applied in this study assumes that such receptive fields must be binary in nature -- each cell must be either on or off. As a result, each hidden unit has the potential to develop one of 512 possible receptive field organizations. (B) Of the 512 possible receptive fields, only these 12 were considered to be representative of simple cells. As is detailed in the text, a statistically significant number of these receptive fields were observed in the two experimental networks.

The network, which is illustrated in Figure 1, consisted of three layers of processors. The first layer was an 11X11 array of input units that served as the network's "retina". The second layer was a 9X9 array of hidden units. Each of these hidden units was connected to a 3X3 receptive field of input units; neighbouring hidden units had overlapping receptive fields. This pattern of connectivity was used because it is unreasonable to assume that every cell in the visual pathway has direct connections to every retinal ganglion cell. The third layer consisted of two output units. Each of these output units was connected to every hidden unit in the second layer of processors. During training, one of these units was trained to respond whenever a horizontal line segment was presented to the input units. The other unit was trained to respond whenever a vertical line segment was presented to the input units. As a result, these output units were intended to be analogous to complex cells in the visual cortex.

In our first experiment, a very small stimulus set was used. Each stimulus was a bar that had a width of 1 pixel and a length of 9 pixels, and had either a horizontal or vertical orientation. These stimuli were presented anywhere in the 9X9 centre of the 11X11 array of input units. As a result, the total stimulus set consisted of 9 different horizontal bars, 9 different vertical bars, as well as a null stimulus in which no input units were activated. The horizontal output unit was trained to respond to any of the horzontal stimuli, the vertical output unit was trained to respond to any of the vertical stimuli, and both units were trained to not respond to the null stimulus. The generalized delta rule (Rumelhart, Hinton, & Williams, 1986) was used to train the network; a learning rate of 0.25 and a momentum value of 0.10 were adopted. The network converged after 797 iterations.

With the input units serving as a simple retina, and with the output units trained to respond like complex cells, we hypothesized that processors in the hidden unit layer would adopt receptive fields analogous to those that characterize simple cells in the visual cortex. If binary inputs are presented to the first layer of processors, then only a small number of possible trigger features would characterize simple cell responses in the layer of hidden units. These possible trigger features are illustrated in Figure 2. After training the network, we applied the trigger feature rule described above to the receptive field of each hidden unit to determine whether any of these units had acquired a trigger feature that belonged to the set illustrated in Figure 2.

The results of the first simulation indicated that 13 of the 81 processing units in the hidden layer had acquired one of the trigger features illustrated in Figure 2, and thus

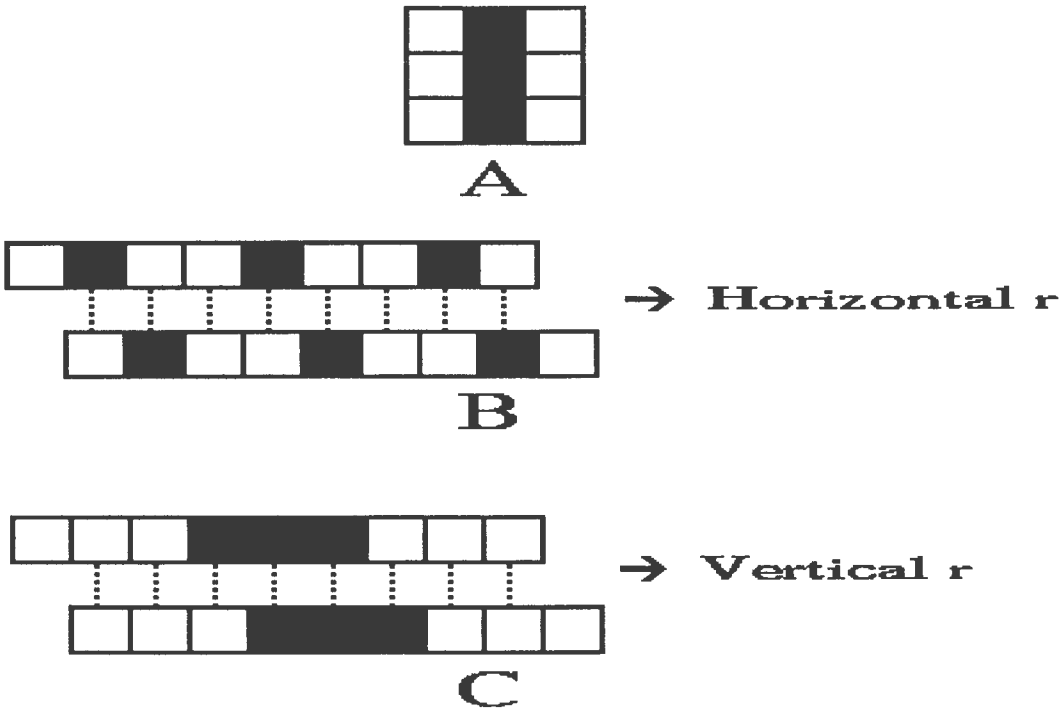


Figure 3
A 3X3 receptive field is used to illustrate how horizontal and vertical systematicity was computed for the large receptive fields in the comparison networks. (A) A receptive field to be measured, as determined by the trigger feature rule. (B) The two dimensional array from A is strung out horizontally as a one dimensional array. A copy is made, shifted one unit to the right, and the two are correlated. We call this the lag-1 horizontal autocorelation. (C) The lag-1 vertical autocorrelation is computed for the same receptive field by stringing it out vertically. The vertical correlation would be substantially higher than the horizontal correlation for this particular receptive field, indicating that it has substantially vertical organization.

could be characterized as being analogous to a simple cell. The binomial test (Siegel, 1956, pp. 36-42) indicated that the likelihood of observing this many simple cells due to chance alone was extremely small ($p < 4.82e\text{-}08$).

A second experiment was performed to determine whether an increased population of simple cells would be observed if a larger stimulus set was employed. This second experiment was identical to the first, with the exception that stimulus bars were 1 pixel wide and only 3 pixels long. As a result, the entire stimulus set consisted of 63 horizontal bars, 63 vertical bars, and the null stimulus. The network converged after 4395 iterations. Again, the trigger feature rule described above was applied to interpret the function of each hidden unit. Of the 81 hidden units, 27 had developed a trigger feature that belonged to the set illustrated in Figure 2, and thus was analogous to a simple cell. The odds of observing this many simple cells due to chance are extremely small, as revealed by the binomial test ($p < 5.98e\text{-}24$).

The results above indicate that the trigger feature rule reveals significant numbers of biologically plausible receptive fields in the network. This result contrasts sharply with that of Moorehead, Haig, and Clement (1989). One reason for this difference might have been our decision to greatly restrict the pattern of connectivity between the hidden units and the input units. To test this possibility, we trained a second network on the two sets of patterns. This comparison network was identical to the one illustrated in Figure 1, with the important exception that each hidden unit was connected to every input unit. The comparison network learned the small stimulus set in 108 iterations, and learned the larger stimulus set in 262 iterations.

The trigger feature rule was applied to determine the receptive fields for each of the hidden units after training. None of the receptive fields were consistent with a strict definition of a simple cell (i.e., we failed to produce any 11X11 analogs of the receptive fields illustrated in Figure 2). We were concerned, however, that with these larger receptive fields our definition of a simple cell receptive field was too conservative. As a result, we quantified the horizontal and vertical systematicity of each receptive field to provide a more liberal definition. To do this, we took each receptive field (as determined by the trigger feature rule), strung it out horizontally and vertically, and computed lag-1

autocorrelations as illustrated in Figure 3. Prior empirical tests had shown us that, for example, a perfect horizontally organized receptive field would produce a horizontal correlation coefficient in the order of 0.90 and a vertical correlation coefficient in the order of -0.10 . None of the receptive fields in the trained comparison networks approached this high level of systematicity. Collapsing over the two training sets, the best horizontal receptive field had a horizontal correlation coefficient of 0.63 and a vertical correlation coefficient of -0.18. The best vertical receptive field had a vertical correlation coefficient of 0.50 and a horizontal correlation coefficient of -0.17. In short, it appears that one of the major reasons that the experimental networks produced significant numbers of simple cell receptive fields was the fact that connections between hidden units and input units were restricted.

CONCLUSIONS

There are two main conclusions to be drawn from the simulations described above. First, the trigger feature rule that we have developed can be usefully applied when interpreting the structure of trained PDP networks. In two separate simulations, this rule revealed that a statistically significant number of hidden units had developed a biologically plausible receptive field, responding to the types of trigger features that simple cells in the visual cortex respond to. Second, Moorhead, Haig and Clement's (1989) conclusion that PDP networks are not useful tools for the study of biological vision systems is clearly premature. When they used a very small number of hidden units, each connected to every input unit, spotmapping did not reveal receptive fields similar to neurons in the visual system. In contrast, with our more biologically plausible assumption of a larger number of hidden units, each with a small window on the input array, our more efficient trigger feature rule indicated a substantial amount of structure consonant with biological networks.

REFERENCES

Barlow, H.B. (1972). Single units and sensation: A neuron doctrine for perceptual psychology? *Perception, 1,* 371-394.

De Valois, R.L., Albrecht, D.G., & Thorell, L.G. (1978). Cortical cells: bar and edge detectors, or spatial frequency filters? In S.J. Cool & E.L. Smith (Eds.) *Frontiers in visual science.* New York: Springer-Verlag. (pp. 544-556).

Georgopoulos, A.P., Lurito, J.T., Petrides, M., Schwartz, A.B., & Massey, J.T. (1989). Mental rotation of the neuronal population vector. *Science, 243,* 234-236.

Hanson, S.J., & Burr, D.J. (1990). What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and brain sciences, 13,* 471-518.

Hubel, D.H., & Wiesel, T.N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of physiology, 160,* 106-154.

Kandel, E.R., Schwartz, J.H., & Jessell, T.M. (1991). *Principles of neural science, Third edition.* New York: Elsevier

Maunsell, J.H.R., & Newsome, W.T. (1987). Visual processing in monkey extrastriate cortex. *Annual review of neuroscience, 10,* 363-401.

Moorhead, I.R., Haig, N.D., & Clement, R.A. (1989). An investigation of trained neural networks form a neurophysiological perspective. *Perception, 18,* 793-803.

Mozer, M.C., & Smolensky, P. (1989). Using relevance to reduce network size automatically. *Connection science, 1,* 3-16.

Richmond, B.J., & Optican, L.M. (1992). The structure and interpretation of neuronal codes in the visual system. In H. Wechsler (Ed.) *Neural networks for perception, V.1.* Boston: Academic Press.

Rolls, E.T. (1990). Principles underlying the representation and storage of information in neuronal networks in the primate hippocampus and cerebral cortex. In S.F. Zornetzer, J.L. Davis, & C. Lau (Eds.) *An introduction to neural and electronic networks.* San Diego: Academic Press.

Siegel, S. (1956). *Nonparametric statistics for the behavioural sciences.* Toronto: McGraw-Hill.

Zipser, D. & Anderson, R.A. (1988). A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature, 331,* 679-684.

ARTSTAR: A Supervised Adaptive Resonance Classifier

Talib S. Hussain and Roger A. Browse
Department of Computing and Information Science
Queen's University, Kingston, Ontario*

Abstract

A new neural network architecture, ARTSTAR, is presented as a supervised modular extension to the ART2 network. ART2 suffers from deficiencies in terms of consistency and overall capability when applied to classification tasks. ARTSTAR uses a layer of INSTAR nodes to supervise and integrate multiple ART2 modules. Supervision takes the form of feedback to the ART2 output layer whenever a data pattern's true classification is known. This feedback technique may take a variety of forms and can model the supervision implemented in existing supervised extensions to ART networks. A more robust classification performance occurs when several ART2 networks are trained in a supervised manner, each under different conditions, and their outputs integrated during testing. These results are demonstrated in tests of ARTSTAR using handdrawn and computer generated digits. The general functionality of ARTSTAR is extensive, and several further modifications to it are discussed.

1 Introduction

Several different unsupervised neural network architectures have been proposed based on the concept of Adaptive Resonance (Carpenter & Grossberg, 1987a; 1987b; 1990, Carpenter, Grossberg, & Reynolds, 1991; Carpenter, Grossberg, & Rosen, 1991a; 1991b, Carpenter, Grossberg, Markuzon, Reynolds & Rosen, 1992). This class of network models was devised to examine the effect that feedback connections have on the formation of categorizations of input data. In general, these models propose that self-organization of data can be achieved through a mechanism which forms a bottom-up interpretation of a given input and then, based on previously learned patterns, forms top-down expectations as to how the input should be categorized until the interpretations and expectations match, or "resonate", within a certain tolerance level, or "vigilance". Carpenter and Grossberg (1987a) first developed the ART1 network, which accepts binary input, and later extended their work to the ART2 network, which accepts analog input (Carpenter & Grossberg, 1987b).

The ART2 network is ideally suited for tasks requiring data patterns to be clustered into groups of similar elements, and for this purpose it is comparable to conventional clustering techniques (Burke, 1991; Baruah & Welti, 1991). Further, ART2 is a self-organizing network capable of dynamic, on-line learning, and can thus learn to modify its clustering schemes to reflect changes in data characteristics over time. Because of ART2's clustering capabilities, one might expect that if an ART2 network were presented with patterns known to belong to certain pre-determined classes, the network would learn to categorize the data into groups equivalent to those classes. In its basic form, however, ART2 does not generally perform well on classification tasks. One reason is that ART2 does not have the provision to accept supervision, and thus could hardly be expected to form a classification scheme which depicts predesignated classes as well as other supervised networks such as backpropagation. Another difficulty with ART2 as a classifier is that the categorizations developed by ART2 are very sensitive to slight changes in structure and training conditions. For instance, two identical networks trained on the same set of data but presented in different orders may exhibit greatly differing classification performance.

* This work was supported by the Natural Sciences and Engineering Research Council (NSERC) and by a grant from the Institute for Robotics and Intelligent Systems (IRIS).

The most obvious step necessary to improve ART2 performance in classification is the incorporation of supervision. Previous research on supervision extensions for ART networks show two possible supervision techniques. One approach involves forcing each ART category formed to respond to data from only one pre-established class during training. This change greatly decreases the chance that data from different classes will be categorized as the same during subsequent testing and thus improves ART performance as a classifier. Three different networks have been developed which implement this approach, generally through an architectural extension to ART. They are ARTMAP (Carpenter, Grossberg & Reynolds, 1991), the Adaptive Resonance Associative Map, or ARAM (Tan, 1992), and the SeMi-supervised Adaptive Resonance Theory, or SMART2 (Merz, St. Clair & Bond, 1992). The latter network also introduces a second technique for supervising ART, which is that the number of categories formed by the network is constrained by allowing new categories to be created only when a pattern is initially misclassified. The classification performance of ARAM and SMART2 has actually been found to be comparable to that of backpropagation (Tan, 1992; Merz et al., 1992).

A second method of improving ART2 classification ability involves the use of redundancy to overcome the sensitivity of the ART2 categorization schemes. The assumption of this approach is that ART2 networks which are trained slightly differently will develop different categorization schemes and that these schemes will contain complementary information which can be integrated to achieve a more robust representation. Several methods of exploiting redundancy in multiple classifiers are available, with the simplest being a majority classifier voting approach (Gargano, 1991; Carpenter et al., 1992). In this technique, the classification returned by the majority of classifiers, each applied to the same input, is considered to be the classification of that input.

This paper presents the ARTSTAR network. The development of ARTSTAR has been motivated by the following goals:

1. to provide an ART-based network capable of effective classification, yet still retaining ART's inherent ability to respond to ongoing changes the organization of its inputs.
2. to provide a supervision strategy which is general enough to include previously developed mechanisms and yet offer the opportunity for incorporating new supervision techniques.
3. to provide a mechanism that can integrate results from several different ART2 modules.

An ARTSTAR network consists of a number of ART2 modules connected to a layer of INSTAR nodes (Grossberg, 1982), with each INSTAR node representing a possible class. The INSTAR layer supervises the output layer of each ART2 module as well as integrates the outputs of all the ART2 modules. During training, the instar layer provides feedback to the ART2 modules based on the desired classification of the training input. The feedback influences the importance that each module assigns to its output nodes, and thereby affects the order in which each ART2 module considers its output nodes as possible winners. For each winning output node selection in an ART2 module, INSTAR learning associates it with the INSTAR node that represents the desired class. Over time, the connections between the ART2 output nodes and INSTAR layer come to represent the incidence of a given ART2 category being associated with a given class.

After learning has taken place, the winning category of an ART2 module will activate all INSTAR nodes in proportion to the probability of that class being correct. Thus, if a given input pattern is presented to a number of redundant modules during testing, each redundant module will return a list of class probabilities. Through the simple method of summing the probabilities for each class, ARTSTAR integrates these redundant responses and presents a single list of ranked classifications.

The feedback mechanism of ARTSTAR permits the manipulation of system parameters which provides a variety of different supervision strategies, including the 'forcing' method used in ARTMAP, ARAM and SMART2. Further, each of these strategies will exhibit a desirable 'dynamic supervision' due to an inheritance of the dynamic learning properties of ART2. Thus, an arbitrary ARTSTAR network may be trained on data in which true classifications are available only periodically, resulting in two interleaved learning phases: simple ART2 clustering periods in which new instances are added to existing categorization schemes, and supervision periods in which the classifications associated with ART2 categories are verified and updated. The generality of the feedback mechanism, coupled with ARTSTAR's ability to perform dynamic supervision and integrate outputs from separate ART2 modules, suggests that ARTSTAR may be applicable to a wide range of tasks.

2 ARTSTAR Network Architecture

ARTSTAR derives its name from its use of both ART2 and INSTAR modules thus it is appropriate to begin the description of ARTSTAR with a brief outline of the ART2 network.

2.1 ART2 Processing

The ART2 network consists of three main components, termed by Carpenter and Grossberg (1987b) as the input representation field (or F1 layer), the category representation field (or F2 layer) and the orienting mechanism (see Figure 1).

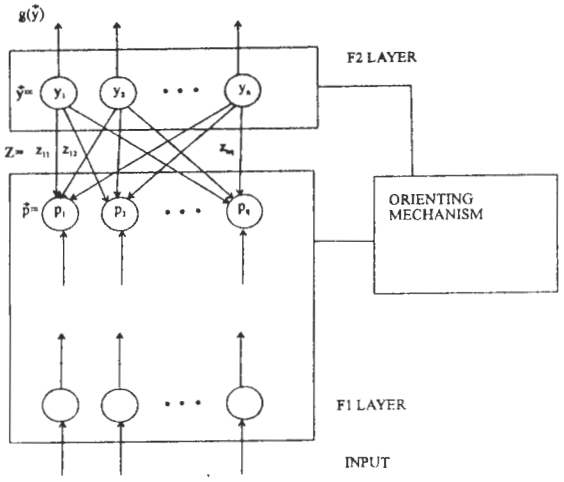


Figure 1: A simplified diagram of ART2 architecture

The nodes of the output layer $y=(y_1\ y_2\ \ldots\ y_n)$ are intended to represent the classes into which input patterns are organized. Each output node represents a class of patterns by storing a template pattern as the weights $Z=(z_1\ z_2\ \ldots\ z_n)^T$, $z_j=(z_{j1}\ z_{j2}\ \ldots\ z_{jq})$ on its connections into the F1 layer. During an ART2 trial, the input pattern is matched against each of the stored templates, resulting in activations at the output layer which represent the extent of match. At this stage of processing the output layer activations are established as the weighted sum of a layer $p=(p_1\ p_2\ \ldots\ p_q)$ within the representation field F1:

$$y=p\cdot Z^T$$

The element of y with the highest activation value is designated as the initial choice of class for the input pattern. The results of this competition, usually denoted as $g(y)$ presents a value d ($0<d<1$ for a given network) for the winning node, and zero otherwise.

The input pattern and the template for the initially chosen node are subjected to a further comparison in the orienting mechanism, and if the match is judged to be within the vigilance level, the initial choice is taken as final and the template for that winning node is updated to more strongly represent the current input. If the match is judged to be outside of the vigilance level, then the output node with the next highest activation value is designated as the initial match and it is subjected to the same processing in the orienting mechanism to determine if it is an acceptable final choice. This process continues through the available choices, in decreasing order of the activation of the output nodes, and if none of them meets the vigilance level test, then a new output node is recruited as a new class and its template is set to the current input.

In ART2 processing, there is no distinction between training and test trials. Every trial results in a classification (the final choice of output node), and with each classification, there is the possibility of updating the weights which represent the stored templates.

The choice of vigilance level will strongly influence the performance of ART2 in the formation of its classification categories. A low vigilance will result in over-inclusive elements, and a high vigilance could, in the worst case, result in a different classification category for each different training pattern.

2.2 Supervision

ARTSTAR incorporates two key properties in addition to those inherited from ART2. The main property is supervision of the ART2 learning process. To achieve this, ARTSTAR includes a layer of INSTAR nodes, one for each true category of the input patterns. The INSTAR nodes both receive input from and send feedback to the ART2 output layer. Supervision is achieved through feedback based on previous associations of ART2 categories with classes, knowledge which is stored in the INSTAR's incoming and outgoing connections.

Consider, as shown in Figure 2, the full connection of a single output element y_i to this layer of INSTAR nodes. In this example, there are weights w_i on the connections for each of the INSTAR elements as shown. Through INSTAR learning during the training

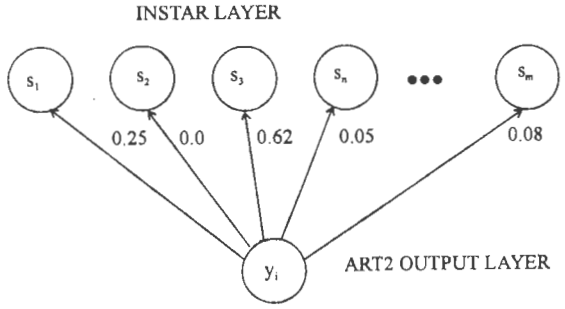


Figure 2: Example connections from an ART2 output node to the INSTAR layer

trials, these weights are set to designate the proportion of training trials for which y_i was the winning ART2 node when each of the true classes was the one associated with the training trial. This means that after training is complete and a test trial is being considered for which y_i is the winning output node, the system can provide an ordered list of possible classifications based on the values of these weights w_i. If required to establish a single class, then the obvious choice is the class possibility whose INSTAR node has the highest activation, and was thus during training most often associated with the winning output node.

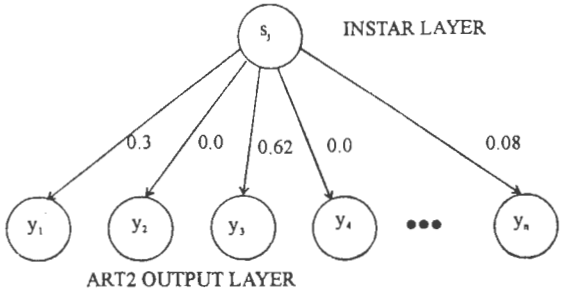


Figure 3: Example connections from INSTAR layer to ART2 output layer

Also consider what it means to have these weights W available during the training phase of ARTSTAR. As shown in Figure 3, the true class of any training trial would have available through W the extent to which previous trials of the same class had been stored on each of the ART2 module output templates. It is possible to use these values to influence the ART2 module to make an initial choice of an output node that has already been chosen most often for training trials of the same true class. The most straightforward way to accomplish this is to augment the computation of the values of the ART2 output layer y with feedback from the INSTAR layer:

$$y = p \cdot Z^T + \gamma s \cdot W^T$$

2.3 Modularity

The second key property introduced in ARTSTAR is modularity. The basic ARTSTAR network can easily be extended such that a number of ART2 networks are connected to the instar layer (see Figure 4). In the modular ARTSTAR, each ART2 module is supervised and associated with pattern classes as in the basic ARTSTAR, but in addition, the class associations of all the winning ART2 output categories are integrated to form a single output. Specifically, the likelihood that the input is of a given class is determined by summing the probabilities for that class indicated by all the winning ART2 nodes. This integration method is similar to the majority classifier voting scheme, and can emulate it under certain training conditions.

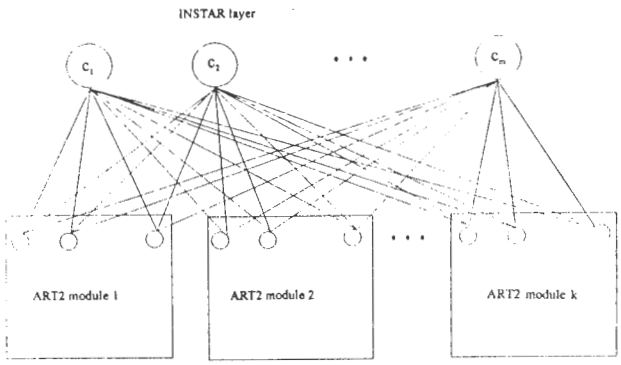


Figure 4: ARTSTAR architecture supporting multiple ART2 modules

2.4 ARTSTAR Processing

The most simple ARTSTAR network consists of a single ART2 module, slightly modified to incorporate feedback, a threshold field, the INSTAR layer, and a feedback field as shown in Figure 5.

The threshold field $\vec{t} = (t_1\ t_2 \ldots t_n)$ is a layer of n nodes, connected individually to the corresponding

ART2 output nodes $\vec{y}=(y_1\ y_2\ \ldots\ y_n)$. Each threshold field node accepts the corresponding ART2 output value and simply thresholds it to 0 or 1 as follows:

$$t_j = \begin{cases} 1 \text{ if } g(y_j) > 0 \\ 0 \text{ otherwise} \end{cases}$$

This step is required because the ART2 module outputs either 0 or d, and ARTSTAR requires the output to be either 0 or 1.

The instar layer $\vec{s}=(s_1\ s_2\ \ldots\ s_m)$ consists of m INSTAR nodes which are fully connected from the threshold field via the weights $W=(\vec{w}_1\ \vec{w}_2\ \ldots\ \vec{w}_n)^T$, $\vec{w}_j=(w_{j1}\ w_{j2}\ \ldots\ w_{jm})$ and which can also accept input from a classification vector $\vec{c}=(c_1\ c_2\ \ldots\ c_m)$. The classification vector is a binary vector with only one non-zero component which indicates the class, and therefore which INSTAR node should be active, when the class of the input is known. During training, each INSTAR node accepts weighted input from the threshold field and performs INSTAR learning upon the weights W as follows:

$$\vec{s}=\vec{t}\cdot W$$

$$w_{ij}=w_{ij}+\lambda_2[t_i c_i - w_{ij}]U(t_j)$$

$$U(t_j)=\begin{cases} 1 \text{ if } t_j>0 \\ 0 \text{ otherwise} \end{cases}$$

where λ_2 is a small learning rate. The output of the INSTAR nodes s_i' varies. If the layer is being trained, each INSTAR node returns the desired output, and if not being trained each node returns the weighted sum of its inputs.

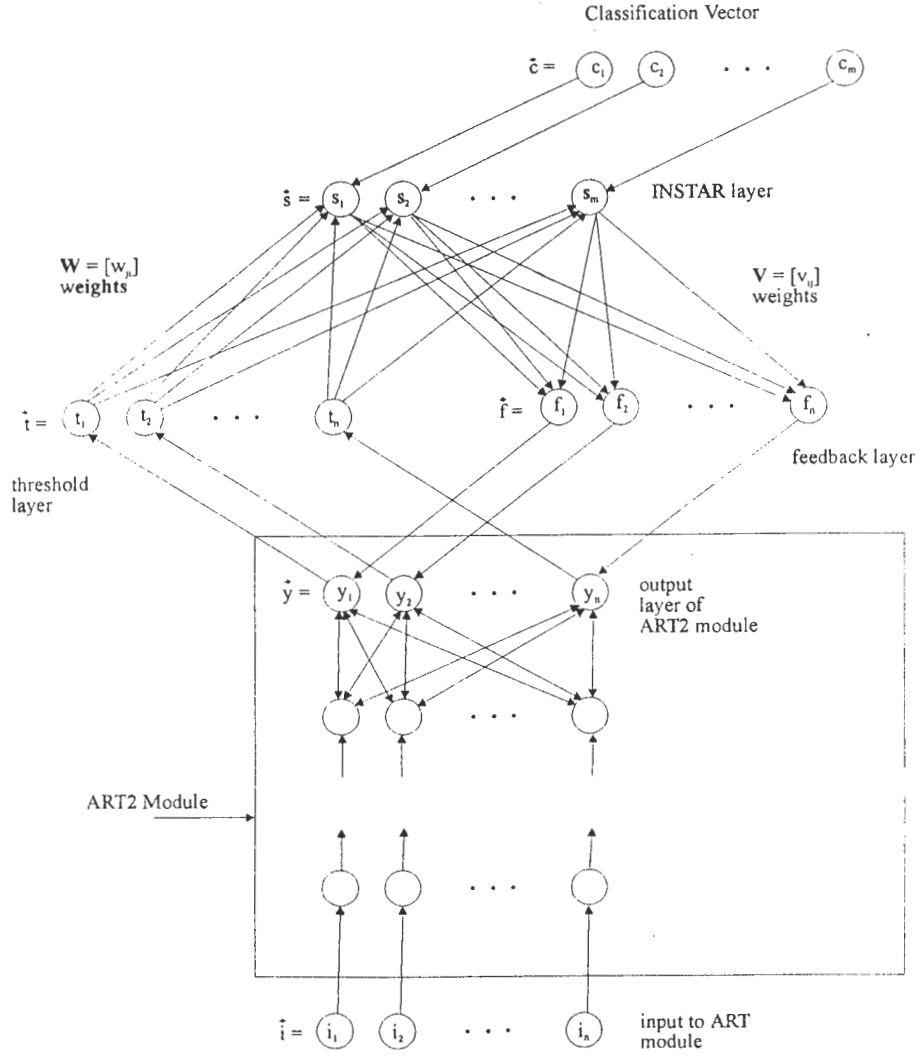


Figure 5: Schematic diagram of ARTSTAR architecture

The feedback field $\vec{f}$ consists of a layer of n nodes which are fully connected from the INSTAR layer by the weights $V = W^T$. This feedback layer directs the supervision of the ART2 module rather than accomplishing it with direct connections from the INSTAR layer as suggested in Section 2.2. This provides threshold control over the feedback that reaches the ART2 module permitting a variety of supervision strategies. The following two-threshold function is used:

$$\vec{f} = \begin{cases} \alpha & \text{if } \vec{s}' \cdot W^T < \theta_2 \\ \beta & \text{if } \vec{s}' \cdot W^T > \theta_1 \\ \vec{s}' \cdot W^T & \text{otherwise} \end{cases}$$

where $\alpha \geq 0$, $\beta \leq \alpha$, $\theta_1 \leq \alpha$ and $\theta_2 \geq \beta$. Then the output layer of the ART2 module is influenced by computing its activation level with:

$$\vec{y} = \rho \cdot Z^T + \gamma \vec{f}$$

This process influences the results of the ART2 network by changing the order in which ART2 considers possible winners. The manner in which the feedback is used by the ART2 modules is justified based on the previous research on supervising ART2. The primary effect of match-tracking in ARTMAP (Carpenter, Grossberg & Reynolds, 1991), the dual-resonance in ARAM (Tan, 1992), and the first design principle of SMART2 (Merz et al., 1992) is to eliminate those nodes which have been associated with a previous class. Ideally, of course, such an inhibitory method is most efficient if the eligible nodes are considered before other nodes. This interpretation suggests that the main goal of feedback during training should not be to eliminate nodes as they are considered in turn, but to minimize the number of nodes that are eliminated in a trial by considering the most-eligible nodes first. The manner in which INSTAR feedback is incorporated into the ART2 equations for the F2 layer is based upon this interpretation.

There are three straightforward forms that the feedback function could take:

1. *constant*: all committed nodes receive equal feedback,
2. *direct*: nodes receive feedback in direct relation to the strength of their corresponding instar weight,
3. *inhibitory*: nodes receive non-zero feedback if and only if they represent the desired class (see Figure 6).

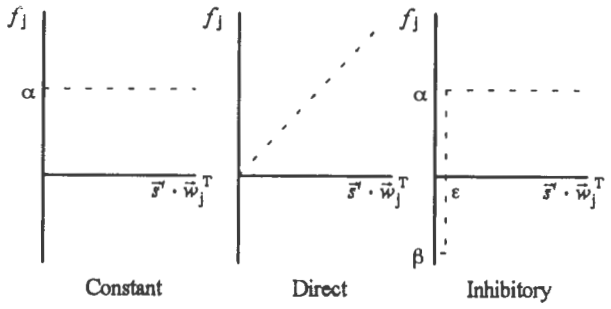


Figure 6: Main feedback functions of ARTSTAR

Each of these feedback functions represents a different level of supervision. The only effect of *constant* feedback is to give more preference to ART2 output nodes which have been committed than to those which haven't. If the feedback is zero, the ART2 network is completely unmodified, and the basic ARTSTAR can be viewed as a simple naming mechanism for output nodes of the ART2 network. *Direct* feedback influences output nodes in proportion to their likelihood of being correct. Thus, over-inclusive nodes are not prevented from occurring, but should be discouraged somewhat. The *inhibitory* feedback represents strong supervision. The ART2 output nodes are essentially prevented from representing more than one class apiece. This implements a supervision similar to those found in existing extensions to ART2 networks, though it is only a single instance of many supervision strategies enabled in ARTSTAR.

3 ARTSTAR Performance

ARTSTAR has been applied to several test domains (see Hussain, 1993). For the purposes of demonstrating its operation this section will describe its application to a data set of handdrawn digits is considered. Three key dimensions of ARTSTAR performance are addressed:

1. the general performance difference between ART2 and ARTSTAR.
2. the effect on performance of the use of the three types of feedback functions used by ARTSTAR, *constant*, *direct*, and *inhibitory*.
3. the effect of integrating multiple, independently-trained modules in ARTSTAR.

Given these dimensions of analysis and the design principles of ARTSTAR, several performance hypotheses are proposed. Firstly, it is expected that

the performance of the ARTSTAR network should be better with *inhibitory* feedback than with *direct* feedback, which in turn should be better than that with *constant* feedback. This is due to the degree of supervision incorporated into the network's training in each case. Secondly, ARTSTAR performance should increase with the number of ART2 modules included. This "redundancy" hypothesis is based on the ARTSTAR design assumption that differently-trained ART2 modules will contain complementary information. Thirdly, the additional redundancy effect that is achieved through the addition of a new module should eventually diminish as the number of modules increases past a certain point. There are two reasons for this expectation. On the one hand, the amount of new complementary information available to ARTSTAR should decrease as more modules are added, while on the other hand, the amount of conflicting information integrated by ARTSTAR should increase with the number of modules. Finally, based upon the results of Tan (1992) and Merz et al. (1992) and upon the design of ARTSTAR, an ARTSTAR network using *inhibitory* feedback should show performance comparable to that of a back-propagation network.

3.1 The Tests

The data used consists of three hundred 16x16 images of the digits 0 through 9 (see Figure 7). About two thirds of the samples were hand drawn by volunteers, and the other third were derived from computer font sets. Each digit is roughly the same size and roughly centred in the image.

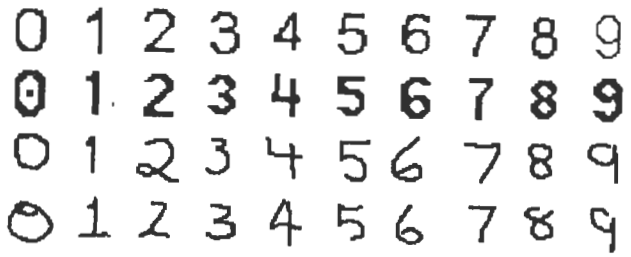


Figure 7: Sample images from digit data set

The complete details of the values of the network parameters used in the tests are provided by Hussain (1993), which also relates the exact algorithm for the version of ART2 used in the modules contained in the ARTSTAR tests. The threshold functions used are those shown in Figure 6, and in all ART2 modules the limiting vigilance value used for all test is 0.97, and the limiting number of ART2 output nodes is 30. In the back-propagation test, a standard back-propagation network is used with 40 hidden nodes and no bias units. The learning rate is set to 0.8 at the beginning of training and linearly decreased to 0.2 during 10,000 training epochs.

The performance of each ARTSTAR is tested in ten trials. During each trial, the complete data set is split randomly into two mutually exclusive halves. Each set contains an equal number of images from each possible class and in each trial, the random split is different. A trial consists of two phases - a training and a testing phase. During the training phase of a trial, each ART2 module of the ARTSTAR is trained separately on the training data, with each module receiving the training data in a different, random, order. A training phase consists of three epochs, and during each epoch, a given ART2 module receives data in a different order than in previous ones. Following the training phase, class names are assigned to each F2 node of each ART2 module of the ARTSTAR based on the class that contributed most to the training of a given node. During the testing phase, data from the testing set is presented once, simultaneously, to all the modules simultaneously. The performance of the ART2 modules is compared with that of the ARTSTAR itself.

3.2 Results

The results of all the tests performed are summarized in Figure 8. In the figure, the performance of ARTSTAR is compared graphically to that of its best ART2 module over the type of feedback function and the number of ART2 modules; the graphs show performance in terms of the percentage of correct classifications. The difference between the dotted lines, representing the performance of the best ART2 module of the ARTSTAR, indicates the effects of the feedback function type (i.e., the supervision effect), while the difference between the solid lines, representing ARTSTAR performance, and their corresponding dotted line indicates the redundancy effect. The graph also includes the average performance of a back-propagation network, over two trials, as well as the chance level of performance for the data set.

In interpreting the results in terms of the hypotheses, several observations can be made. Firstly, the supervision hypothesis can be seen to hold in general, as shown by the difference between the dotted lines in Figure 8. The *inhibitory* feedback function always results in an improved classification performance by ART2 as compared to the *direct* and

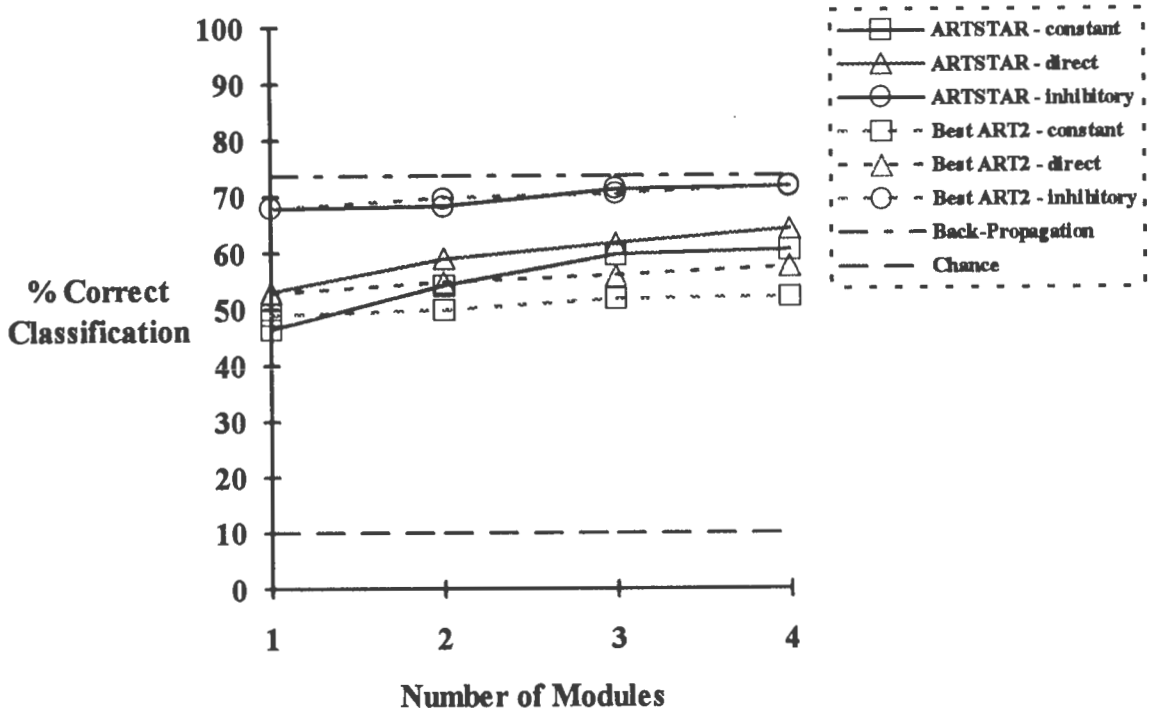


Figure 8: Classification performance of ARTSTAR on digits under several conditions

constant feedback functions, and the *direct* function results in a modest performance improvement over the *constant* function. Further, the performance improvement of ARTSTAR over ART2 using the *inhibitory* function is far larger than any improvement using the *direct* function.

Secondly, the redundancy hypothesis can also be seen to hold in general. The difference between the solid and dotted lines in Figure 8 show that, for the *constant* and *direct* feedback functions, as the number of modules increases the ARTSTAR performs ever better over its best ART2 module; in the figure, the gap between the lines continuously increases. This performance difference is a direct result of the addition of more modules and thus reflects the redundancy effect. Conversely, the redundancy hypothesis does not hold for the *inhibitory* function. This suggests that the strong supervision might eliminate most, if not all, of the inconsistency in the network's categorizations; additional modules contribute no more complementary, just more conflicting information.

Thirdly, there is some support for the redundancy effect diminishment hypothesis. Figure 8 demonstrates nicely that the amount of improvement due to adding another module decreases as the number of modules increases; the solid lines flatten out.

Finally, both ART2 and ARTSTAR performed better than chance but worse than the back-propagation network, but the performance of ARTSTAR with inhibitory feedback closely approached that of back-propagation, as expected.

An additional observation, not directly related to the hypotheses, can be made concerning the results. As the number of modules increases, the performance of the best ART2 module increases as well; the dotted lines show positive slope. This is most probably due to the fact that as the number of modules increases, the chances that a very good ART2 module will be developed increases. Thus, in addition to the benefit due to redundancy, adding more modules, on average, results in a better "best" ART2 module and thus in better ARTSTAR performance.

Overall, the ARTSTAR network demonstrates the desired effects of supervision and redundancy. The best classification ability seems to be obtained from a four-module ARTSTAR using *inhibitory* feedback, and the performance of such an ARTSTAR can approach that of back-propagation. It should be noted, however that the high performance is due almost entirely to the supervision effect - there is no redundancy effect with the *inhibitory* function. ARTSTAR also shows improved performance over normal ART2 when using a *direct* feedback function. In this case, the improvement is due to a combination of the supervision and redundancy effect.

4 Conclusions

The ARTSTAR neural network extends and improves the ART2 network (Carpenter & Grossberg, 1987b) in an attempt to address some of the deficiencies exhibited by ART2 when applied to classification tasks. Specifically, ARTSTAR incorporates two fundamental design principles, the supervision of the ART2 learning process and the integration of multiple ART2 networks. A concise structural extension is proposed based on Grossberg's INSTAR node (Grossberg, 1982). ARTSTAR thereby improves the classification capability of ART2 while preserving the benefits of ART2's self-organization and on-line learning characteristics.

The supervision of ART2 has been examined by several researchers (Grossberg, Carpenter & Reynolds, 1991; Tan, 1992; Merz et al., 1992), all of which have used one common supervision technique. The ARTSTAR network can be made to implement the same technique, but its supervision process is more general than those previously proposed and can take several forms. The integration of multiple ART2 networks, each trained slightly differently on the same set of data, has been briefly considered by Carpenter et al. (1992) and Tan (1992), but only in the context of a post-hoc technique of improving performance. ARTSTAR actually incorporates such redundancy into its structure through its modularity, thereby inherently exhibiting the improved performance. Thus, ARTSTAR is a superset of not just the ART2 network, but also of existing extensions to ART which attempt to improve the performance of ART as a classifier.

The primary application of ARTSTAR considered is the straightforward classification task, on which it has been demonstrated to perform better than the normal ART2 network. However, ARTSTAR also exhibits a much more general functionality because of its modularity, and a variety of other tasks are potential applications of ARTSTAR (e.g., multi-resolution classification, hierarchical classification, data fusion, and invariant pattern recognition).

There are several aspects of ARTSTAR which can be examined in future work. Extensive tests of ARTSTAR properties are required, and the applicability of ARTSTAR to new tasks should be tested. Additional feedback functions should be analyzed to see if ARTSTAR exhibits any novel properties using them. Finally, further tests can be carried out comparing the learning times as well as the performance of ARTSTAR relative to back-propagation. ARTSTAR should have applications not suited for a back-propagation system, and these should be characterized. Finally, modifications and expansions to the ARTSTAR network should be examined.

One extension currently being researched is an extension of the INSTAR feedback to allow greater functionality. Currently, ARTSTAR provides feedback based solely upon the classification vector and does not exploit the differences between the output of the ARTSTAR network and that vector during training. Incorporating feedback based on errors in performance during training should result in an improved ARTSTAR classifier. One possible method of accomplishing such feedback is to assign a separate vigilance factor to each F2 output node of each ART2 module, and to use training performance to adjust the vigilance of the nodes. Thus, for example, a node which made many training errors could be given a high vigilance so that it would become more discriminating.

A second change is to the process of integration of ART2 modules. ARTSTAR currently integrates the outputs of all the modules with an equal emphasis on each module, and ARTSTAR can easily be revised so that different modules may have different levels of importance. For example, each module can be directly connected to the INSTAR layer via a bias node which modifies that module's contribution to the activation of each INSTAR node.

ARTSTAR is a useful, novel neural network architecture. It succeeds in improving the classification capability of ART2, yet is more flexible than existing techniques which also attempt this; it is a modular, supervised network which can be applied to a wide variety of problems, and it exhibits a number of useful properties, though the research presented merely touches upon a few of these. Several future directions for future work on ARTSTAR are possible, including not only further tests, but also several additional design

modifications. ARTSTAR is thus an interesting network which should provide some useful contributions to neural network research.

References

Baruah, A.B. & Welti, R.C. (1991). "Adaptive resonance theory and the classical leader algorithm." Proceedings of the IEEE International Joint Conference on Neural Networks - Seattle (1991), II, p. A-913.

Burke, L.I. (1991). "Clustering characteristics of adaptive resonance." Neural Networks, 4, p. 485-491.

Carpenter, G.A. & Grossberg, S. (1987a). "A massively parallel architecture for a self-organizing neural pattern recognition machine." Computer Vision, Graphics and Image Processing, 37, p. 54-115.

Carpenter, G.A. & Grossberg, S. (1987b). "ART 2: Self-organisation of stable category recognition codes for analog input patterns." Applied Optics, 26, p. 4919-4930.

Carpenter, G.A. & Grossberg, S. (1990). "ART 3: Hierarchical search using chemical transmitters in self-organising pattern recognition architectures." Neural Networks, 3, p. 129-152.

Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.H. & Rosen, D. (1992). "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps." IEEE Transactions on Neural Networks, 3, p. 698-713.

Carpenter, G.A., Grossberg, S. & Reynolds, J. (1991). "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organising neural network." Neural Networks, 4, p. 565-588.

Carpenter, G.A., Grossberg, S., & Rosen, D. (1991a). "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition." Neural Networks, 4, p. 493-504.

Carpenter, G.A., Grossberg, S., & Rosen, D. (1991b). "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system." Neural Networks, 4, p. 759-771.

Duda, R.O. & Hart, P.E. (1973). Pattern Classification and Scene Analysis. New York: Wiley.

Filho, E. & Bisset, D.L. (1990). "Applying the ART1 architecture to a pattern recognition task." Parallel Processing in Neural Systems and Computers, R. Eckmiller, G. Hartmann and G. Hauske (Eds.). Elsevier Science Publishers (North-Holland), p. 343-349.

Gargano, M.L. (1991). "Classifier voting in the neural networks." Proceedings of the IEEE International Joint Conference on Neural Networks - Seattle (1991), I, p. 388-391.

Grossberg, S. (1982). Studies of Mind and Brain. Boston: D. Reidel Publishing.

Hussain, T.S. (1993). "ARTSTAR: A Supervised Modular Adaptive Resonance Network Classifier". M.Sc. Thesis, Department of Computing and Information Science, Queen's University, Sept. 1993.

Merz, C.J., St. Clair, D.C. & Bond, W.E. (1992). "SeMi-supervised Adaptive Resonance Theory (SMART2)." Proceedings of the IEEE International Joint Conference on Neural Networks (1992), III, p. 851-856.

Tan, A-H. (1992). "Adaptive resonance associative map: A hierarchical ART system for fast stable associative learning." Proceedings of the IEEE International Joint Conference on Neural Networks (1992), I, p. 860-865.

Using Redundancy to Improve the Performance of Artificial Neural Networks*

David A. Medler and **Michael R. W. Dawson**
Biological Computation Project
Department of Psychology, University of Alberta
Edmonton, Alberta
CANADA T6G 2E9

Abstract

For Artificial Neural Networks (ANNs) to be effective modelling tools, they must draw upon biological characteristics: One characteristic often overlooked in the design of ANNs is the replication, or redundancy, of processes within the brain. This paper examines the effects of redundancy on the performance of ANNs trained on either a pattern classification task (e.g. parity, encoder) or a function approximation task (e.g. forward kinematics). Results suggest that there is an optimal level of redundancy that increases the likelihood of network convergence while decreasing overall network processing time. ANNs with this level of redundancy consistently perform better than standard ANNs on pattern classification tasks. Furthermore, redundant ANNs trained on the function approximation task are more accurate in terms of overall system error than standard ANNs. These results imply that redundancy may be effectively used to increase the performance of ANNs, both in accuracy and speed.

1 Introduction

The design of Artificial Neural Networks (ANNs) is normally based on engineering principles-- make the system as simple and efficient as possible. From a strict computing science viewpoint, there is little wrong with this approach; however, from a cognitive science viewpoint, this approach is highly suspect. A growing number of scientists are now questioning this performance approach on the grounds that ANNs are moving away from their biological basis and are therefore losing validity as models in cognitive science [e.g. Lewandowsky, 1993; McCloskey, 1991]. To be effective models, ANNs must draw upon biological characteristics, especially those associated with the brain [Dawson and Shamanski, 1994; Dawson, Shamanski, and Medler, 1993]. One biological characteristic often overlooked in the design of ANNs is redundancy: Redundancy is the replication of processes within the brain.

Redundancy in biological systems has been documented since the nineteenth century when it was proposed that functional recovery following unilateral brain lesions was facilitated by replicated processes between the left and right hemispheres of the brain [Gall and Spurzheim, 1810-1819; cited in Almli and Finger, 1992]. Although we now know that the two hemispheres of the brain perform vastly different functions, redundancy within the two hemispheres is still held as a viable theory of functional recovery [Almli and Finger, 1992; Marshall, 1984]. More recently, studies of hydrocephalus patients suggest that normal psychological functioning is still evident with only half the normal brain tissue mass [Berker, Lorber, and Smith, 1983; cited in Glassman, 1987]. This implies that the brain is at least twice as large as is needed for immediate survival, and the extra baggage of the normal brain simply replicates functions already present.

Further neurophysiological evidence for redundancy in biological systems comes from recent studies of animal physiology. Kovac, Davis, Matera, and Croll [1983] found several physiological systems within the nervous system of *Pleurobranchaea californica* that produced essentially the same behavior; when combined, though,these systems greatly enhanced the precision of simple and complex movements. Furthermore,Strehler and Lestienne [1986] examined the firing patterns of neurons within a monkey's visual cortex and found redundant coding in the regularity of triplets of impulses triggered by specific stimuli. Similarly, Swindale [1986] noted that orientation selectivity in the visual cortex is produced by more than one mechanism, and in more than one location. This physiological evidence is complemented by a sizable theoretical literature on the biological relevance of redundancy.

The vast majority of theoretical work on the relevance of biological redundancy has centered on the factors surrounding the evolution of redundancy. Although one of the earlier assumptions concerning redundancy was that it allowed recovery of functioning following brain trauma, it is unlikely that such a rarely survived event like brain damage could exert any natural selection pressure for neural spare capacity [c.f.

* This research was supported by National Science and Engineering Research Council of Canada (NSERC) Research Grant 2038 and NSERC Equipment Grant 138704, awarded to MRWD.

Glassman, 1987]. If we assume that recovery from brain damage is just a convenient side effect of redundancy, we must consider other evolutionary stresses.

Most evolutionary changes involve small, simple changes that allow better adaptation to the surrounding environment; The faster a system can evolve, the greater the chance of survival. Therefore, it would be more advantageous to evolve several small systems that work in parallel to achieve a goal, than a large and highly specialized system [Swindale, 1986]. In fact, such a system is evident in the neural wirings of *Pleurobranchaea californica* [Kovac et al., 1983]. This view is echoed by Calvin [1983] who considered the evolution of neural timing systems required by early hominids for hunting with thrown objects. At short distances, a single timing neuron is sufficient to allow the proper release time needed for a hit; however, the timing precision required for a strike increases eight-fold with a mere doubling of throwing distance. Therefore, the brain combined the efforts of several timing neurons to increase the precision above that of any single neuron. Consequently, redundancy may have evolved not because brain damage was anticipated, but because it was easier to replicate, and thus improve, what was already present than to develop a single system beyond reproach.

A slightly different theoretical approach to redundancy comes from Jacobson [1976] and Glassman [1987]. Jacobson [1976] considered the connections between neurons involved in a memory trace based on Hebb's model of the cortex, and defined redundancy as "to mean the condition that pairs of cells joined along one effective pathway are joined again along another" (p. 150). Using mathematical calculations and assuming initial random connections between neurons, Jacobson showed that redundancy is an inevitable consequence of the connections within the cortex. Glassman [1987], on the other hand, looked at the probability of overall system failure for any large structure. With no redundancy, failure within a finite time is guaranteed; therefore, if the brain had no redundancy, the chances of it functioning for any significant amount of time are slim.

We have seen that redundancy is a viable, if not necessary, biological property, but can it be effectively implemented in ANNs? Recently, there has been a flurry of connectionist research on using multiple nets to solve problems (e.g. committees, agent teams, stacked generalization, model averaging, error correcting codes). As an example, Baxt's [1992] medical diagnosis network is based on two networks working in parallel: one network is trained to classify positive examples of myocardial infarction, and the other network is trained to classify negative examples. By combining their outputs, Baxt has produced a network that has a hit rate of 97.50% and a false alarm rate of 1.63%, which is considerably better than any human. Although this network is not strictly redundant (i.e. each network is trained on different pattern sets), it gives some indication of the increased accuracy associated with redundancy.

Another form of computational redundancy widely studied today centers around committee machines. Committee machines are based on the principle of using several computers (or networks) at once to solve the same problem. The training algorithm for such machines is rather unique [see Schapire, 1990]: Briefly, the first machine is trained on one pattern set, and then subsequent machines are trained on new pattern sets composed of equal amounts of correctly and incorrectly classified patterns that have been passed through previous machines. Once trained, however, there is little agreement as to the best way of combining the outputs of the different committee machines. Several alternatives have been suggested, from a simple "winner-take-all" or "voting" strategy, to summing the outputs, to calculating the mean output, to implementing a separate network to choose which machine's output is the most appropriate. Regardless of the combining strategy used, the committee machines invariably perform better than single networks alone.

The above research examples, however, have centered on improving the performance of ANNs from an engineering perspective solely. For example, it is not clear that any of the output strategies listed above, or even the training algorithms used for committee machines, are biologically plausible.

Constraints borrowed from biological networks, nevertheless, may have positive effects on the performance of ANNs as illustrated by Izui and Pentland's [1990] research on redundant networks. Using biological redundancy as a model, they mathematically analyzed the functional effects of one of neuronal duplication. Their mathematical calculations predict that redundant networks are more accurate, faster, and stable than standard networks. These predictions were confirmed by both a feedforward neural network trained on the XOR problem, and a feedback neural network trained on the travelling salesman problem. From these results, Izui and Pentland claim that the "highly redundant nature of biological systems is *computationally* important and not merely a side-effect of limited neuronal transmission speed and lifetime" (p. 237). Although Izui and Pentland's research has laid the mathematical foundations of network redundancy, their practical work requires expansion before redundancy is accepted as a useful addition in ANN design. For example, larger problem sets should be considered as well as the applicability of redundancy to different network architectures.

Three different questions are addressed by this current research. First, is there an optimal level of redundancy that improves ANN convergence without increasing the amount of processing required? Second, how do redundant ANNs fare on different classes of problems (e.g. pattern classification versus function approximation)? Third, can redundancy be effectively used with different types of network processing units (e.g. monotonic versus non-monotonic)? It is hypothesized that when the optimal level of redundancy is used, redundant networks will have better convergence on pattern classification problems than standard networks. Furthermore, redundant networks should be more accurate than standard networks on function approximation tasks. Finally, redundancy should be effective with both types of processing units.

2 Experiment 1: Levels of Redundancy

Adding redundancy to a network creates an interesting question from an engineering viewpoint: Are the added hardware requirements of adding extra processing nodes more than compensated for by an increase in performance? In other words, can we trade simplicity for efficiency? Theoretically, redundancy will not increase the overall network processing time as all redundant layers are working in parallel; however, the number of processing steps required will increase proportional to the number of redundant layers. Therefore, we can compare the performance of redundant ANNs to either the total processing time of a standard ANN, or to *1/N* processing sweeps of a standard ANN, where *N* is equal to the level of redundancy.

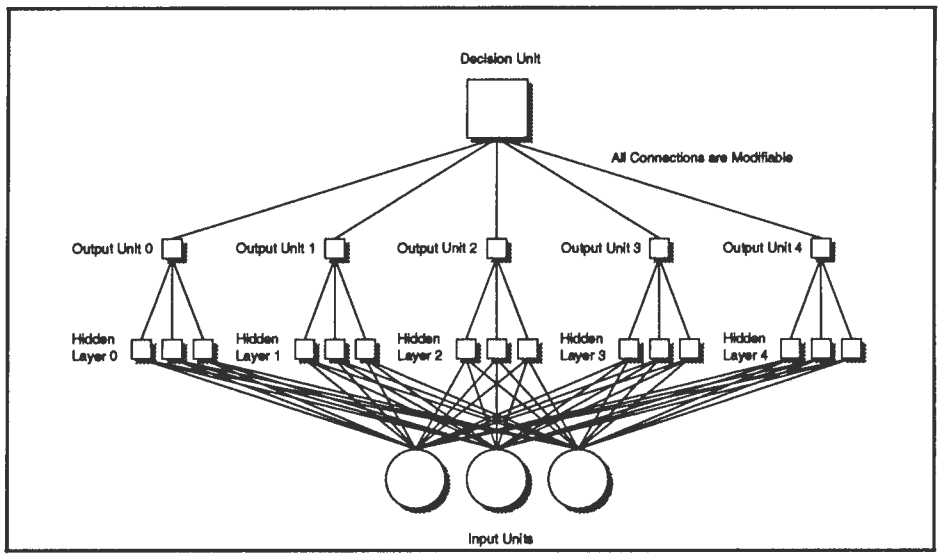


Figure 1. 3-Parity Network Structure with 5 Levels of Redundancy

The problem that now exists is to find the optimal level of redundancy where the increase in hardware requirements is offset by an equal or greater increase in performance. It has been calculated that the brain has at least two, and as many as seven, different layers of redundancy [Glassman, 1987]. Therefore, to assess the optimal level of redundancy for an ANN, the performance of a standard ANN trained on varying levels of a difficult pattern classification task (i.e. 2- to 8-parity) will be compared to the performance of ANNs with from two to eight levels of redundancy.

2.1 Method

2.1.1 Network Architecture

The *standard* network architecture consisted of an input layer, a hidden unit layer, and an output layer: The number of input units and hidden units was equivalent to the size of the parity problem (e.g. ANNs trained on 3-parity had 3 input units, 3 hidden units, and 1 output unit). Connection weights were randomly assigned from a rectangular distribution over the range [-1, +1], and processing unit biases were initialized to 0. All biases and connections within the network were modifiable.

The *redundant* network architecture was created by replicating the hidden unit layer and the output unit layer a set number of times. Each of the replicated output units was then connected to a *Decision Unit*, which acts as the redundant network's output unit. All connections leading into the Decision Unit are modifiable; therefore, the Decision Unit's response is a weighted sum of the replicated output units. Figure 1 shows the redundant network structure for an ANN with five levels of redundancy trained on a 3-parity problem. It should be noted that, as opposed to a three-layer network, no connections exist directly between each of replicated networks. Furthermore, each of the replicated networks was initialized independent of the others. Connection weights were randomly distributed over the range [-5, +5] to introduce more variability in the network, and all unit biases were set to 0. Seven different levels of redundancy were tested: 2, 3, 4, 5, 6, 7, and 8.

2.1.2 Training Stimuli

Parity is a linearly inseparable pattern classification task defined by the number of active input units: If the number of 1's in the input pattern is odd, then the output is 1, otherwise it is 0 [Minsky and Papert, 1969]. ANNs were trained on 2-, 3-4-, 5-, 6-, 7-, or 8-parity problems which had training set sizes of 2, 8, 16, 32, 64, 128, and 256 respectively. Each training set had equal numbers of positive and negative examples of parity.

2.1.3 Training Procedure

The network was trained with the backpropagation algorithm using the *generalized delta rule* (GDR) [see Rumelhart, Hinton, & Williams, 1986]. Backpropagation is described as a steepest descent optimization algorithm for traversing the surface of a weight space whose height measures error. Descent through the weight space is aided by two parameters: momentum (α) and rate-of-learning (η). Momentum is a technique for escaping local minima within the weight space by averaging the weight change for one item with the weight change for the previous item. The rate-of-learning parameter is used to dictate how large a "step" to make when traversing the weight space. For all parity problems, $\alpha = 0.9$, and $\eta = 0.1$.

To train the network, a pattern was randomly sampled--without replacement-- from the pattern set and presented to

the network. The network's actual output was then compared to the desired output, and connection weights and unit biases were modified according to the above algorithm. If the absolute difference between the actual output and desired output was less than 0.05 then a "hit" was recorded. One *sweep* of the network was completed once all patterns were presented to the network. Training of the network continued either until the maximum number of sweeps was completed (30,000) or until each pattern in a sweep produced a hit.

2.2 Results and Discussion

The performance of standard ANNs versus redundant ANNs was compared using three measures: probability of convergence, sweeps to convergence, and total processing steps to convergence. As can be seen from Table 1, the standard networks failed to classify 5-parity and above within 30,000 processing sweeps, whereas the redundant networks found a solution from 30% to 100% of the time on all parity problems. It should be noted that as the level of redundancy increased, so did the probability of convergence; however, only networks with 5 levels of redundancy converged 100% of the time on all sizes of parity problems.

Table 1. Median Processing Time and Steps to Convergence as a Function of Parity and Redundancy

	Level of Redundancy							
Problem	0	2	3	4	5	6	7	8
2 Parity								
Time	2551	932	841	819	663	641	660	491
Steps	2551	1864	2522	3276	3315	3846	4620	3928
n	10	10	10	10	10	10	10	10
3 Parity								
Time	1214	2235	973	630	587	688	447	526
Steps	1214	4470	2919	2520	2935	4128	3129	4208
n	10	9	10	10	10	9	10	10
4 Parity								
Time	14126	1732	1051	709	608	680	550	543
Steps	14126	3464	3153	2836	3040	4080	3850	4344
n	4	9	10	10	10	10	10	10
5 Parity								
Time	----	1548	997	831	634	748	747	693
Steps	----	3096	2991	3324	3170	4488	5229	5544
n	0	6	10	10	10	10	10	10
6 Parity								
Time	----	1846	1492	932	738	821	720	470
Steps	----	3692	4476	3728	3690	4926	5040	3760
n	0	8	10	10	10	10	10	10
7 Parity								
Time	----	1826	1663	874	724	579	716	664
Steps	----	3652	4989	3496	3620	3474	5012	5312
n	0	3	7	9	10	9	9	10
8 Parity								
Time	----	1811	1835	884	758	579	692	566
Steps	----	3622	5505	3536	3790	3474	4844	4528
n	0	4	9	8	10	7	10	9

Note. Maximum number of sweeps = 30000; n = number of converged networks out of 10.

Similarly, when sweeps to convergence is considered, there is a general decrease in sweeps with an increase in redundancy. Again, though, this decrease begins to asymptote around 5 levels of redundancy, which suggests a floor effect. A slightly different function appears with the total processing steps to convergence, calculated by multiplying the number of sweeps by the level of redundancy. This time, there is a slight quadratic function with its lowest points being around 4, 5, or 6 levels of redundancy depending on the problem difficulty. When the number of processing steps is averaged across all parity problems, networks with 4 levels of redundancy perform best, followed by networks with 5 and then 6 levels of redundancy.

Taking all of the above results into consideration, it appears that networks give best overall performance with 5 levels of redundancy for this type of linearly inseparable pattern classification problem. It does appear, however, that networks trained on easier problems (e.g. 2- and 3-parity) do not benefit, and may actually suffer, from redundancy. Nevertheless, these results show that the added hardware requirements of redundancy are more than compensated for by an impressive increase in performance. Although these results only generalize to the parity problem, all further experiments within this paper will adopt a redundancy level of 5.

3 Experiment 2: Pattern Classification

Experiment 1 has shown that redundancy improves the performance of ANNs trained on the parity problem; however, we do not know if theses results will generalize to other types of pattern classification problems, or if redundancy only improves the performance of networks with monotonic activation functions. Experiment 2 looked at the effects of redundancy on the number of network sweeps required for ANNs to learn two different types of difficult pattern classification tasks: 3-, 5-, 7-, and 9-parity, and 424-, 838-, and 16416-encoder. Furthermore, the effect of redundancy on the "standard" ANN architecture [e.g. Rumelhart, Hinton, and Williams, 1986] was compared to the effect of redundancy on a different architecture [Dawson and Schopflocher, 1992].

Originally, the backpropagation algorithm was developed under the assumption that the activation function for processing units had to be differentiable and monotonic [Rumelhart, Hinton, and Williams, 1986]; Such processing units are termed *integration devices* by Ballard [1986]. Recently, however, Dawson and Schopflocher [1992] have shown that processing units with a non-monotonic activation function-- called *value units* [Ballard, 1986]-- can learn pattern classification problems much faster than integration devices. Consequently, it is hypothesized that redundant ANNs will converge faster than standard ANNs, and that value unit ANNs will perform better then integration device ANNs. Therefore, the best performance is expected from the redundant value unit network.

3.1 Method

3.1.1 Network Architecture

The standard networks used for the parity classification problems were two-layer networks with one output unit. The number of input units and hidden units, however, were equivalent to the size of problem being solved: namely 3, 5, 7, or 9 units for the respective parity problem. Networks for the encoder problems had 4, 8, or 16 input and output units , and 2, 3, or 4 hidden units as dictated by the size of problem. Connection weights were randomized from a rectangular distribution over the range [-5, +5] for integration device networks using a sigmoidal activation function, or [-1, +1] for value unit networks using a Gaussian activation function. Processing unit biases, regardless of activation function, were initialized to zero. The redundant networks were created as described in Experiment 1.

3.1.2 Training Stimuli

The 3-, 5-, and 7-parity training sets used in this experiment were the same as in Experiment 1: A 9-parity set with 512 training patterns was also included. The training sets for the encoder problems consisted of 4, 8, or 16 orthogonal input patterns composed of a single 1 and filler 0's (e.g. 1 0 0 0, 0 1 0 0, 0 0 1 0, 0 0 0 1). The output patterns and input patterns were equivalent.

3.1.3 Training Procedure

The networks were trained with the backpropagation algorithm using either the GDR for processing units with a sigmoidal activation function [Rumelhart, Hinton, and Williams, 1986], or a modification of the GDR for processing units with a Gaussian activation function [Dawson and Schopflocher, 1992]. For the integration device networks, the parameters were set at $\alpha = 0.9$ and $\eta = 0.1$. Parameters for the value unit networks were $\alpha = 0$ and $\eta = 0.05$.

Training of the ANNs proceeded as described in Experiment 1. A hit was recorded if the actual output was 0.95 or higher when a 1 was desired, or 0.05 or lower when a 0 was desired, and the maximum number of sweeps before failure was set at 30,000. Second, the maximum number of sweeps allowed was held constant at 30000 for all networks. Training continued until all patterns in the set were learned or until the maximum number of sweeps was reached. As the initial random assignment of connection weights introduces variability in learning, each of the four different networks (standard vs. redundant, integration device vs. value unit) was initialized and trained 10 times. The minimum, median, and maximum number of sweeps to convergence, and the number of ANNs reaching convergence were recorded.

3.2 Results and Discussion

Table 2 shows the minimum, median, and maximum number of sweeps required to reach convergence and the total number of networks out of 10 to reach convergence for the 3-, 5-, 7- and 9-parity problems. The redundant networks solved the problems in fewer sweeps than the standard networks. Furthermore, the redundant networks converged on a solution 100% of the time while the standard networks often failed to converge on a solution even after 30000 sweeps. When equalized for the total number of network processing steps, the redundant networks only outperform the standard networks as the problem difficulty increases. Finally, it should be noted that the value unit networks converged much faster than the integration device networks for both the standard network architecture and the redundant network architecture. Also, the standard value unit networks converged on a solution more often than the standard integration device networks, particularly with the more difficult problems.

Table 2. Parity Problem: Sweeps to Convergence as a Function of Network Architecture and Processing Unit Type

	Network Architecture							
	Standard				Redundant			
Sweeps	3	5	7[a]	9	3	5	7	9
Integration Device ANNs								
Minimum	661	5817	-----	-----	261	576	353	397
Median	2599	6943	-----	-----	623	717	1237	1378
Maximum	24850	8068	-----	-----	1017	1047	2853	3695
n	8	2	0	0	10	10	10	10
Value Unit ANNs								
Minimum	49	213	1042	3052	37	34	71	360
Median	81	258	2744	5848	67	84	97	918
Maximum	200	1015	28322	14310	156	130	302	1449
n	10	9	8	6	10	10	10	10

Note. Maximum number of sweeps = 30000; n = number of converged networks out of 10.
a. Due to the difficulty of the 7- and 9-parity problems, different values of η were tried. Value units learned best with $\eta = 0.01$, whereas integration devices failed to learn at all values of η.

Similar results are obtained when we look at the different encoder problems. Redundancy decreases the amount of processing time and increases the likelihood of convergence for both integration device networks and value unit networks (see Table 3). When the number of processing steps are taken into consideration, however, redundancy does not help the integration device networks; On the other hand, value unit networks profit greatly from redundancy. In fact, the worst performance of the redundant value unit networks is better than the best performance of the redundant integration device

networks.

Table 3. Encoder Problem: Sweeps to Convergence as a Function of Network Architecture and Processing Unit Type

	Network Architecture					
	Standard			Redundant		
Sweeps	424	838	16416	424	838	16416
Integration Device ANNs						
Minimum	1200	2232	5143	353	588	827
Median	1399	3377	7177	514	787	2265
Maximum	4024	10399	22636	879	3060	8884
n	10	9	6	10	10	10
Value Unit ANNs						
Minimum	360	482	772	52	63	74
Median	545	775	944	77	96	95
Maximum	975	1088	1921	145	154	127
n	10	10	10	10	10	10

Note. Maximum number of sweeps = 30000; n = number of converged networks out of 10.

In conclusion, convergence on pattern classification problems is much faster with redundant ANNs than with standard ANNs. Furthermore, redundant ANNs converge on a solution 100% of the time regardless of problem type or size, whereas the standard ANNs often failed to reach convergence. Also, standard value unit networks converged more often than standard integration device networks When the networks are equalized for total number of processing steps, the redundant integration device networks only outperform the standard networks on the more difficult problems, whereas redundancy always improves the performance of value unit networks. These findings support Dawson and Schopflocher's (1992), conclusions the value unit networks outperform integration device networks on linearly inseparable pattern classification problems.

4 Experiment 3: Function Approximation

Experiment 1 and Experiment 2 have conclusively shown that redundancy can improve the performance of ANNs trained on difficult pattern classification tasks. The last question to be addressed is whether or not redundancy will improve the performance of ANNs trained on a function approximation task. With function approximation, however, the number of sweeps to reach convergence is no longer an appropriate measure of network performance; therefore, performance will be evaluated via overall network error.

The function approximation task chosen is based on Churchland's [1992] crablike creature which is programmed to reach to a point in space. Our simulated robotic arm, however, will use a neural network to essentially learn forward kinematics. Previous research [Calvin, 1983; Kovac et al., 1983] has suggested that redundancy in biological systems increases the accuracy of simple movements. Therefore, it is hypothesized that redundant ANNs trained on a function approximation task will have less error in responding than standard ANNs.

4.1 Method

4.1.1 Network Architecture

The standard network used was a two-layer network with two input units, two hidden units, and two output units. Connection weights were randomly assigned from a rectangular distribution over the range [-5,+5], and processing unit biases were initialized to zero. The redundant network was created as before with the exception that there were now two Decision Units to correspond with each replicated network's two output units.

4.1.2 Training Stimuli

A schematic diagram of the simulated robot and the problem space is shown in Figure 2. An object was placed randomly in front of the simulated robot: If the object fell within an unreachable area (i.e.grey area in Figure 2) then a new position was randomly chosen. Inputs to the network were the two angles (μ, ω) that the eyes subtended when converged on the object, while the desired network outputs were the angles (ρ, ϕ) that the shoulder joint and elbow joint made

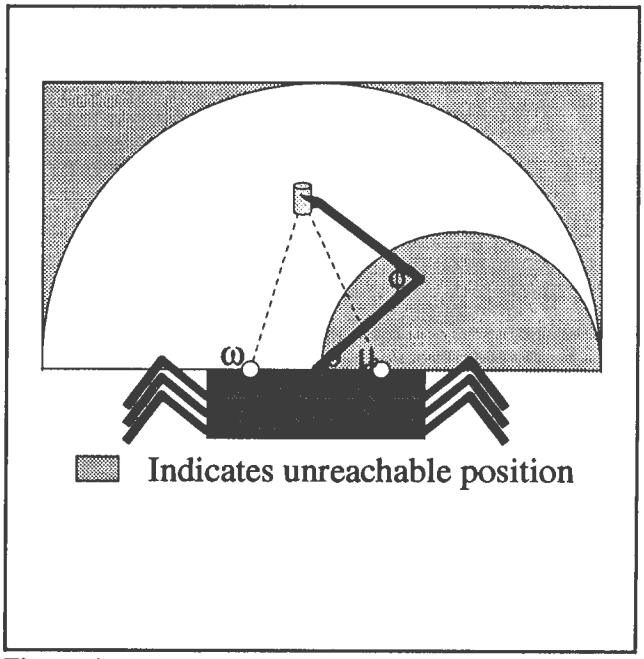


Figure 2. Definition of Problem Space for Simulated Robotic Arm.

in order for the arm to contact the object. All angles were normalized to fall within the range of 0 to 1. The inputs could be considered two-dimensional sensory-state space coordinates, and the outputs would then be considered as separate two-dimensional motor-state space coordinates. The network, therefore, learns the appropriate mapping between the two state spaces [see also Zipser and Anderson, 1988]. As the mapping of the two state spaces forms a continuous function, there are an infinite number of input/output pairs; however, practicality limited the training set to 50 randomly chosen pairs.

4.1.3 Training Procedure

Training of the networks proceeded as in Experiment 1 with some minor changes. First, a "hit" was defined when the absolute error between the desired output and the actual output was less than 0.001. Second, the networks were trained for 50,000 sweeps. Momentum and rate of learning for the simulated robotic arm were $\alpha = 0.9$, and $\eta = 0.1$.

To assess the network's ability to learn the function approximation problem, maximum network sweeps were increased from 100 to 50000 in $\log_{10}$ steps. Total network sum of squared errors (SSE)-- as measured by the difference between desired and actual network response-- as well as the SSE for each individual output (ρ, ϕ), were recorded at each maximum sweep step. As the initial randomness of connection weight assignment produces great variability in network learning, five different networks were trained for both the standard network architecture and the redundant network architecture.

4.2 Results and Discussion

The total SSE range and median for the simulated robotic arm for both the standard and redundant networks are shown in Figure 3. As can be seen, median SSE decreases faster for the redundant networks than for the standard networks. In fact, the average median SSE for the redundant network is significantly less than the average median SSE for the standard network ($\bar{x}$ = 0.355, 0.892 respectively; $F(1,44) = 23.899$, $p < .001$). Furthermore, the range of the total SSE is significantly less for the redundant network than for the standard network ($F(1,44) = 23.90$, $p < .001$). This holds true for both the elbow joint ϕ ($F(1,44) = 15.95$, $p < .001$) and the shoulder joint ρ ($F(1,44) = 6.91$, $p < .05$).

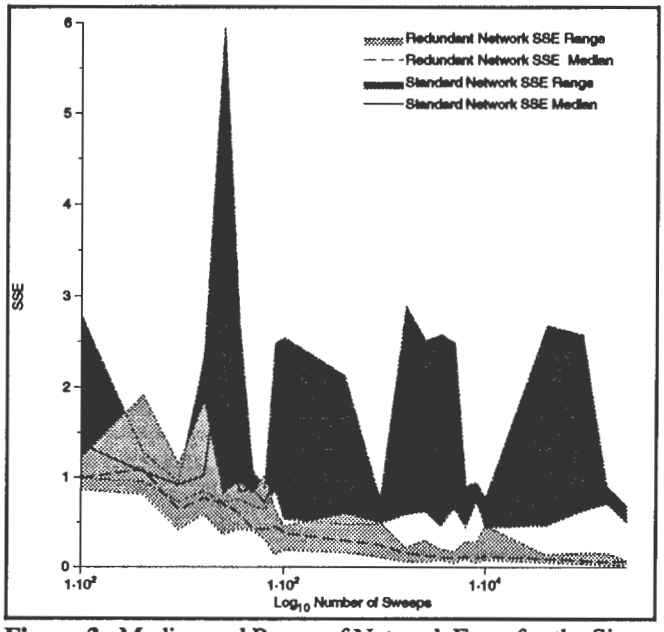


Figure 3. Median and Range of Network Error for the Simulated Robotic Arm

As hypothesized, redundant networks are significantly more accurate than standard networks on function approximation problems. Not only is the median SSE less for the redundant ANNs than for the standard ANNs, but the SSE range is significantly less for redundant networks as well. This means that the responding of the redundant ANNs is much less variable-- or more stable-- than the responding of the standard ANNs. Also, the advantage for the redundant ANNs increases with the number of sweeps completed when total network processing time is considered. This advantage even holds at the higher end of the sweep scale when the redundant and standard networks are equalized for total number of processing steps.

5 General Discussion

The results from both the PC problem and the FA problem confirm Izui and Pentland's [1990] mathematical analysis of redundant networks: Redundancy produces faster convergence, more accurate results, and more stable networks than comparable standard networks. In terms of the relevance of redundancy to the performance of ANNs in general, redundant networks should be considered as a viable alternative to standard networks. The initial cost of the extra hardware associated with redundancy is far out-weighed by the savings in training, accuracy in responding, and network stability produced by redundant processes.

Our results have shown that there is another alternative to the combining algorithms used by committee machines (e.g. mean response, winner-take-all, median response, etc.). The modifiable connections from the individual output units to the decision unit allows the network to train itself. As opposed to taking the mean output response of individual networks, which gives equal weighting to all networks, the amount of contribution is weighted according to how well the individual networks classify the problem. Furthermore, all individual networks contribute to the final result, unlike winner-take-all or median response methods. Consequently, the modifiable connections of the decision unit have proven to be a functional alternative to those methods conventionally used while preserving some semblance of biological systems.

Some evolutionary theories are supported by the performance of the redundant ANN. For example, the increased precision of the redundant network over the standard network on the FA problem

lends credence to Calvin's [1983] hypothesis about redundancy evolving to increase the precision of a system. In fact, as the upper limit of network sweeps increases, the worst redundant network is more precise than the best standard network. Also, the number of sweeps to train both the FA network and the PC network suggests that it is easier to evolve several crude mechanisms working in parallel than one extremely effective mechanism [Swindale, 1986].

Further research will consider the possibility of loss of redundancy accounting for loss of functioning in patients with debilitating diseases. As stated earlier, it is widely held that redundancy in the brain allows for functional recovery after brain damage [Almli and Finger, 1992]. It follows that loss of redundancy may cause loss of functioning. Modelling redundancy via computer simulation has a distinct advantage over biological models, in that precise ablations can be performed on artificial neural networks. Therefore, predications can be made about the performance of biological functioning when redundancy is compromised.

For example, the results of Experiment 3 show that the variability in making a response is much greater for a non-redundant network than a fully redundant network; therefore, as redundancy decreases, variability in responding should increase. Monitoring the variability changes should be an effective tool for estimating how much damage the system has suffered, and should even predict when terminal drop will occur. A practical application of this theory has already been hinted at by Patterson, Foster, and Heron, who conclude that for assessing damage by Multiple Sclerosis, "variability is a more sensitive indicator of visual pathway damage than the usual measure of mean" (1980, p.143). By attempting to model this increase in variability, we may be in a better position to understand the underlying damage associated with such diseases as Multiple Sclerosis and Alzheimers.

References

Almli, C. R., & Finger, S. (1992). Brain injury and recovery of function:Theories and mechanisms of functional reorganization. *Journal of Head Trauma Rehabilitation, 7*, 70-77.

Ballard, D. (1986). Cortical structures and parallel processing: Structure and function. *TheBehavioral and Brain Sciences, 9*, 67-120.

Baxt, W. G. (1992). Improving the accuracy of an artificial neural network using multipledifferently trained networks. *Neural Computation, 4*, 772-780.

Calvin, W. H. (1983). A stone's throw and its launch window: Timing precision and its implications for language and hominid brains. *Journal of Theoretical Biology, 104*, 121-135.

Churchland, P. M. (1992). *A neurocomputational perspective: The nature of mind and the structure of science*. Cambridge, MA: MIT Press.

Dawson, M. R. W., & Schopflocher, D. P. (1992). Modifying the Generalized Delta Rule to train networks of non-monotonic processors for pattern classification. *Connection Science, 4*, 19-31.

Dawson,M.R.W., & Shamanski, K. S. (1994). Connectionism, confusion, and cognitive science. *Journal of Intelligent Systems, In press.*

Dawson, M.R.W., Shamanski, K. S., & Medler, D. A. (1993). From connectionism to cognitive science. In L. Goldfarb (Ed.) *Proceedings of the Fifth University of New Brunswick Artificial Intelligence Symposium.* Fredericton, NB: UNB Press.

Glassman, R. B. (1987). An hypothesis about redundancy and reliability in the brains of higher species: Analogies with genes, internal organs, and engineering systems. *Neuroscience & Biobehavioral Reviews, 11*, 275-285.

Izui, Y., & Pentland, A. (1990). Analysis of neural networks with redundancy. *Neural Computation, 2*, 226-238.

Jacobson, J. Z. (1976). Relative possibilities of loops and redundant connections in neural nets. *Journal of Mathematical Psychology, 13*, 148-162.

Kovac, M.P., Davis, W.J., Matera, E.M., & Croll, R.P. (1983). Organization of synaptic inputs to paracerebral feeding command interneurons of *Pleurobranchaea californica.* I. Excitatory inputs. *Journal of Neurophysiology, 49*, 1517-1538.

Lewandowsky, S. (1993). The rewards and hazards of computer simulations. *Psychological Science, 4*, 236-243.

McCloskey, M. (1991). Networks and theories: The place of connectionism in cognitive science. *Psychological Science, 2*, 387-395.

Marshall, J. F. (1984). Brain function: neural adaptations and recovery from injury. *Annual Review of Psychology, 35*, 277-308.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons*. Cambridge, MA: MIT Press.

Patterson, V. H., Foster, D. H., & Heron, J. R. (1980). Variability of visual threshold in Multiple Sclerosis: Effect of background luminance on frequency of seeing. *Brain: A Journal of Neurology, 103*, 139-147.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D E. Rumelhart, J. L. McClelland, and the PDP Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol 1.* (pp 318-362). Cambridge, MA.: MIT Press.

Schapire, R. (1990). The strength of weak learnability. *Machine Learning, 5*, 197-227.

Strehler, B. L., & Lestienne, R. (1986). Evidence on precise time-coded symbols and memory of patterns in monkey cortical neuronal spike trains. *Proceedings of the National Academy of Sciences of the United States of America, 83*, 9812-9816.

Swindale, N. V. (1986). Parallel channels and redundant mechanisms in visual cortex. *Nature, 322*, 775-776.

Zipser, D., & Andersen, R. A. (1988). A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature, 331*, 679-684.

A Diagnosis Method for Multiple Failures in a Nonlinear and Dynamic Process

Takashi Washio, Ph.D.
Safety Engineering Department
Mitsubishi Research Institute, Inc.
1-8-1, Shimomeguro, Meguro-ku,
Tokyo 153, Japan
washio@mri.co.jp

Masatake Sakuma and Masaharu Kitamura, Ph.D.
Nuclear Engineering Department
Tohoku University
Aramaki-Aza-Aoba, Aoba-ku,
Sendai, 980, Japan
g21447@cctu.cc.tohoku.ac.jp

Abstract

Many practical applications of diagnosis require the reliable identification of multiple faults of components and sensors in quantitative measures. However, the state of the art is considered to be still insufficient to meet these severe requirements especially for nonlinear and dynamic systems.

This research proposes a method to achieve these practical requirements using the frameworks of optimum constraints, minimal conflicts based diagnosis, and causal ordering of physical systems. First, the detection of faulty behaviors of an objective system is performed based on the quantitative consistency checking between observations and the optimum constraints called as "minimal over-constraints". Second, once if some inconsistencies are detected, the minimal conflicts based diagnosis derives the candidates of faulty mechanisms. Third, the anomalous quantities directly disturbed by the faulty mechanisms are identified systematically based on causal ordering. Furthermore, the quantitative deviations of these quantities are evaluated using the minimal over-constraints.

The practicality of the proposed method is demonstrated through an example to diagnose an electric water heater.

1 Introduction

Many practical applications of diagnosis require the reliable and real-time identification of multiple failures of components and sensors in quantitative measures. The approaches of model-based constrains without using any knowledge of failure modes have a significant advantage to diagnose any unexpected failures [de Kleer and Williams, 1987; Hamilton, 1988; Torasso and Console, 1989]. However, the state of the art might be still insufficient to meet the severe requirements in the practical applications. This research proposes a method to achieve the practical requirements by using the frameworks of optimum constraints, minimal conflicts based diagnoses, and causal ordering under the following premise.

Premise 1 : The objective system for diagnosis is represented by physical or functional constraints which may be nonlinear and dynamic.

Figure 1 shows the entire diagnostic procedure. In the step (1), the detection of faulty behaviors of an objective system is performed first based on the quantitative consistency checking between (A) the observed information and (B) a certain type of optimum constraints of the normal system. Once if some inconsistencies are detected, the minimal conflicts based diagnoses [Reiter, 1987; de Kleer and Williams, 1987; de Kleer et al., 1992] are applied. The constraints in the knowledge (B) are called as "minimal over-constrained subsets" [Washio and Kitamura, 1992; Washio et al., 1993]. They are established to have the maximum resolution of the consistency checking to identify faulty mechanisms under the following second and third premises which are widely common in practical process systems.

Premise 2 : The arrangement of installed sensors is initially given and fixed during the operation of the system, and has some redundancies to identify the system states.

Premise 3 : Any quantitative expectations of dynamic system behaviors are not available without using the system description and the sensors' observations.

In the step (2), the anomalous quantities directly disturbed by the faulty mechanisms are identified systematically based on (C) the causality information among the quantities. Furthermore, the quantitative deviations of these anomalous quantities are evaluated based on the knowledge (B). The knowledge (C) represents the orders of the determination of quantities [Simon, 1977; Iwasaki and Simon, 1986; Iwasaki, 1989]. An extended theory of "causal ordering" proposed in our previous work [Washio, 1989; Washio and Kitamura, 1992; Washio et al., 1993] is adopted in this research.

The knowledge (B) and (C) can be prepared systematically by off-line processing in advance, and the on-line and real-time processing for the consistency checking and the deviation evaluation does not require the search of any new constraints. The diagnosis speed may be applicable to the real-time use due to this optimized feature and the recent progress of computer hardware.

The performance of the proposed method is exemplified through the diagnosis of an electric water heater depicted in fig.2. A resistant wire is electrically shielded from the surroundings, and its resistance has a nonlinear feedback effect from water temperature. The water is assumed to be always mixed well to avoid the spatial fluctuation of its temperature.

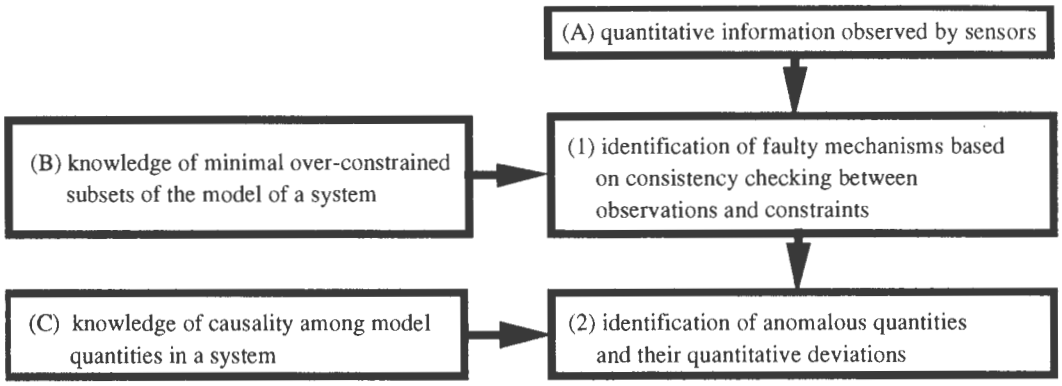


Figure1 Entire procedure of a proposed diagnosis method..

Some major physical quantities are measured by given sensors. The physical model of this process system is expressed as follows.

$I = I_p$ (1), $I_g = I$ (2), $V = IR$ (3), $F_h = VI$ (4), $H = \int_{-\infty}^{t} F_h\, dt$ (5), $T = H/(cM)$ (6), $R = r + k(T - t_c)^2$ (7), $I_p^* = I_p$ (8), $I_g^* = I_g$ (9), $V^* = V$ (10), $M^* = M$ (11), $T^* = T$ (12)
(* indicates the measurements of sensors.)
I_p, I, I_g: electric currents of a power supply, a resistant wire and the ground; R, V, F_h: resistance, voltage and heat generation rate of the resistant wire; H, M, T: contained heat, mass and temperature of water; c=4.2J/cal: specific heat coefficient of water, t_c=300°K : standard temperature; $r = 100\,\Omega$, $k = 5\,\Omega/°K^2$: resistance at t_c and temperature coefficient of the wire.

Each sensor model should be explicitly introduced into the system model to enable a uniform diagnosis of sensor failures and component failures. This example has the following characteristics to demonstrate the generality of our proposing method.

(1) nonlinear mechanisms (2) feedback loops
(3) dynamic behaviors (4) sensor models

2 Minimal Over-Constrained Subsets and Failure Identification

2.1 Minimal Over-Constrained Subsets

First, Reiter's framework of the system definition is introduced for general discussion [Reiter 87]. A system is a triple (SD, COMPS, OBS) where they stand for the system description, the system components and a set of observations, respectively. Each constraint c in the system model belongs to the SD, i.e., $c \in SD$, because they are used to derive the normal behaviors of the system. Furthermore, the constraints take a unique position in our framework that each constraint c provides a basic granule of anomaly, playing the role of a system component to be diagnosed, i.e., $c \in COMPS$. Accordingly, eq.(1)-(12) belong to both of the SD and the COMPS in our case.

The constraints in the SD are always over-constrained by the information in the OBS under the aforementioned premise 2. Especially, the over-constraints with one degree have the minimal sizes in the sense of number of their elements involved. They are expected to provide the maximum resolution in the consistency checking under the premise 3, because any extra information other than the SD and the OBS is not applicable to the consistency checking. Under this circumstance, the following definitions are proposed [Washio and Kitamura, 1992; Washio et al., 1993].

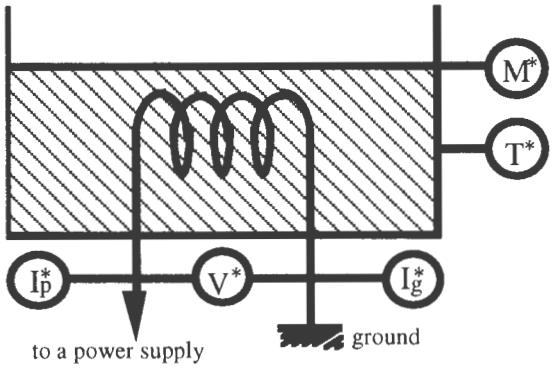


Figure2 An electric water heater
(* stands for observations by sensors.).

over-constrained subset of nth order:C^n
a set of m constraints in the SD involving n undetermined quantities where m>n and all constraints form a connected graph.
minimal over-constrained subset of nth order:M^n
a set of (n+1) constraints in the SD involving n undetermined quantities and not involving any other over-constrained subsets where all constraints form a connected graph.

The undetermined quantity is neither of a directly observed quantity and a nominally fixed quantity in the SD and the OBS under the premise 2. This categorization between undetermined and determined quantities introduces the information of sensor arrangement explicitly to the diagnostic constraints. The following assumption must be introduced for the valid use of these definitions in the consistency checking.

Assumption 1 : The model constraints $\{c \mid c \in SD\}$ are mutually independent which provide each minimal over-constrained subset M well-posed.

The independency of model constraints in a nonlinear system is not always guaranteed, because the relations among quantities are state-dependent. However, the model constraints of a process system are almost independent under its normal operations in practical applications. Hence, the present over-constraint condition can be adopted widely to process systems.

Although the efficiency of the derivation of all minimal over-constrained subsets is not the main issue for the off-line preparation of this knowledge in advance, a generic and efficient algorithm depicted in fig.3 capable of handling practically large scale models has been investigated [Washio, 1992; Washio et al., 1993]. The S is a constraint-quantity matrix of the SD. Each (i,j) element of the S is 1 if the ith constraint contains the jth quantity, otherwise it is 0. This algorithm is far more efficient than the thorough search, since the enumeration number of subsets in the SD within this algorithm is almost proportional to 2^N where N is the number of the undetermined quantities in the SD. Whereas, that of the thorough search is 2^K where K is the size of the SD and larger than N.

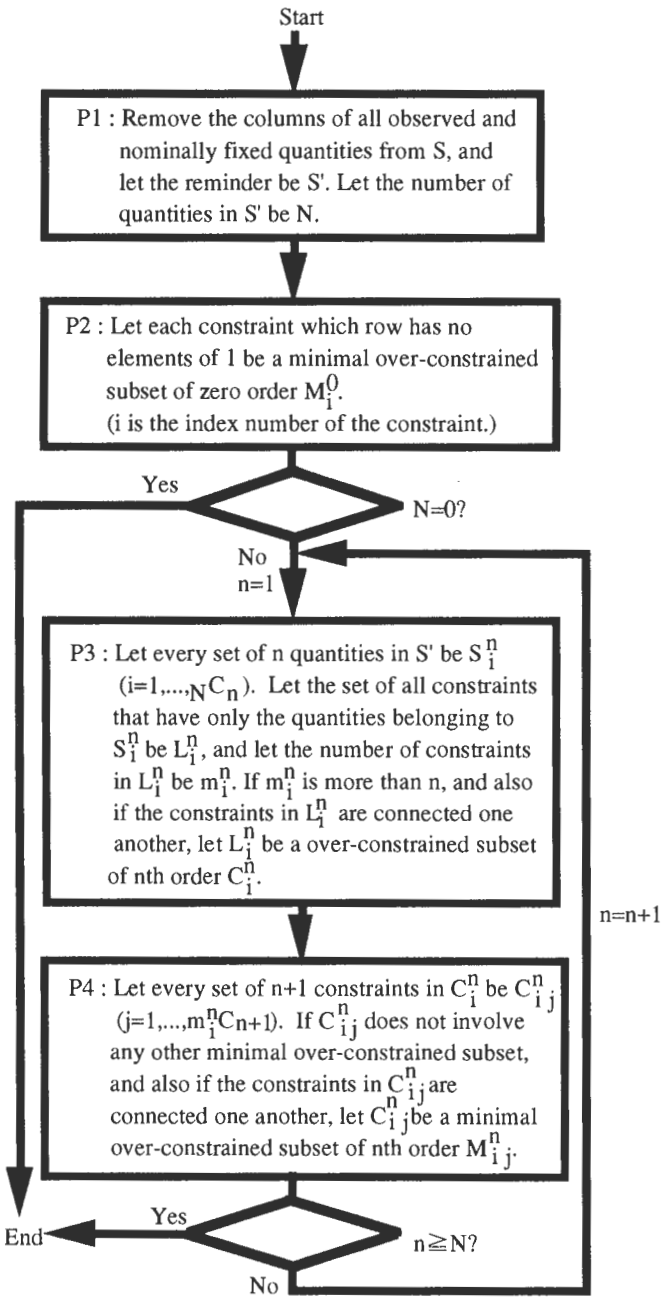


Figure 3 An algorithm to derive all minimal over-constrained subsets.

By applying the above algorithm to eq.(1)-(12), totally the following 10 minimal over-constrained subsets are derived.

$$\left.\begin{array}{l} M^3_1 = \{1,2,8,9\} \\ M^5_1 = \{1,3,7,8,10,12\} \\ M^5_2 = \{2,3,7,9,10,12\} \\ M^7_1 = \{1,4,5,6,8,10,11,12\} \\ M^7_2 = \{2,4,5,6,9,10,11,12\} \\ M^7_3 = \{3,4,5,6,7,10,11,12\} \\ M^8_1 = \{1,3,4,5,6,7,8,10,11\} \\ M^8_2 = \{1,3,4,5,6,7,8,11,12\} \\ M^8_3 = \{2,3,4,5,6,7,9,10,11\} \\ M^8_4 = \{2,3,4,5,6,7,9,11,12\} \end{array}\right] \quad (13)$$

2.2 Consistency Checking

The following definitions and the associated theorems establish a systematic scheme of this consistency checking. The "deletion" of a constraint c from the SD is defined as an operation to remove the c while remaining the quantities involved in the c. A "self-contained subset" is a subset of the SD in which the number of undetermined quantities is identical to that of the constraints while forming a connected graph [Simon, 1977; Iwasaki and Simon, 1986]. A self-contained subset determines the values of its quantities by itself. For example, if the deletion of eq.(10) in the minimal over-constrained subset M^5_2 is performed, the resultant constraints in the M^5_2 become as follows.

$I_g = I$ (2), $V = IR$ (3), $R = r+k(T-t_c)^2$ (7),
$I_g^* = I_g$ (9), V^* (10), $T^* = T$ (12)

V* and V remain in eq.(10) and eq.(3) of the model, respectively. The constraints of eq.(2), eq.(3), eq.(7), eq.(9) and eq.(12) form a self-contained subset of five undetermined quantities and five constraints. Also, the eq.(10) forms a tiny self-contained subset of zero undetermined quantity and zero constraint.

[Theorem 1] If the deletion of any one constraint c in a minimal over-constrained subset M is conducted, the M becomes one or more self-contained subset(s). ■

<Proof> By definition, the M becomes a subset of n constraints with n undetermined quantities by the deletion of a constraint.

(i) In case that the extra connections exist among quantities involved in the deleted c, the M remains to form a connected graph, and thus becomes a self-contained subset.

(ii) In case that the deleted c involves some unique connections among quantities, the M is partitioned into Q new subsets ($Q \geqq 2$). Each new subset involves k_i undetermined quantities ($n=\sum_{i=1}^{Q} k_i$). As the M forms a connected graph by definition, each new subset also forms a connected graph. Furthermore, as the M does not involve any other over-con-

strained subset by definition, each new subset which is a part of the original M is not over-constrained. Therefore, each new subset involves k_i or less constraints. On the other hand, the total number of the undermined quantities in all new subsets, i.e., n, is identical to the total number of the remaining constraints. Accordingly, each new subset involves k_i constraints which is identical to the number of undetermined quantities in the subset, and hence each new subset is a self-contained subset. □

[Theorem 2] Any undetermined quantity x in a minimal over-constrained subset M appears in two or more constraints within the M. ■

<Proof> An assumption is introduced that an undetermined quantity x belongs to a unique constraint c in a minimal over-constrained subset M. In this case, the following smaller minimal over-constrained subset M' having n constraints and n-1 undetermined quantities can be always obtained by the removal of the c from the M.

$M' = M - c \subset M$

This is contradictory to the definition of minimal over-constrained subsets. □

[Theorem 3] Two or more self-contained subsets which can independently determine the value of an undetermined quantity x in a minimal over-constrained subset M always exist in the M. ■

<Proof> Due to the theorem 2, a constraint c involving an undetermined quantity x can be selected for its deletion from the multiple candidates in the M. The deletion of the c always derives one or more self-contained subset(s) in the M as stated in the theorem 1. Hence, two or more self-contained subsets which can determine the value of the x respectively are always obtained by the deletion of each c involving the x in the M. □

Any undetermined quantity x in a minimal over-constrained subset M can be chosen for the comparison among its values derived by the multiple self-contained subsets in the M. Once the self-contained subsets for the derivation of the x have been set, the values of all undetermined quantities including the x in those subsets are sequentially determined by following the scheme of the causal ordering [Simon, 1977; Iwasaki and Simon, 1986; Iwasaki, 1989] while treating the directly observed and nominally fixed quantities as exogenous quantities. If the residuals among the values of the x exceed a certain threshold value, some constraints in the M are considered to be faulty. This procedure is applied to every minimal over-constrained subset M in the SD.

In the example of the electric water heater, the following multiple failures are numerically simulated.

> The electric shield of the resistant wire and the voltage sensor were broken by a mechanical shock at the time 200sec. 30% of the electric current began to leak between the power supply and the resistant wire, and the indication of the voltage sensor has been changed and fixed at the level of 150V.

Ripples of 20% sine wave were added to the voltage of the power supply in order to evaluate the performance of the consistency checking in the dynamic behavior. Figure 4 represents the result of the consistency checking for each minimal over-constrained subset M. The undetermined quantity x for the checking was arbitrarily chosen in each M. All subsets except M^8_4 became inconsistent at the time 200sec.

Generally speaking, each minimal over-constrained subset is not very robust to the errors in the system model and the observation noise because of its low redundancy for consistency checking of an undetermined quantity x. However, various and efficient remedies in the field of numerical state estimation theory can be applied to this difficulty. For instance, Kalman filter technique [Kalman, 1960] provides a powerful measure to distinguish the physical inconsistency from the observation noise. In the mean time, the difficulty is also reduced, as the failure identification described in the following section usually has gradually degraded performance for the misjudgments in the consistency checking.

2.3 Failure Identification

The minimal conflicts can be easily derived from the result of the aforementioned consistency checking based on any method of "minimal diagnoses" [Reiter, 1987; de Kleer and Williams, 1987] and "kernel diagnoses" [de Kleer et al., 1992], because each minimal over-constrained subset on the SD is also the collection of constraints c belonging to the COMPS as mentioned previously. In our current work for process diagnosis, the following assumption is introduced.

Assumption 2 : The mutual cancellation of anomalous behaviors of multiple failures is hardly occurred in process systems.

This assumption makes our diagnosis basically equivalent to Raiman's approach [Raiman, 1990]. Once all minimal conflicts are obtained, the following standard procedure derives the possible sets of faulty constraints [Reiter, 1987; de Kleer and Williams, 1987; de Kleer et al., 1992].

(1) "Multiply" the minimal conflicts to give a disjunction of conjunctions.
(2) Delete any conjunction containing a complementary pair of literals.
(3) Delete any conjunction covered by some other conjunction.
(4) The remaining conjunctions are the prime implicants of the original minimal conflicts.

These prime implicants are the possible interpretations of faulty states. The following premise named "minimal principle" [Reiter, 1987] being ordinary in many diagnosis methods is also adopted in our work.

Premise 4 : The combination of abnormal constraints explicitly stated in each prime implicant is faulty, and the rests are considered to be normal.

In the example, as the expression of "All equations in M^3_1 are normal." is against the inconsistent result of M^3_1, its negation

$$M^3_1 : AB(1) \vee AB(2) \vee AB(8) \vee AB(9), \qquad (14)$$

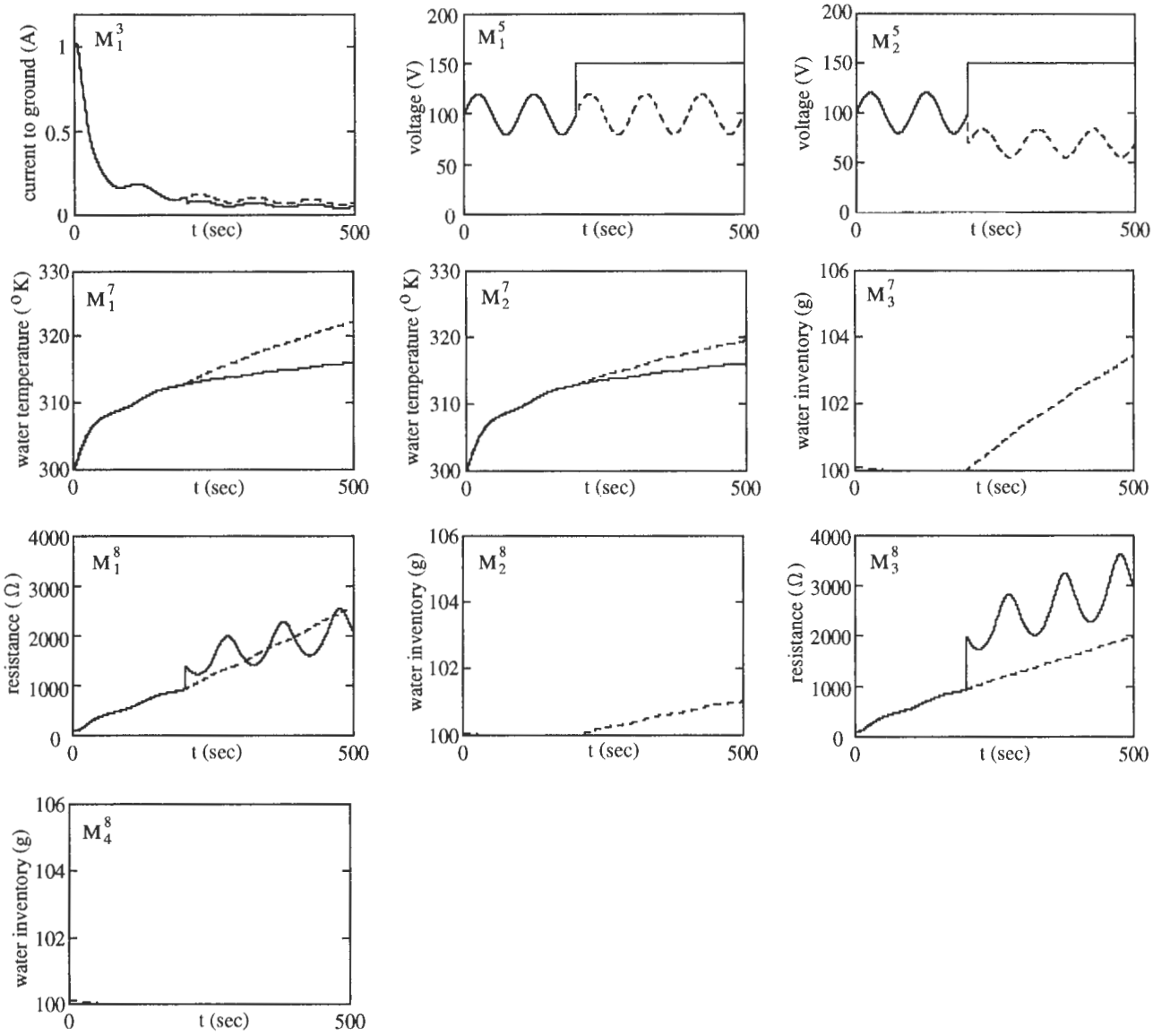


Figure 4 Result of consistency checking (solid lines:observations, dashed lines:evaluations by constraints).

becomes a minimal conflict. The minimal conflicts for the other inconsistent minimal over-constrained subsets are derived as well.

M^5_1: AB(1)∨AB(3)∨AB(7)∨AB(8)∨AB(10)∨AB(12),
M^5_2: AB(2)∨AB(3)∨AB(7)∨AB(9)∨AB(10)∨AB(12),
M^7_1: AB(1)∨AB(4)∨AB(5)∨AB(6)∨AB(8)∨AB(10)∨ AB(11)∨AB(12),
M^7_2: AB(2)∨AB(4)∨AB(5)∨AB(6)∨AB(9)∨AB(10)∨ AB(11)∨AB(12),
M^7_3: AB(3)∨AB(4)∨AB(5)∨AB(6)∨AB(7)∨AB(10)∨ AB(11)∨AB(12),
M^8_1: AB(1)∨AB(3)∨AB(4)∨AB(5)∨AB(6)∨AB(7)∨ AB(8)∨AB(10)∨AB(11),
M^8_2: AB(1)∨AB(3)∨AB(4)∨AB(5)∨AB(6)∨AB(7)∨ AB(8)∨AB(11)∨AB(12),
M^8_3: AB(2)∨AB(3)∨AB(4)∨AB(5)∨AB(6)∨AB(7)∨ AB(9)∨AB(10)∨AB(11).
(15)

The notation of "An equation in M^8_4 is abnormal." is against the consistency of M^8_4 under the assumption 2. Thus, the minimal conflicts of M^8_4 are

M84 : ¬AB(2), ¬AB(3), ¬AB(4), ¬AB(5), ¬AB(6), ¬AB(7), ¬AB(9), ¬AB(11), ¬AB(12). (16)

The aforementioned procedure (1)-(4) derives the following results from these minimal conflicts.

AB(1)∧ ¬AB(2)∧ ¬AB(3)∧ ¬AB(4)∧ ¬AB(5)∧ ¬AB(6)∧ ¬AB(7)∧ ¬AB(9)∧AB(10)∧ ¬AB(11)∧ ¬AB(12) (17)

¬AB(2)∧ ¬AB(3)∧ ¬AB(4)∧ ¬AB(5)∧ ¬AB(6)∧ ¬AB(7)∧AB(8)∧ ¬AB(9)∧AB(10)∧ ¬AB(11)∧ ¬AB(12) (18)

The eq.(17) stands for the violation to the electric current balance between the power supply and the resistant wire (eq.(1))

and the anomaly of the voltage sensor (eq.(10)). This result is correct for the original failures in the simulation. An erroneous solution of the eq.(18) can not be eliminated under the SD and the OBS.

3 Identification of Anomalous Quantities and Their Quantitative Deviations

3.1 Causal Ordering and Identification of Anomalous Quantities

Causal ordering [Simon, 1977; Iwasaki and Simon, 1986; Iwasaki, 1989] is required to identify anomalous quantities directly disturbed by faulty mechanisms. In the conventional framework, the determination orders of process quantities are derived based on the specification of exogenous quantities in the system and the time derivative quantities to change their integrals. However, any of the inlet flow and the outlet flow can be exogenous in a water pipe, because they just mutually balance. Furthermore, in Faraday's law of induction,

$$dB/dt = -rot(E) \quad \text{or} \quad B=-\int rot(E)dt, \qquad (19)$$

B : magnetic flux density, E : electric field intensity,

the change of B directly determines the value of E, i.e., the time integral determines its time derivative within a fundamental physical law. Accordingly, the arbitrary specification of exogenous quantities and the unique assumption of the causality in time differential equations may mislead the result of the causal ordering for physical systems. This discussion we have made [Washio, 1989] is also supported by Y. Iwasaki and H.A. Simon [Iwasaki and Simon, 1993].

The authors proposed an extended theory to reduce the ambiguity of the causal ordering for physical systems [Washio, 1989; Washio and Kitamura, 1992; Washio et al., 1993]. The specific heat law (eq.(6)) in our example defines the quantitative relation between H and T under the exogenously given heat capacity cM. Either of the values of H and T is physically determined in this law, but cM is not changed by H and T within this law. The authors named this type of the application independent constraints on the direction of the disturbance propagation among quantities in a physical constraint as "inherent causal structure" of the constraint [Washio, 1989; Washio et al., 1993]. The details of the generic method to determine the inherent causal structure of each equation can bee seen in authors' works [Washio, 1989; Washio, 1990]. Once the inherent causal structure of each equation has been identified, its knowledge representation with the quantitative relation of the equation is given by the following manner. First, let X_ℓ be a set of exogenously given quantities in the equation, and let Y_ℓ be a set of the other quantities in the equation. Any element in Y_ℓ has a possibility to be physically determined. Subsequently, the quantities in each set are located on either of the right hand side (rhs) and the left hand side (lhs) by the following rule.

$$\begin{aligned} &\text{if } X_\ell \neq \{\phi\} \text{ then } G_\ell(Y_\ell) = F_\ell(X_\ell), \\ &\text{if } X_\ell = \{\phi\} \text{ then } G_\ell(Y_\ell) = 0, \qquad (20) \\ &\text{where } X_\ell \cap Y_\ell = \phi,\ Y_\ell \neq \phi, \end{aligned}$$

F_ℓ :rhs of equation, and G_ℓ :lhs of equation.

This knowledge representation of a equation is called as an "assumptive structural equation".

If an assumptive structural equation has only one quantity on its lhs, the value of the quantity is uniquely determined by the other quantities on the rhs. Thus,

determining equation: an equation having unique quantity on the lhs,
determined quantity : the unique quantity on the lhs of a determining equation,

are defined. When the model of the objective system is a set L of the assumptive structural equations, let a set of all quantities in L be S. The unambiguous determination orders of the quantities in L can be derived by the systematic algorithm depicted in fig.5. Its resultant revised equations stand for the determination orders of the quantities from their rhses to the lhses.

The model of the electric water heater can be represented by the following assumptive structural equations [Washio, 1989; Washio, 1990].

$$I-I_p = 0 \ (1'),\ I_g-I = 0 \ (2'),\ V/I = R \ (3'),\ F_h = VI \ (4'),$$
$$H = \int_{-\infty}^{t} F_h dt \ (5'),\ H/T = cM \ (6'),\ R = r+k(T-t_c)^2 \ (7'),$$
$$I_p{*}=I_p \ (8'),\ I_g{*}=I_g \ (9'),\ V{*}=V \ (10'),\ M{*}=M \ (11'),\ T{*}=T \ (12')$$

All quantities in eq.(1') and (2') are located on their lhses, because they are balance equations. The eq.(3) (Ohm's law) defines the relation between V and I under an exogenously given resistance R. The heat generation rate F_h in eq.(4') (Joule's law) is unidirectionally determined by V and I, because this law represents an irreversible process in a thermodynamic phenomenon. The eq.(5') stands for a standard time evolution. The structure of eq.(6') has been aforementioned. The eq.(7) represents another irreversible process from T to R. The rests are for sensors, and their structures are trivial. The causal ordering procedure of fig.5 is applied to this

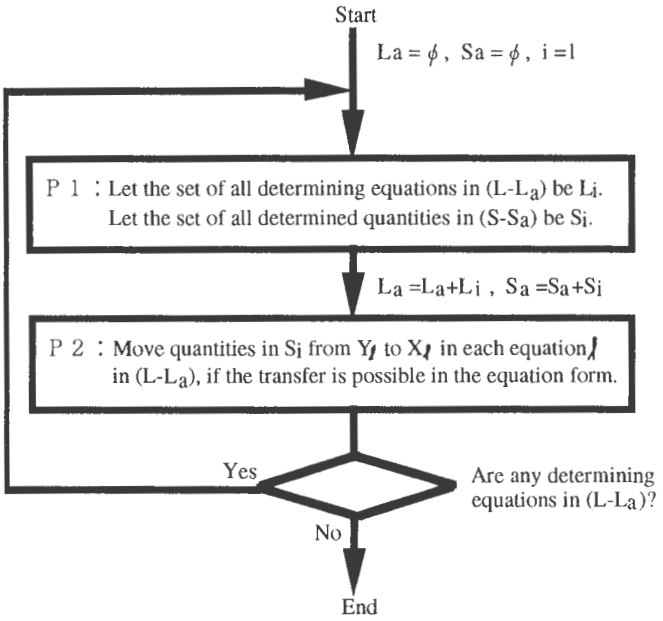


Figure 5 An algorithm of extended causal ordering.

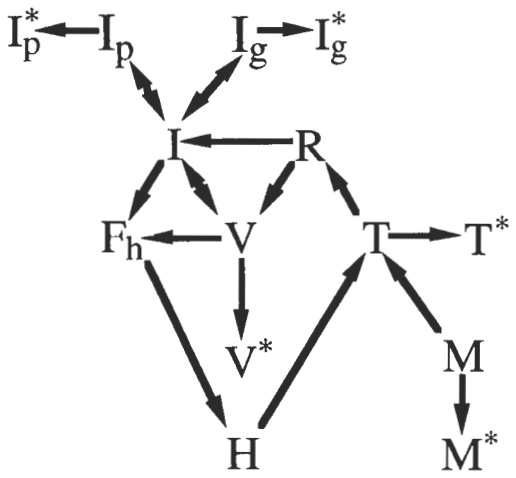


Figure 6 The causal network of an electric water heater.

model. In the step P1, eq.(4'),(5') and (7')-(12') are identified as determining equations. In the step P2, a determined quantity H of eq.(5') is moved from the lhs to the rhs in eq.(6').

$$T = H/(cM). \qquad (6'')$$

As no other determined quantities appear in any lhses, the procedure goes back to the step P1. Then a new determined quantity T in eq.(6") is identified. However, the loop is halted in the step P2, because no T exists in any lhses. The resultant equations of eq.(1')-(5'), (6"), and (7')-(12') indicate the determination orders of the quantities. The orders are depicted in form of a causal network in fig.6. The quantities remaining on each lhs of eq.(1'), (2') and (3') influence bidirectionally.

In case of eq.(17) resulted in the aforementioned diagnoses, the anomalous quantities directly disturbed by the fault of eq.(1) are identified as any of I_p and I based on the final structure of eq.(1'). In practice, any of I_p and I can be changed by the break of the electric shield between the power supply and the resistant wire. Also, the quantity directly disturbed by eq.(10) is identified as V*.

Many physical systems partially involve the bidirectional causality as shown in this example, and the derivation of the exact causal structure of large systems is highly difficult within our physical intuition. Accordingly, this systematic causal ordering method provides an efficient remedy to identify anomalous quantities directly disturbed by faulty mechanisms.

3.2 Evaluation of Quantitative Deviations

The following theorem assures the ability to evaluate the quantitative deviation of any anomalous quantity directly disturbed by any multiple failures identified in the section 2.3.

[Theorem 4] For any abnormal constraint AB(c) belonging to a diagnosis : D, at least one inconsistent minimal over-constrained subset exists which involve AB(c) and does not involve the other abnormal constraints in the D. ■

<Proof> An assumption is introduced that any inconsistent minimal over-constrained subset involving the AB(c) involves some other abnormal constraints in a D. In the step (1) of the standard procedure described in the section 2.3, the following smaller diagnosis D' can be always obtained by selecting an abnormal constraint except the AB(c) from every minimal conflict corresponding to each inconsistent minimal over-constrained subsets.

$D' = D - AB(i) \subset D$

This is contradictory to the requirement in the step (3) of the procedure that D does not involve any other diagnoses. □

[Theorem 5] When a quantity x is contained in an AB(c), any minimal over-constrained subset M involving the c involves a self-contained subset which determines the value of x without including the c. In the mean time, the minimal over-constrained subset M involves another self-contained subset which determines the value of x by using the c. ■

<Proof> Due to the aforementioned theorem 3, the former self-contained subset is derived by the deletion of the c in the M. The latter is obtained by the deletion of a constraint connected with the c through x in the M. □

As a consequence of the theorem 4 and the premise 4, for every AB(c) directly disturbing anomalous quantity x, one minimal over-constrained subset always exists in which the AB(c) is the unique abnormal equation. Accordingly, the actual anomalous value and the normal value of each anomalous quantity x can be always evaluated by the former and latter self-contained subsets in the theorem 5. The value of the x in these subsets is determined by the causal ordering similarly to the section 2.2.

In case of the diagnosis of eq.(17), the anomalous quantities are I_p, I disturbed by the eq.(1) and V* by the eq.(10). As the minimal over-constrained subset M^3_1 involves eq.(1) but not eq.(10), the actual anomalous value of I_p is evaluated by the subset {8} which is obtained by the deletion of eq.(1) from M^3_1. Also, the normal value of I_p is evaluated through

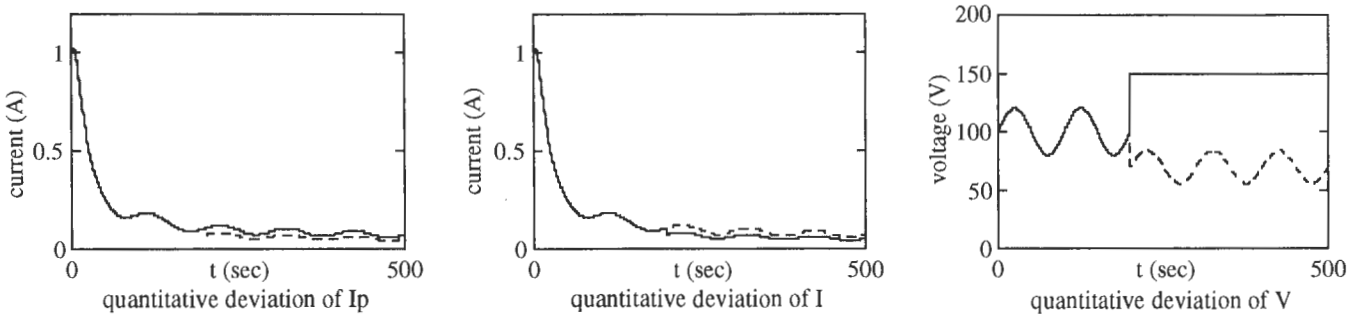


Figure 7 Deviations of anomalous quantities (solid lines:actual anomalous values, dashed lines:normal values).

the subset {1,2,9} obtained by the deletion of eq.(8) from M^3_1. The actual anomalous value and the normal value of I are evaluated through {2,9} and {1,8} respectively in the same manner. For the quantity V*, its actual anomalous value is obtained by its direct measurement. The normal value is derived by the M^5_2 which involves eq.(10) but not eq.(1). Figure 7 shows the quantitative deviations of these anomalous quantities. These results are quantitatively consistent with the multiple failures introduced in the simulation.

4 Related Works

In the ATMS-based methodology [de Kleer and Williams, 1987], conflicts are generated incrementally as new measurements are made. A heuristic probing of the obvious and semi-obvious conflicts using causality information [Bakker and Bourseau, 1992] and one step look ahead random probing [de Kleer et al., 1992] indicate good efficiency to identify faults, when the objective system is large and has many possible probing points. On the other hand, the preparation of all minimal over-constraints beforehand in our approach usually dose not face the difficulty of the combinatorial explosion, since the size of COMPS and the number of given sensing points in a process component are quite limited. The definition of minimal over-constraints does not depend on any causality information.

The idea to prepare all schemes for consistency checking in advance has also been presented by Biswas and Yu [Biswas and Yu, 1993]. They proposed "partial conflicts" to derive a conflict for each observation. The elements of COMPS in their work are parameters attributed to each process mechanism. Their framework essentially requires the linearization of process models and the steady state assumption of the process, and hence is not applicable to highly nonlinear and dynamic systems. On the contrary, the basic element in our approach is a constraint among the parameters and state variables. The nonlinear and dynamic features of the system do not limit its application.

5 Conclusion

The operations and the knowledge used in this method are systematic, complete, well-defined and well-combined to synthesize an efficient and reliable procedure for diagnosis. This proposed method can diagnose multiple failures of component mechanisms and sensors occurred in a system. Nonlinear and dynamic process in which the quantities are intimately connected one another can be diagnosed in high resolution. As the computational load required in the on-line processing is quite limited, the real-time and quantitative diagnosis can be performed without loosing the maximum performance of this method. The specifications of this method can meet the severe requirements in the practical applications.

Acknowledgments

The authors wish to express our thanks to Dr. Hiroshi Motoda in Hitachi Advanced Research Laboratory and Prof. Toyoaki Nishida in Advanced Institute of Science and Technology for their useful comments. The authors extend the gratitude to Dr. Shuichi Koike and Dr. Hideaki Takahashi in Mitsubishi Research Institute, Inc. for their extensive support.

References

[de Kleer and Williams, 1987] de Kleer, J. and Williams, B.C. : Diagnosing Multiple Faults, Artificial Intelligence, Vo.32, pp.97-130, 1987.

[Hamilton, 1988] Hamilton, T.P. : HELIX:A Helicopter Diagnostic System Based on QualitativePhysics, Artificial Intelligence in Engineering, Vol.3,No.3, pp.141-150, 1988.

[Torasso and Console, 1989] Torasso, P. and Console, L.: Diagnostic Problem Solving, North Oxford Academic, 1989.

[Reiter, 1987] Reiter, R. : A Theory of Diagnosis from First Principles, Artificial Intelligence, Vol.32, pp.57-95, 1987.

[de Kleer et al., 1992] de Kleer, J., Mackworth, A.K. and Reiter,R. : Characterizing diagnosis and systems, Artificial Intelligence, Vol.56, pp.197-222, 1992.

[Washio and Kitamura, 1992] Washio, T. and Kitamura, M. : A New Approach for Plant Component Diagnosis Based on Credible and Transparent Physical Knowledge, Proc. of 8th Power Plant Dynamics, Control & Testing Symposium, pp.15.01-15.16, American Nuclear Society, 1992.

[Washio et al., 1993] Washio, T., Sakuma, M., and Kitamura, M. : A Diagnosis Method for Multiple Process Failures, Proc. of DX-93:Fourth International Workshop on Principles of Diagnosis, pp.327-340, Univ. College of Wales, Aberystwyth, UK, 1993.

[Simon, 1977] Simon, H.A. : Models of Discovery, D. Reidel Pub. Co., Dordrecht, Holland, 1977.

[Iwasaki and Simon, 1986] Iwasaki, Y. and Simon, H.A. : Causality in Device Behavior, Artificial Intelligence, Vol.29, No.1, pp.3-32, 1986.

[Iwasaki, 1989] Iwasaki, Y. : Causal Ordering in a Mixed Structure, Proc. of AAAI-88, Vol.1, pp.313-318, 1988.

[Kalman, 1960] Kalman, R.E. : A new approach to linear filtering and prediction problems, Trans. ASME, Series D, J. of Basic Emgineering, Vol. 82, No.1, pp.33-45, 1960.

[Washio, 1989] Washio, T. : Causal Ordering Methods Based on Physical Laws of Plant Systems, MITNRL-033, MIT Nuclear Reactor Laboratory, 1989.

[Raiman, 1990] Raiman, O. : A circumscribed diagnosis engine, Proc. of International Workshop on Expert Systems in Engineering, Lecture Notes in Artificial Intelligence, Vol.462, pp.90-101, Springer, Berlin, 1990.

[Iwasaki and Simon, 1993] Iwasaki, Y. and Simon, H.A. : Retrospective on "Causality in device behavior", Artificial Intelligence, Vol.59, pp.141-146, 1993.

[Washio, 1990] Washio, T. : Derivation of Exogenously-Driven Causality Based on Physical Laws, Journal of Japanese Society of Artificial Intelligence, Vo.5, No.4, pp.482-491, 1990 (in Japanese).

[Bakker and Bourseau, 1992] Bakker, R.R. and Bourseau, M.: Pragmatic Reasoning in Model-Based diagnosis, Proc. of 10th European Conference on Artificial Intelligence, pp.734-738, 1992.

[de Kleer et al., 1992] de Kleer, J., Raiman, O., and Shirley, M.H. : One step lookahead is prettygood, Reading in Model-Based Diagnosis, edited W. Hamscher, J. de Kleer and L. Console, pp.138-142, Morgan Kaufmann, 1992.

[Biswas and Yu, 1993] Biswas, G. and Yu, X. : A Formal Modeling Scheme for Continuous Systems: Focus on Diagnosis, Proc. of 13th International Joint Conference on Artificial Intelligence, pp.1474-1479, 1993.

Automated Model Generation and Simulation*

Kyungsook Han and **Andrew Gelsey**
kshan@cs.rutgers.edu gelsey@cs.rutgers.edu
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
U. S. A.

Abstract

Understanding or predicting the behavior of a complex physical system requires the construction and execution of a model of the system. Such a model is often handcrafted by the person studying the system, and the modeling process is not formalized to be reusable by others. We describe a method which uses first principles to automatically create models and simulators for complex motions, and an implemented system called ORACLE. Given a description of a problem involving a physical system, ORACLE automatically identifies relevant model fragments, instantiates them for the particular entities and physical phenomena in the problem, composes the instantiated fragments to form a model, and executes the model. Knowledge of physical phenomena is represented with general model fragments which can be shared and reused by many models. Experimental results show that the method is capable of generating correct models of several different types of physical systems if enough domain knowledge is available.

1 Introduction

Solving problems about complex physical systems generally involves the creation and execution of models of the physics needed to reason about the problem. Models are normally constructed by the person studying the system. Despite the considerable time and effort spent, a handcrafted model is often error-prone. Modifying a handcrafted model to solve similar problems about other physical systems is also difficult, and may take more time than building a new model for the systems. The work of this paper is motivated by two goals. The first goal is to automate the model formulation and simulation process for complex spatial reasoning tasks. In particular, we focus on an important subclass of spatial reasoning – moving objects. The second goal is to make the modeling process as general as possible so that common domain theories can be shared and reused instead of being duplicated.

Consider a spring one end with attached to a fixed point and the other end attached to a block, as illustrated in Figure 1a. This harmonic oscillator is a common textbook example which is often used in qualitative physics research. It is well known that the oscillator has one degree of freedom, i.e., displacement of the block from its equilibrium position, and its motion is oscillatory on a straight line. However, if you consider a block and a spring in more general configuration (Figure 1b), predicting the behavior is not as simple as before. Is the motion going to be still oscillatory? More interesting questions include: (1) What if a spring is attached to a corner of a block instead of the center of the face? (2) What if a block attached to a spring is put in arbitrary position and orientation before being released? (3) What if two blocks are connected by a spring? (4) What if multiple blocks connected by multiple springs are put in arbitrary positions and orientations? (see Figure 2 for an example)

Different forms of these problems require spatial reasoning to formulate equations of motion, in particular the ability to reason explicitly about vector quantities and moving frames of reference. Many qualitative physics approaches by AI researchers which can solve the linear harmonic oscillator problem [Forbus, 1984; Kuipers, 1986; Struss, 1988; Weld, 1988; Williams, 1986] cannot handle the more complex problems we describe above because they lack this spatial reasoning ability.

Consider, now, predicting the behavior of another physical system, a sailboat (Figure 3), which appears

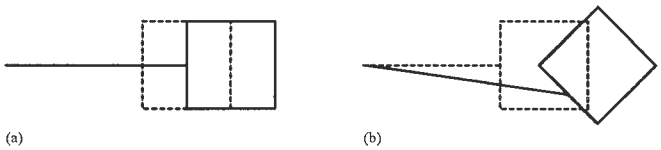


Figure 1: (a) The block on a spring is pulled from its equilibrium position and released. (b) The block is pulled and rotated from its equilibrium position and released.

*This research was partially supported by the Advanced Research Projects Agency (ARPA) and National Aeronautics and Space Administration under NASA grant NAG2-645 and by the National Science Foundation through grant CCR-9209793.

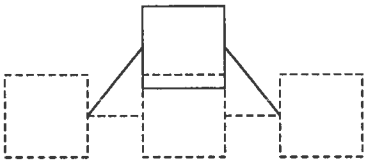


Figure 2: 3 blocks connected by 2 springs. The middle block is pulled directly to the side and released.

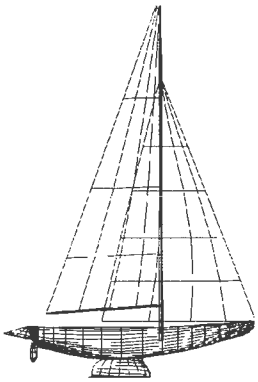


Figure 3: *Stars & Stripes*, winner of the 1987 America's Cup competition.

quite different from the spring-block systems. Can the modeling system of the spring-block systems be used to predict the behavior of a sailboat? Do we need a different modeler? Or the same modeler, but just more "knowledge structures" of the same types that the modeler of the spring-block systems handles?

The remainder of this paper provides an overview to our approach to the problem of automating the model formulation and simulation process, and describes a modeling system called ORACLE, implemented in the mathematical manipulation language Maple [Char *et al.*, 1991]. We will use the spring-block system of Figure 1b as a running example, and later show how ORACLE handles a multiple spring-block system and how it is extended to handle different physical systems.

2 A Framework for Model Building and Simulation

2.1 Ontology and Representation

The principal elements of our ontology are entities, phenomena, model fragments, and models, each represented in a frame [Minsky, 1975]. An *entity* is a physical object which either constitutes a physical system by itself (i.e., primitive object) or is a part of a physical system (i.e., composite object). The properties of an entity are expressed as variables in equations. The block entity, for example, has properties such as position and velocity. An entity is represented in a frame with slots for the properties. Facets allowed in a slot are *value*, *form*, *if_needed*, and *if_added*. The value facet is initially set to null but will be assigned a vector, scalar, string, set, or any other expression as it becomes known. The form facet distinguishes the slot type (e.g., scalar or vector) and is consulted when the system creates a new Maple

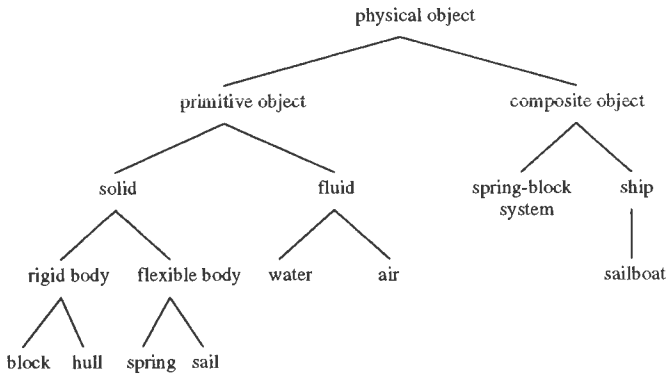


Figure 4: Taxonomy of entities for the examples of the paper. Properties of a class are inherited to its subclasses. For example, properties of rigid body are inherited by block and hull.

variable name during the problem solving process. For example, if the system is asked to compute the position of a block b1, a set of new variables {b1x(t), b1y(t), b1z(t)} will be created for the position vector and used in equations. The if_needed facet or if_added facet holds the procedure call, invoked when a slot value is needed or added. The if_needed procedure of the velocity slot in the example below says that velocity is derivable from position. The entities are organized in a tree as shown in Figure 4, which includes a set of entities to be used in the examples of this paper.

```
block=[AKO=rigid_body,
    position(t)=[value=null, form=[x(t),y(t),z(t)]],
    velocity(t)=[value=null, form=[u(t),v(t),w(t)],
        if_needed=[derive_velocity, position(t)]],
        ....  (other slots not shown)  .... ]
```

A *phenomenon* is a process which changes one or more properties of an entity in a physical system. Force from a spring, for example, is a phenomenon which can change position and/or orientation of an entity which is attached to the spring.

A *model fragment* is a characterization of a physical phenomenon by a set of entities, variables, assumptions, and equations. There may be more than one model fragments for a single phenomenon, each with different assumption or approximation. The equations of a model fragment are applicable when the corresponding phenomenon occurs. The spring force, for example, exerted on an object attached to end2 of a linear spring with linear damping is represented as follows (syntax slightly modified for readability):

```
Springforce2=[phenomenon='spring force at end2',
    entities=[s=linear_damped_spring],
    variables=[k=s[force_const], b=s[damping_coeff],
        e1(t)=s[end1(t)], e2(t)=s[end2(t)],
        l=s[rest_length], f(t)=s[force2(t)]],
    equations=[f(t)=-k*(||e2(t)-e1(t)||-l)*
        (e2(t)-e1(t))/||e2(t)-e1(t)||-
        b*diff((||e2(t)-e1(t)||-l)*(e2(t)-e1(t))/
        ||e2(t)-e1(t)||,t)]]
```

It says that s is a linear damped spring, k is a force constant of the spring, b is a damping coefficient, e1(t) and e2(t) are the position vectors of end1 and end2, l is the rest length, and f(t) is the spring force at

end2. $\| e2(t) - e1(t) \|$ is the vector norm representing the length of the spring at time t, $\| e2(t) - e1(t) \| - l$ is the signed length change from the rest length, and $(e2(t) - e1(t))/ \| e2(t) - e1(t) \|$ is a unit vector with direction from end1 to end2.

A *model* is a composition of model fragments applicable to a physical system in a particular situation. *Simulation* is the execution of a model.

The motion of an entity at any instant can be described by a set of differential equations in the twelve components of four vectors: position, orientation, velocity, and angular velocity.[1] The differential equations are usually nonlinear and do not have a solution in closed form, so they must be solved by numeric integration. For a moving entity, ORACLE constructs a model with the four vectors (position, orientation, velocity, and angular velocity) as *state variables*, which take numeric values during simulation.

The state variables of each subpart of an entity are initially defined in the local reference frame, which is assumed to be fixed to the entity. Then each subpart defined in its local reference frame is translated and rotated by having its reference frame redefined in a common inertial reference frame. The system chooses the common inertial reference frame from local reference frames which are not accelerated. If there is no such reference frame (i.e., all the local reference frames are noninertial), it introduces a new inertial reference frame. If there are several inertial reference frames, the choice is arbitrary.

2.2 The Algorithm

ORACLE takes as input a structural description of a physical system supplemented with information about nonstructural properties, constraints (if any) and variables of interests. As output, it produces a model of the motion of the system and the variable values obtained by solving the model. The algorithm of ORACLE consists of three phases: (1) problem analysis, (2) model creation, and (3) model execution. In the first phase, ORACLE represents each entity of a problem statement in a frame by copying a class frame and filling in slots for property values specified in the problem statement. It also transforms vector quantities expressed in the local reference frames into those in the inertial reference frame, formulates initial conditions, and executes if_added procedures in the slots. A model fragment specifying forces on a component of a composite object is instantiated by if_added procedures in the this phase. After constraints are analyzed, variables are examined to determine if their values are already known in their slot values or derivable from other variables. In the second phase, additional model fragments which have not been instantiated are retrieved and a model is constructed from them. In the final phase, the constructed model is solved for the problem. If ORACLE runs out of potentially relevant model fragments before it finds a valid solution, it prints the situation, asks more information, and quits. The top-level algorithm of ORACLE is outlined in Algorithm 1.

[1]The degrees of freedom of a moving entity are six instead of twelve because the velocity function and the angular velocity function are derivable by differentiating the position and orientation functions, respectively. The motion of a physical system with n subparts can be characterized by maximum $12n$ state variables with $6n$ degrees of freedom.

Algorithm 1 ORACLE's top-level algorithm

Problem Analysis Analyze a problem statement.

1. Analyze entities, and create frames of the entities and a set INIT of initial-value conditions.
2. For each constraint, determine its type and represent them in equations.
3. Analyze variables and generate a set DRVD of differential equations.

Model Creation Search for relevant model fragments and compose a behavioral model with them.

1. For each entity E of the problem statement
   For each model fragment MF indexed by the "mf" slot of E
   If MF has not been instantiated for E AND every variable of MF either corresponds to an entity property or variable of the input or can be derived from them AND the assumption (if any) of MF does not violate any entity property or constraint of the input
   Put MF in a list MFS.
2. model M = DRVD
3. #equations = #equations(M)
4. retry: For each model fragment MF in MFS
   (a) Instantiate MF for the problem.
   (b) $M = M \cup \{MF\}$
   (c) #equations = #equations(M)
   (d) If #equations = #variables, do **model execution.**
5. Print the dead-end situation, and quit.

Model Execution Solve the model M either analytically or by numeric simulation.

1. Determine the types of equations of the model and solve them with INIT for the variables.
2. If a valid solution is obtained, print the model and solutions, and quit.
3. If a valid solution is not obtained, retract the most recent MF from the model and go to retry.

2.3 An Example

We illustrate how ORACLE works with the spring-block system of Figure 1b. Suppose the following problem description is given as an input. There is no particular constraint in this problem and the system is asked to compute the four state variables of the block.

```
entities=[b1=[block, mass=1,
        principal_moments_of_inertia=[1/6,1/6,1/6],
        position(0)=[3,0,0],
        orientation(0)=[Pi/4,Pi/2,0],
        velocity(0)=[0,0,0],
        ang_velocity(0)=[0,0,0]],
    s1=[spring, force_const=10,
```

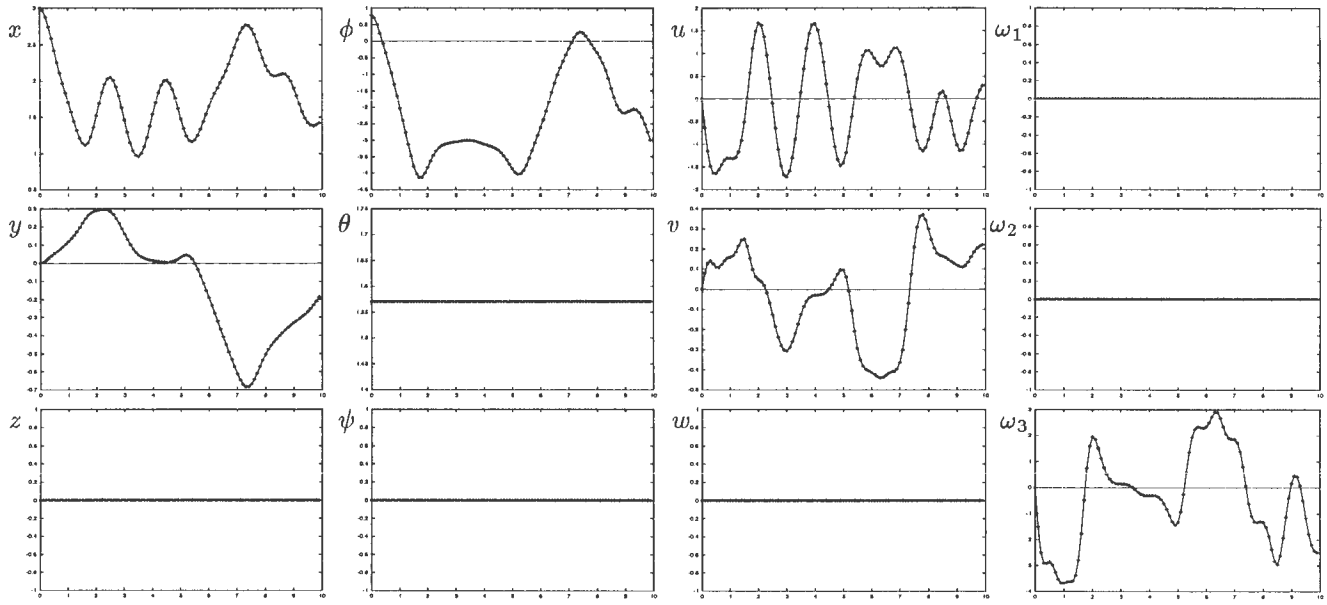


Figure 5: Plots of the 12 state variables of the block b1 as functions of time.

```
        damping_coeff=1/10, rest_length=3/2,
        end1(t)=[0,0,0], end2(t)=b1[-1/2,0,0]],
    sb=[composite_object, parts={b1,s1}]];
constraints=[ ];
variables=[b1[position(t)], b1[orientation(t)],
           b1[velocity(t)], b1[ang_velocity(t)]];
```

For each entity b1, s1, and sb, a frame is created and the given properties of the entities are recorded in their slot values. The if_added procedure in the end2 slot of s1 computes the spring force acting on b1 using a model fragment Springforce2 and records the value in the force slot of b1. The position of end2 in the inertial reference frame is computed from a translation and a rotation of the local reference frame of b1. The initial conditions of the block are also formulated.

```
INIT =
{b1x(0)=3, b1y(0)=0, b1z(0)=0,
 b1phi(0)=0, b1theta(0)=Pi/2, b1psi(0)=0,
 b1u(0)=0, b1v(0)=0, b1w(0)=0,
 b1omega1(0)=0, b1omega2(0)=0, b1omega3(0)=0}
```

None of the four state variables of b1 can be assigned a value simply by looking at slot values of b1, but the velocity and the angular velocity functions can be derived by differentiating the position and the orientation functions, respectively, according to their if_needed facets. The system generates trivial differential equations for the velocity and angular velocity by the procedures attached to the if_needed facets.

```
DRVD =
{b1u(t)=diff(b1x(t),t),
 b1v(t)=diff(b1y(t),t),
 b1w(t)=diff(b1z(t),t),
 b1omega1(t)=diff(b1theta(t),t)*cos(b1phi(t))+
    diff(b1psi(t),t)*sin(b1theta(t))*sin(b1phi(t)),
 b1omega2(t)=diff(b1theta(t),t)*sin(b1phi(t))-
    diff(b1psi(t),t)*sin(b1theta(t))*cos(b1phi(t)),
 b1omega3(t)=diff(b1phi(t),t)+
    diff(b1psi(t),t)*cos(b1theta(t))}
```

Now ORACLE focuses on finding equations for the position and the orientation. The equations for them cannot be derived from other variables since they are basic variables, so ORACLE looks for relevant model fragments. It examines model fragments, indexed by the mf slot of the block. ORACLE decides that Newton2 and Euler are potentially relevant because the entities (solid and rigid body, respectively) of the model fragments are superclass of a block and the equations of the model fragments contain at least one variable of the problem. Model fragments of Newton2 and Euler are as follows.

```
Newton2=[phenomenon='Newton's second law of motion',
         entities=[r=solid],
         variables=[f(t)=r[net_force(t)],
                    p(t)=r[momentum(t)]],
         assumptions=[ ],
         equations=[f(t)=diff(p(t), t)]]

Euler=[phenomenon='time-dependency of ang_velocity',
       entities=[b=rigid_body],
       variables=[Omega(t)=b[ang_velocity(t)],
                  M(t)=b[ang_momentum(t)],
                  T(t)=b[net_torque(t)]],
       assumptions=[ ],
       equations=[add(diff(M(t), t),
                  crossprod(Omega(t), M(t))) = T(t)]
```

The entity and variable names of the model fragments are instantiated as those of the problem and they are substituted in the equations of the model fragments. The angular momentum is derived from principal moments of inertia and angular velocity by the if_needed procedure in the ang_momentum slot. Likewise, the net torque is derived from force and position vector of the point at which the force acts.

$$M(t) = \begin{pmatrix} I_1\Omega_1(t) \\ I_2\Omega_2(t) \\ I_3\Omega_3(t) \end{pmatrix}, \quad T(t) = \sum r \times f(t)$$

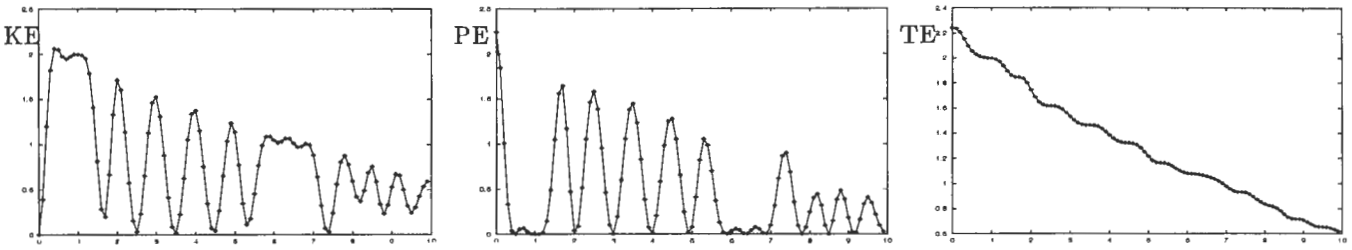


Figure 6: The kinetic, potential, and total energy of the single spring-block system as functions of time during the simulation.

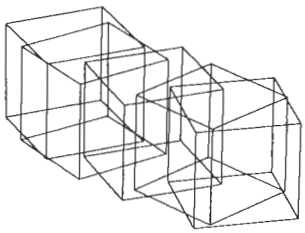


Figure 7: Motion of the block b1. Spring not shown.

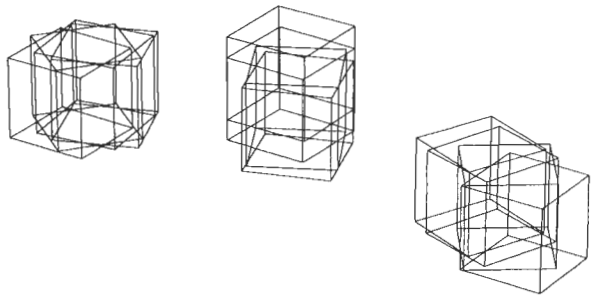


Figure 8: Motion of the multiple spring-block system in Figure 2. Springs not shown.

The principal moments of inertia (I_1, I_2, I_3) and the position vector (r) of the spring-attached point can be assigned from the information of the problem description. The angular velocity is one of the state variables asked by the problem, and its functions are derived in DRVD. The value of force $f(t)$, which has computed using a model fragment Springforce2 (shown earlier in section 2.1), is available in the force slot of b1, and substituted in the equations of Newton2 and Euler.

The system has now total 12 equations in component forms (6 from the model fragments and 6 from DRVD) plus 12 initial conditions for 12 unknowns. Several of the differential equations are nonlinear, and when ORACLE attempts to solve the model analytically, it does not find a solution in closed form. ORACLE then solves the differential equations by numeric simulation. ORACLE displays the simulation result by showing the state variables as functions of time using gnuplot (Figure 5). Animation of the moving block is shown (Figure 7) using PADL-2 solid modeling system [Hartquist, 1983]. Note that the motion of the block is much more complex than that of the linear harmonic oscillator. The kinetic energy, potential energy, and total energy of the system are also displayed as part of validation criteria of the results (Figure 6). The total energy in Figure 6 decreases over time due to the nonzero damping coefficient of the spring of the problem statement.

3 Other Examples

3.1 Multiple Spring-block System

The previous section showed how ORACLE predicts the behavior of the single spring-block system. Can the modeling system of the single spring-block system be used to predict the behavior of the multiple spring-block systems such as Figure 2? The answer is "yes". The multiple spring-block system has additional entities and phenomena, but they are simply the multiple occurrences of the same types as the single spring-block system. Having already enough knowledge represented in general form to handle the single spring-block system, ORACLE can handle the multiple spring-block systems with no change. The way it solves this problem is the same as it does for the single spring-block system. It computes the positions of ends of each spring in the inertial reference frame by transforming the local reference frame of its associated block, computes spring forces acting on the blocks, and derives differential equations for the velocity and angular velocity of the 3 blocks from the procedures attached to the if_needed facets. It then instantiates model fragments Newton2 and Euler, and composes a model. Notice that model fragment sharing occurs within the model because each of those model fragments is instantiated more than once for different entities. The composed model has total 36 equations plus 36 initial conditions for 36 unknowns. The result of the execution of the model indicates that although none of the blocks are initially rotated, the end blocks rotate as well as translate due to spring forces which are not parallel to the radius vectors of the points to which the springs are attached. If the spring damping is ignored (i.e., damping_coeff $= 0$), the middle block shows translational motion only, but it shows both translational and rotational motions if the spring damping is considered (damping_coeff $\neq 0$). Figure 8 shows part of animation scenes for the case with nonzero damping_coeff. In fact ORACLE can handle multiple rigid bodies connected by springs in arbitrary positions and orientations because the way of identifying relevant model fragments and composing them is not restricted by the number of entities or their connections.

3.2 Sailboat

ORACLE can be extended to model a more complex and different type of physical system, a sailboat (Figure 3). A sailboat is a composite object whose driving force comes from the differential motion of air over water. Before we model the sailboat, we can ask the same question as before. Can we use the modeling system of the spring-block systems to predict the behavior of a sailboat in fluids? The answer is "yes", provided that the modeling system has enough domain knowledge to handle the problem. We do not need to build a different modeling system. A modeling system with the same algorithm and the same model fragments plus additional model fragments and entities can predict the behavior of the sailboat.

New classes of entities added to the knowledge base are fluids (water and air) and lifting surfaces (hull and sail). The sailboat, water, and air entity have their own reference frames, which move as their entities move. New phenomena include hydrodynamic and aerodynamic forces, each with two components (lift and drag), and skin friction. A single model fragment is used to represent both hydrodynamic and aerodynamic frictional drag forces, and later instantiated for them. Likewise, a single model is used to represent both hydrodynamic and aerodynamic lift forces.

```
FDrag=[phenomenon='frictional drag force on
        an object in fluid',
    entities=[s=physical_object, f=fluid],
    variables=[FD=s[fdrag(t)],
        v=s[rel_fluid_speed(t)],
        fd=s[rel_fluid_direction(t)],
        Pa=s[parasitic_area],
        rho=f[density]],
    assumptions=[ ],
    equations=[FD=1/2*Pa*rho*v^2*fd]]

Lift=[phenomenon='lift and lift induced force on
        an object in fluid',
    entities=[s=physical_object, f=fluid],
    variables=[LF=s[lift(t)],
        L=s[lift_magnitude(t)],
        v=s[rel_fluid_speed(t)],
        fd=s[rel_fluid_direction(t)],
        pd=s[perpendicular_rel_fluid_dir(t)],
        Ca=s[effective_capture_area],
        rho=f[density]],
    assumptions=[s[rel_fluid_speed(t)] > 0],
    equations=[LF=L*pd+L^2/(2*Ca*rho*v^2)*fd]]
```

Notice that the model fragment Lift has a nonempty assumption slot. There is another model fragment of lift with a different assumption; it says that lift force is zero when the relative fluid speed is zero. After the entities and model fragments are added, ORACLE can solve several types of problems on a sailboat, but we will focus on one type of problem in this section. Suppose that a sailboat is heading in the angle of 49 degrees from the direction of wind at uniform speed 16.9 ft/sec and that water is at rest. The system is asked to compute the sailboat speed which will balance all the forces involved.

ORACLE first infers all the forces on the sailboat from the forces acting on its components, hull and sail. It instantiates the model fragments FDrag and Lift for each of them and records the summation of them in the net_force slot of the sailboat.

$$F = \sum_{i \in \{hull,\ sail\}} (FDrag_i + Lift_i)$$

It then searches for a model fragment which relates forces with speed, and finds the model fragment of Newton2. It substitutes the equations of the forces in the equation of Newton2, $F(t) = d(p(t))/dt$. Since the problem states that all the forces are balanced, the net force on the sailboat must be zero, implying the momentum $p(t)$ is constant. The right hand side of the equation becomes zero from the constant momentum, resulting in an algebraic equation. However, the problem is under constrained in the sense that total number of equations in component form is 2 (Fx=0, Fy=0, Fz becomes a trivial equation 0=0) but the total number of unknowns in the equations is 3 (boat speed, sail lift magnitude, and hull lift magnitude). ORACLE prints the situation, asking for further information. The user provides an additional equation, $\partial(Fx)/\partial(sail\_lift\_magnitude) = 0$, by making a simplifying assumption that the sail is controlled as to maximize the sailboat force in the direction of boat heading. The equations are solved algebraically, producing a solution, boat speed = 15.7 ft/sec.

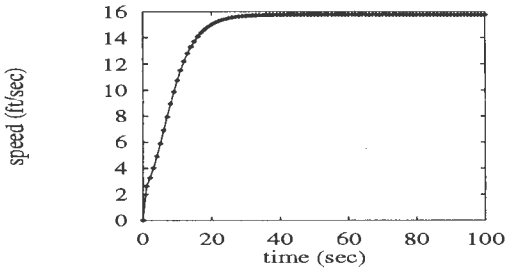


Figure 9: The sailboat speed as a function of time.

The previous example showed how ORACLE composes a model to compute the sailboat speed at a fixed point in time, the equilibrium state of forces in that case. If we are interested not only in such a speed but also in how the boat arrives at the speed, starting from zero speed, the boat speed must be computed as a function of time. Relevant model fragments are retrieved and instantiated in a similar way. In this case, however, the net force on the sailboat is not necessarily zero all the time because the boat accelerates until it reaches the equilibrium state of forces. Therefore, the right hand side of the equation of Newton2 does not become zero, but stays as $d(p(t))/dt$. Since $p(t) = d(m \cdot v(t))/dt$, ORACLE solves the differential equation, $F(t) = d(m \cdot v(t))/dt$ for $v(t)$ by numeric simulation. A plot of the simulation result in Figure 9 shows that the sailboat ultimately accelerates to the same speed as the one predicted by the algebraic method, thus confirming the algebraic solution. Also notice that the model fragment Newton2 used for modeling the spring-block systems is reused for modeling the sailboat and that model fragments Lift and FDrag are shared by hull and sail.

4 Related Work

Falkenhainer and Forbus [1991] describe a form of compositional modeling in which a model is generated by composing model fragments which are initially obtained by matching the terms of a query to a domain theory and then elaborated later. While a composite object in ORACLE can consist of any heterogeneous parts, they require the existence of a unique minimal covering of parts taken from a single part-of hierarchy to generate a simplest possible model. They focus on modeling thermodynamics and do not have a capability of handling detailed structural relations among parts and choosing appropriate reference frames for parts.

The SIGMA system [Keller and Rimon, 1992] is a tool which aids a scientist-user in building a model. After an interaction with the user, it produces a model specified in data flow graph and executes the model to compute a unknown quantity. Like ORACLE, SIGMA represents domain knowledge in frame. But it is a user-assistant system rather than an autonomous model-building system, and has several restrictions in constructing and executing a model, which ORACLE does not have. For example, multiple quantities cannot be computed simultaneously, and model fragments cannot be put together in an arbitrary order due to the strict backchaining control strategy of its model building process. It converts the input values into a common, consistent set of scientific units, but does not have a provision to transform a vector quantity measured in one reference frame to another, which is necessary in dealing with moving objects.

Nayak [1992] describes a method to construct a device model by selecting an appropriate model for each component of the device using structural, behavioral, and expected behavioral constraints. In his system, a model is formulated by composing a set of model fragments, as in ours. However, the uses of the models produced by the two systems are different. While ORACLE constructs a model to predict motions of physical systems, his system builds a model to explain causal relations between parameters of a device. Another difference is that he uses order of magnitude reasoning for behavior generation while we use numeric simulation. His order of magnitude reasoning method is restricted to generating the behavior at a fixed point in time, but we can predict the behavior changing with time as well as the behavior at a fixed point.

The MSG system developed by Ling *et al.* [1993] constructs a model for heat transfer, automatically generating partial differential equations from geometric models. Yip [1993] describes a method to formulate an approximate model from a given detailed model based on the theory of asymptotic order of magnitude. He simplifies a model by examining the limiting cases where the model becomes singular. The IDEAL system [Falkenhainer, 1993] is similar to that of [Yip, 1993] in the sense that it derives a simplified model from a given detailed model. But they differ in that IDEAL uses two approximation operators (what he calls dominance-reduction and iso-reduction) instead of order of magnitude reasoning and produces each simplified model's credibility domain as well, which specifies the range of model parameter values for a given error tolerance.

Another relevant line of work concerns model selection rather than model generation. The framework of Addanki *et al.* [1991] facilitates the selection of an appropriate model from alternative models, which are generated *a priori* and organized in a graph. Weld [1992] provides a more general approach to model selection by reasoning about model accuracy. Ellman *et al.* [1993] introduces gradient magnitude model selection to guide model selection in the sailboat design problem.

Yet another related works concern simulation generation instead of model generation. The SIMLAB system [Palmer and Cremer, 1991] produces a simulator from a user-provided physics model. Given a mathematical model of a physical phenomenon and instructions for solving the resulting equations, SIMLAB transforms the model into an executable simulation code to analyze the phenomenon. However, the user still has the burden of creating the mathematical model. The program built by Berkooz *et al.* [1992] is similar to SIMLAB. It is basically a compiler for translating differential equations expressed in mathematical and programming constructs into an executable code. The SINAPSE system [Kant, 1992] also automatically transforms a given model into a program in desired language, though again the human user must create the input model.

A number of mechanical device simulators are commercially available, such as ADAMS [Dawson, 1985], and DADS [Haug, 1989]. These programs, like most simulators, incorporate physics knowledge such as Newton's laws of motion directly into algorithms rather than representing them explicitly. The simulators include powerful algorithms for forming and solving the equations of motions for a wide variety of mechanisms, but lack the flexibility that ORACLE has to explicitly instantiate general model fragments in particular situations.

Previous AI research in spatial reasoning about mechanical devices [Faltings, 1987; Gelsey, 1989; Gelsey, 1990; Joskowicz and Sacks, 1991] has devoted considerable attention to reasoning about contacts between solid bodies, a problem ORACLE does not presently address. Like the commercial simulators, these programs incorporate knowledge of physical phenomena directly into algorithms rather than attempting to explicitly instantiate general model fragments in particular situations, as ORACLE does.

5 Conclusion and Future Work

Automating the reasoning process about physical systems with multiple moving components in arbitrary configurations is difficult because it requires significant spatial reasoning. We have presented a method and an implemented system for automatically generating and simulating models of such systems from first principles. Evidence of the generality of our approach across different types of physical systems was demonstrated by the experimental results of testing it on the spring-block systems in a variety of configurations and the sailboats in fluids. ORACLE can also model many other types of physical systems with no or minor changes, including multiple rigid bodies connected by springs, propeller-

driven airplanes, and spinning balls. This applicability to broad class of physical systems is possible because knowledge is represented in general form so that common domain theories can be reused and shared.

There are several directions in which this work can be extended. Adding more model fragments and entities would expand the types of physical systems covered by ORACLE. It would also be a valuable test for the extensibility of the system. Another extension is possible by having ORACLE suggest possible directions from reasoning about equations and unknowns when something goes wrong during problem solving Some spatial reasoning problems can be solved by qualitative interpretation of the quantitative models produced by ORACLE. For example, qualitative description of motions (such as translational, rotational, oscillatory, or tumbling) can be easily obtained by postprocessing the simulation results of the models. Coverage of space of a moving object, any regularity of the coverage over time (such as monotonically decreasing coverage of a damped spring), or possible contact/collision with other moving objects (intersection of the coverages over same time intervals) can also be produced by postprocessing the simulation results.

References

[Addanki *et al.*, 1991] S. Addanki, R. Cremonini, and J. S. Penberty. Graphs of Models. *Artificial Intelligence*, 51:145–177, 1991.

[Berkooz *et al.*, 1992] G. Berkooz, P. Chew, J. Cremer, R. Palmer, and R. Zippel. Generating spectral method solvers for partial differential equations. Technical Report 92-1308, Cornell University, 1992.

[Char *et al.*, 1991] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *MapleV Language Reference Manual.* Springer-Verlag, New York, 1991.

[Dawson, 1985] G. Dawson. The Dynamic Duo: Dram and Adams. *Computers in Mechanical Engineering*, March 1985.

[Ellman *et al.*, 1993] T. Ellman, J. Keane, and M. Schwabacher. Intelligent Model Selection for Hill-climbing Search in Computer-Aided Design. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 594–599, 1993.

[Falkenhainer and Forbus, 1991] B. Falkenhainer and K. D. Forbus. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

[Falkenhainer, 1993] B. Falkenhainer. Ideal physical systems. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 600–605, 1993.

[Faltings, 1987] B. Faltings. *Qualitative Place Vocabularies For Mechanisms in Configuration Space.* PhD thesis, University of Illinois at Urbana-Champaign, July 1987.

[Forbus, 1984] K. D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24:85–168, 1984.

[Gelsey, 1989] A. Gelsey. Automated Physical Modeling. In *Proc. of the 11th International Joint Conference on Artificial Intelligence*, pages 1225–1230, 1989.

[Gelsey, 1990] A. Gelsey. *Automated Reasoning about Machines.* PhD thesis, Yale University, 1990. YALEU/CSD/RR#785.

[Hartquist, 1983] G. Hartquist. Public PADL-2. *IEEE Computer Graphics and Applications*, pages 30–31, October 1983.

[Haug, 1989] E. J. Haug. *Computer Aided Kinematics and Dynamics of Mechanical Systems, Volume 1: Basic Methods.* Allyn and Bacon, Boston, etc., 1989.

[Joskowicz and Sacks, 1991] L. Joskowicz and E. P. Sacks. Computational Kinematics. *Artificial Intelligence*, 51:381 – 416, 1991.

[Kant, 1992] E. Kant. Code synthesis for mathematical modeling. In *Working Notes of AAAI Fall Symposium on Intelligent Scientific Computation*, pages 54–59, 1992.

[Keller and Rimon, 1992] R. M. Keller and M. Rimon. A Knowledge-based Software Development Environment for Scient ific Model-Building. In *Proc. of the 7th Knowledge-Based Software Engineering Conference*, 1992.

[Kuipers, 1986] B. Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29:289–388, 1986.

[Ling *et al.*, 1993] S. R. Ling, L. Steinberg, and Y. Jaluria. MSG: A Computer System for Automated Modeling of Heat Transfer. *Artificial Intelligence for Engineering Design, Analysis and manufacturing*, 7(4):287–300, 1993.

[Minsky, 1975] M. Minsky. A Framework for Representing Knowledge. In *The Psychology of Computer Vision*, pages 211 – 277. McGraw-Hill, New York, 1975.

[Nayak, 1992] P. P. Nayak. *Automated Modeling of Physical Systems.* PhD thesis, Stanford University, 1992. STAN-CS-92-1443.

[Palmer and Cremer, 1991] R. S. Palmer and J. F. Cremer. SIMLAB: Automatically creating physical systems. Technical Report 91-1246, Cornell University, 1991.

[Struss, 1988] P. Struss. Global Filters for Qualitative Behaviors. In *Proc. of the 7th National Conference on Artificial Intelligence*, pages 275 – 279, 1988.

[Weld, 1988] D. S. Weld. Comparative Analysis. *Artificial Intelligence*, 36:333–374, 1988.

[Weld, 1992] D. S. Weld. Reasoning about model accuracy. *Artificial Intelligence*, 56:255–300, 1992.

[Williams, 1986] B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proc. of the 5th National Conference on Artificial Intelligence*, pages 105 – 112, 1986.

[Yip, 1993] K. M. Yip. Model Simplification by Asymptotic Order of Magnitude Reasoning. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 634–640, 1993.

How to Automatically Generate an Inference Engine from Declarative Specifications

Bruno Ginoux
Electricité de France
Direction des Etudes et Recherches
1, Avenue du Général de Gaulle. 92141 Clamart. France

Abstract

In order to reconcile the very high-level language advantages: abstraction, concision, readability and declarativity, with the run-time efficiency of low-level languages, we have developed a system which automatically generates an optimized imperative target program from a program specification which is expressed in a very high-level declarative language.
This system, called Descartes, is an expert system for programming. It is implemented as a knowledge-based system and supports a transformational methodology of specification refinement. It has been developed at the Electricité de France R&D Center for the last three years.
In this paper, we present one of the sub-systems of Descartes, called Cogito, which generates an imperative algorithm from the Descartes specification. Our aim is not to give a comprehensive description of the Cogito system. It is rather to show it at work on a concrete example. We have chosen to take as an example, the generation of an inference engine. This paper describes the process of generation, starting from the specification of the engine in the Descartes language, and explaining the successive transformations made by Cogito until an optimized imperative algorithm is obtained.

1 Introduction

The Cogito system is an expert system for programming. More precisely, it is an expert in algorithm design. It is written as a knowledge-based system and supports a transformational methodology of specification refinement (see for example [Barstow, 1979]). The system is fully automatic. Its input is a high-level, abstract and declarative program specification. Its output is an imperative computation strategy written as an algorithm. This algorithm, which is still abstract, (independent of any target language) is then progressively transformed into a target program by the Ergo and Sum systems with which Cogito cooperates within a system called Descartes.

The Cogito language is a formal language which uses a mathematical syntax and provides the two basic concepts of sets and functions. Data are specified by means of mathematical definitions of sets which refer to a conceptual description of data expressed in a semantic data model. Processes which apply to these data are in turn described by the defining of mathematical functions operating on these sets of data.

Cogito is not dedicated to any particular application domain. The system has already been used in various domains such as management and industrial applications, e.g., customer management and scheduling loading and unloading operations in nuclear plant cores.

In this paper, our aim is not to exhaustively describe the Cogito system (see [Ginoux, 1991]). It is rather to show Cogito at work on a concrete example. Therefore, we will just recall when necessary along the paper the main characteristics of the system.

As an example, we have chosen to present the generation of an inference engine. In Section 2, we describe the conceptual data scheme as well as the specification of the inference engine in the Descartes language. Section 3 presents the transformations that Cogito performs in order to get an efficient algorithm.

2 The Specification

We want to specify a simplified inference engine for production rules without variables. This inference engine is simplified because all the premises have the form: entity = value, alll the actions have the form: entity := value and no order is imposed on the actions of a rule.

The rule base is seen as a circular sequence of rules. At each step, the engine starts from the last fired rule, looks for the next firable rule in the sequence and fires this rule except if it has come back to the last rule having modified the working memory. In this later case, the inference cycle is over.

The conceptual scheme of the application domain is given in Figure 1. Elementary sets and functions can be derived from this scheme. Indeed, entities and relationships occuring in the scheme are seen as elementary sets while elementary functions come from attributes and from relationships. We will use the following entity sets: **RULES**, **PREMISES**,

ACTIONS, **ENTITIES** and **TIME** which obviously come from the corresponding entities.

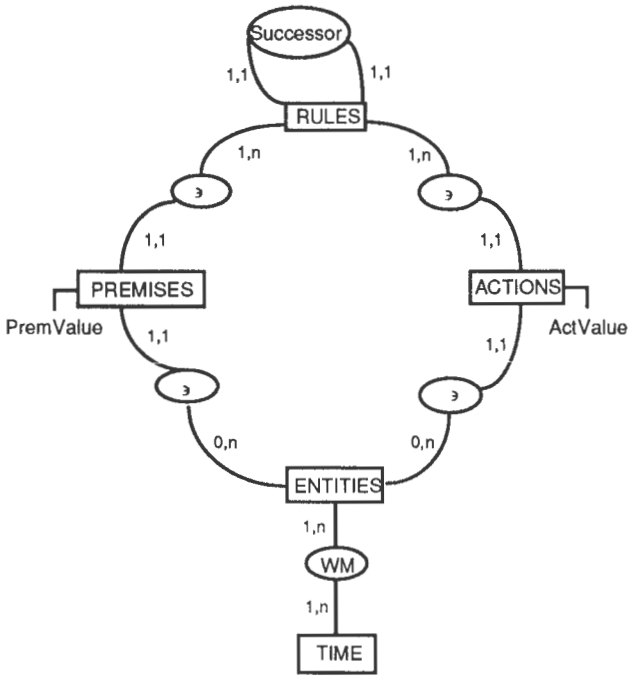


Figure 1

We 'll use also the following functions:

1 Derived from attributes:

- **PremValue**: PREMISES ---> VALUES, which, for any premise, gives the value to which the entity is compared in the premise.
- **ActValue**: ACTIONS ---> VALUES, which, for any action, gives the value assigned to the entity mentioned in the action.

2 Derived from relationships:

- **PremEntity**: PREMISES ---> ENTITIES, which, for any premise, gives the entity concerned.
- **ActEntity**: ACTIONS ---> ENTITIES, which, for any action, gives the entity concerned.
- **Premises**: RULES ---> $\mathcal{P}$(PREMISES), which, for any rule, gives the set of its premises.
- **Actions**: RULES ---> $\mathcal{P}$(ACTIONS), which, for any rule, gives the set of its actions.

Once these elementary sets and operators have been derived from the conceptual scheme, **the basic vocabulary is available to define more complex concepts**. Here, the user will define the following functions which correspond to the specification of the inference engine:

Satisfied: This function takes in input a given premise "p" and a given instant "n", and gives "true" if "p" is satisfied at "n" and "false" otherwise. "p" is satisfied if the value to which the entity is compared in the premise "p" is equal to the value of the entity in the working memory at the previous instant.

. **Satisfied (p,n)** : PREMISES × TIME → $\mathbb{B}$
PremValue(p) = ValInWM(PremEntity(p), n-1)

Firable: This function takes as input a rule "r" and an instant "n", and gives "true" if the rule "r" is firable at the instant "n" and "false" otherwise. A rule is firable if all its premises are satisfied.

. **Firable (r,n)** : RULES × TIME → $\mathbb{B}$
And (Satisfied(p,n))
∀p ∈ Premises(r)

Candidate: This function takes as input two rules "r1", "r2" and an instant "n", and gives the first firable rule located between "r1" and "r2" in the rule base (which is a circular sequence of rules). The order on rules is defined by the relationship "Successor" in the conceptual scheme. If there is no firable rule between "r1" and "r2", the function returns ⊥.

. **Candidate (r1, r2, n)**: $RULES^2 \times TIME \rightarrow RULES \cup \{\perp\}$
If $r_1 = r_2$ **Then** ⊥
Else **If** Firable(r_1, n) **Then** r_1
Else Candidate(Successor(r_1), r_2, n)

Fired: This function takes as input an instant "n", and gives the rule which has been fired at this instant. The rule to be fired is the first firable rule between the rule which has been fired at the previous instant and the last rule which has modified the working memory. The (RULES, Successor) tuple corresponds to the sequence of rules of the RULES set ordered by the "Successor" relationship.

. **Fired (n)**: TIME → RULES ∪ {⊥}
If n = 0 **Then** Last (RULES, Successor)
Else Candidate(Successor(Fired(n-1)), LastModRule(n-1), n)

LastModRule : This function takes as input an instant "n", and gives the last rule having modified the working memory. This rule is either the rule fired at the instant "n" (this is the case if at least one action of this rule modifies its entity) or the last modifying rule at the previous instant. This function allows to stop the process of looking for the next firable rule. Indeed, it is useless to search a new firable rule as soon as the last modifying rule has been reached because it is obvious that even if a rule is still firable, it will not change the content of the working memory.

. **LastModRule (n)**: TIME → RULES
If n = 0 **Then** Last (RULES, Successor)
Else **If** Fired (n) ≠ ⊥ and ∃ a ∈ Actions(Fired(n))/
ModAction(ActEntity(a),a,n)
Then Fired (n)
Else LastModRule(n-1)

ModAction: This function takes as input a given entity "e", a given action "a" and a given instant "n", and gives "true" if the action "a" modifies the value of "e" at "n" and "false" otherwise. "a" modifies the value of "e" if "e" is the entity mentioned in the action "a" and if the value of the entity in the action "a" is different to the value of the entity in the working memory at the previous instant.

. **ModAction (e,a,n)**: ENTITIES × ACTIONS × TIME → $\mathbb{B}$
ActEntity(a) = e and ActValue(a) ≠ ValInWM(e, n-1)

ValInWM : This function takes as input an entity "e" and an instant "n", and gives the value of this entity in the working memory at this instant. This value is either the value mentioned in one of the actions of the rule fired at "n" (if an action of this rule modifies the entity) or the value of the entity in the working memory at the previous instant.

. **ValInWM(e,n)**: ENTITIES X TIME → VALUES
If n = 0 **Then** InitialValue (e)
Else
If Fired (n) ≠ ⊥ and ∃ a ∈ Actions(Fired(n)) / ModAction(e, a, n)
Then ActValue(a)
Else ValInWM (e, n-1)

All these functions represent pieces of knowledge. This knowledge is expressed in a declarative way and is independant of any specific problem. This is the declarative part of the specification.

But a Cogito specification has a second part which corresponds to a computation statement. This statement indicates which function is to be computed as well as its computation domain. This is the imperative part of the specification, and this part is reduced to one instruction. The problem is to get the working memory at the instant when the execution stops, that is when there is no firable rule between the last fired rule and the last rule having modified the working memory. Thus, the computation statement is the following:

COMPUTE R = { ValInWM (e,k)/ e ∈ ENTITIES and k = MU n/ Fired(n) = ⊥, n ∈ $\mathbb{N}$ }[1]

3 The Transformations

3.1 Preliminary analysis

First, this specification is processed by a parser. Functions, as well as the computation domain, are represented as syntactic trees. For example, the"ValInWM" function has the following internal representation:

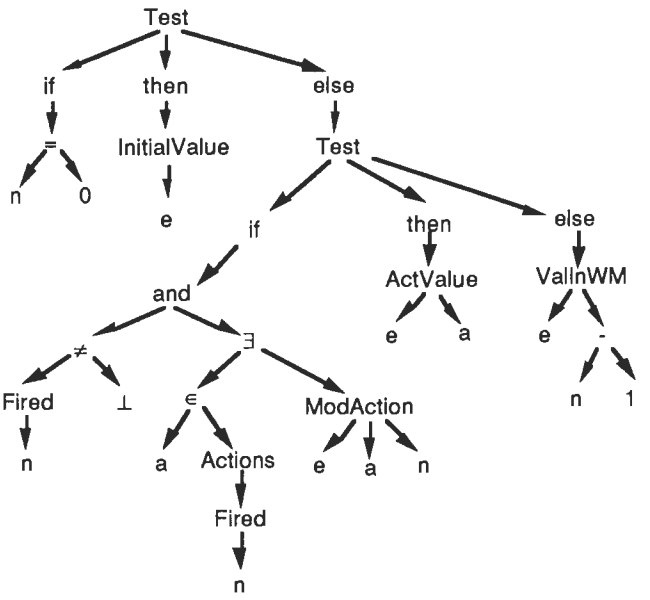


Figure 2

3.2 Unfolding

The first transformation consists of unfolding user-defined functions. Unfolding means replacing a function call by the body of the function properly instantiated by the parameters of the call. Here, this means replacing a node identified as a user-defined function by the (instantiated) tree representing the function. This transformation, well-known and often used in program synthesis ([Burstall and Darlington, 1977]) is correctness-preserving. Moreover, and this is the reason for unfolding, it makes simplifications appear, which will improve the efficiency of the generated program.
The starting tree for unfolding is the tree of the computation statement. The unfolding process is iterated as long as this tree contains user-defined functions. Obviously, recursive functions are not unfolded. A recursive call is kept in the tree and a specific subroutine is created to deal with the computation of this function. So, recursive functions must have been detected prior to the unfolding process.

3.3 Identification of recursive functions

First, the system classifies the functions in three categories : the functions which are sure to be considered as recursive in the generated program, the functions which are sure to be non recursive and the other functions. A function which calls itself inside its own body is sure to be recursive. A function for which there exists no path between it and itself in the graph of function calls is sure to be non recursive and therefore can be unfolded. The other functions are those which call themselves indirectly, through

[1] where "k = MUn / Fired(n) = ⊥" indicates that k is the smallest integer such as Fired(k)=⊥.

a sequence of function calls. These functions are recursive in the mathematical sense but some of them can be unfolded. For example, if f and g call them each other but don't call themselves directly, f, for instance, can be unfolded in the body of g. So, f disappears and g calls itself in its own body and therefore belongs now to the first category. The choice of keeping f or g is not important except if one of the function is the function to compute which must not be unfolded.

By using this technique, the graph of function calls between recursive functions can be simplified until it remains only recursive functions of the first category as well as the function to compute whatever it is recursive or not. In our example, the starting graph is:

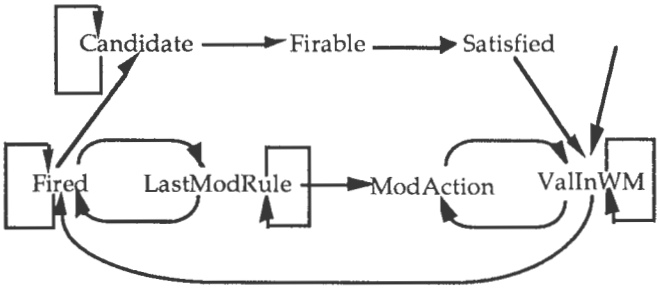


Figure 3

The "Fired", "Candidate", "LastModRule" and "ValInWM" functions are recursive because they call themselves. Moreover, there exists a path of function calls from the three other functions to themselves. So, there are no non-recursive function to unfold. However, by using the principle exposed above, some functions can be unfolded. By unfolding "Satisfied" inside "Firable" and then "Firable" inside "Candidate", the "Satisfied" and "Firable" functions disappear. In the same way, unfolding "ModAction" inside "ValInWM" and inside "LastModRule" makes the "ModAction" function disappear. So, we get:

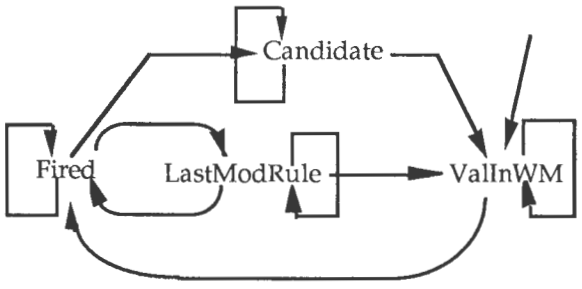


Figure 4

3.4 Simplifications

Syntactic simplifications consists of identifying identical sub-expressions in order to compute them only once. If we consider for example the "ValInWM" function in which the call to "ModAction" has been unfolded, we get:

. **ValInWM (e,n)**: ENTITIES X TIME → VALUES
If n = 0 **Then** InitialValue (e)
Else If Fired (n) ≠ ⊥ and ∃ a ∈ Actions(Fired(n)) /
ActEntity(a) = e and ActValue(a) ≠ ValInWM(e, n-1)
Then ActValue (a)
Else ValInWM (e, n-1)

Each of the sub-expressions: "Fired(n)", "ActValue(a)" and "ValInWM (e, n-1)" is used twice. It would be unefficient to compute them twice since we are in a side-effect-free context, which ensures that two identical sub-expressions yield the same result. In terms of syntactic tree, this means looking for identical nodes and, each time any two nodes are proved identical, deleting one of them and reporting the entering arcs into the node which is kept.

But Cogito can also perform semantic simplifications. This second kind of transformations consists for example of proving that two sub-expressions which do not have the same computation formula (which denote differently) are identical. Very often, this is done by reasoning about the cardinalities of the functions. Other semantic simplifications concern the handling of sets, for example, the optimization of cartesian products. However, we are not going to detail here these aspects (see [Ginoux, 1991]) because our example is not directly concerned by these transformations.

3.5 Recursion removal

Finding techniques for removing recursion has long been a significant research interest. In the domain of recursion removal, Cogito 's knowledge is based on the use of a catalog of *clichés*, each *cliché* corresponding to a programming paradigm ([Burstall and Darlington, 1976], [Garijo, 1978]). More precisely, a *cliché* is an algorithmical pattern allowing to iteratively compute a functional recursive pattern without requiring any stack, but a fixed size array.

There are three main reasons for using *clichés* to remove recursion: it easily takes into account the simple cases, it produces optimized algorithms, and the system can be extended in an incremental way.

Here, we have four recursive functions which are (for clarity reasons, F, G, H and K are not detailed) :

- Fired(n) = F(Fired(n-1),Candidate(r1,r2,n), LastModRule(n-1))
- LastModRule(n) = G(LastModRule(n-1),Fired(n), ValInWM(e,n-1))
- ValInWM(e,n) = H(ValInWM(e,n-1),Fired(n))
- Candidate(r1,r2,n) = K(Candidate(Successor(r1), r2,n),ValInWM(e,n-1))

To deal with the first three functions, the system uses the same *cliché*, which is concerned with a

particular polyadical recursive function pattern. This pattern is: f(x) = **If** a(x) **Then** b(x) **Else** h(f(θ_1(x)),....,f(θ_k(x)), g(x)), where a(x) is the stop condition, b(x) the function giving the result when the stop condition is reached, and the θ_i the functions which allow to decrease the argument until it reaches a value which satisfies the stop condition.

There are two preconditions in order to apply the *cliché*. Firstly, the initialization points (the values which stop the recursion) must be known. This is the case if the a(x) function only mentions constants. Secondly, the θ_i functions must be inversible and their inverse must be known.

Our three recursive functions: "Fired", "LastModRule" and "ValInWM", all fulfill these preconditions. For example:

```
. LastModRule (n):    TIME → RULES
   If n = 0 Then Last (RULES, Successor)
   Else   If Fired (n) ≠ ⊥ and ∃ a ∈ Actions(Fired(n))/
                ActValue(a) ≠ ValInWM(ActEntity(a), n-1))
          Then Fired (n)
          Else LastModRule(n-1)
```

Because of the unique recursive call "LastModRule(n-1)", the θ function is: θ = λn. n - 1. Using its knowledge in the field of mathematics, the system knows that the inverse of this θ function is λn. n + 1. Besides, the initialization point is known because of the "**If** n=0 **Then** Last(RULES, Successor)" statement.

So, the *cliché* is applicable. The principle of the iterative pattern corresponding to the cliché consists of starting from the initialization point(s) and then of climbing to the initial argument by applying the θ_i^{-1} functions inside a *repeat* loop.
The pattern uses also an array containing at each instant all the values required to compute the next one. The values contained in this array are shifted from right to left at each step of the iteration in order to save memory space.

In our particular case, the array, called T_{LMR}, must maintain one value since the computation of the next value only needs the previous value. So, there is one value to initialize in the array before the *repeat* loop can begin. We know, for this initialization point, the value of the "LastModRule" function. Besides, the incrementation of the argument at each step of the iteration is one (k := k + 1) because the θ^{-1} function is λn. n + 1.
The "Fired" and "ValInWM" functions can also be computed by using this *cliché*. This requires to create two other sub-routines working on two other arrays: T_F for the Fired function and T_{WM} for the ValInWM function. It must be noticed that T_{WM} must take into account all the entities. Therefore, it must be a two-dimensioned array.

```
Begin    /* Computation of LastModRule */
  If n = 0 Then result := Last(RULES, Successor)
  Else
    T_LMR(0) := Last(RULES, Successor)
    /* initialisation of an associated array */
    k := 0
    Repeat
      k := k + 1            /*  k := θ^-1(k) */
      T_LMR(1) := If Fired (k) ≠ ⊥ and ....
      /* computation of LastModRule(k) where
         recursive calls to LastModRule (k - 1)
         are replaced by T_LMR(0) */
      ForEach i From 0 To 0 Do
        T_LMR(i) := T_LMR(i+1)
          /* shifting of the values in the
          associated array */
      EndFor
    Until k = n
    result := T_LMR(1)
  EndIf
End
```

Now, let us see how to deal with crossed-recursion. Indeed, as we previously saw in Figure 4, the graph of function calls that Cogito builds, shows that there exists a path from each function to each other. Therefore, these functions are recursively crossed.
These functions must consequently be computed simultaneously, that is, in the same loop. In fact, this is possible because each function is to be computed on the same argument, starts from the same initialization point (n=0), and depends on the same θ = λn. n - 1 function. It is then possible to make a loop fusion in order to compute the functions in a unique *repeat* loop. It must be noticed that the "Candidate" function can't be merged inside this loop because its argument, on which are made the recursive calls, is different. A separate sub-routine remains necessary.

The problem to solve consists of ordering the computation of these three crossed-recursive functions. Indeed, there are mandatory precedences to respect. In order to determine the order of computation, Cogito builds a new graph by means of the "preceeds" relation defined by the following principle [Arsac, 1983] :

f preceeds g ⇔ **f(x) appears in g(x)** (1) **or** **g(x-1) appears in f(x)** (2)

(1) deals with mandatory precedences. This is the case, for example, between the Fired and the LastModRule functions. (2) allows, in given contexts, to choose an order which saves memory space.

By applying this principle on our example, we get the following graph:

Figure 5

Here, there is no conflict (no cycles) in the precedence graph. Furthermore, if conflicts had occured, they could have been solved because the computation precedence between two functions can never be mandatory in the two directions, except if the functions are not well-defined and are in fact uncomputable. A non-mandatory precedence can always be replaced by a storage of the useful previous value in a data structure. This is, in fact, the way to solve cycles in the graph.

As there is no cycle, the system can choose any total order compatible with the partial order defined by the precedence graph. Moreover, an optimization can be done. Indeed, it is useless to use arrays to store previous values. Variables are sufficient: Since the depth of the recursion is one, there is only one value to store in order to compute the next one. And since there are no cycle in the graph, it is ensured that the new value can always override the old value which cannot be useful anylonger. So, the T_{LMR} array can be replaced by a unique variable: LASTMODRULE, T_F can be replaced by the unique variable: FIRED and T_{WM} which is a two-dimensioned array (defined on ENTITIES X TIME) can be replaced by a one-dimensioned array (on ENTITIES), since the temporal dimension is yet useless. Consequently, it is no longer useful to shift the values inside the arrays at each step of the iteration. This is automatically done when the new value of the variable overrides the old one.

If we don't explicit the "Fired", LastModRule" and "ValInWM" functions, the result of the loop fusion is an algorithm which looks like:

```
Begin   /* parameter: n */
  | initialization of LASTMODRULE
  | initialization of FIRED
  | initialization of T_WM(e)  ∀ e ∈ ENTITIES
  | .....
  | k := 0
  | Repeat
  |   | k := k + 1
  |   | FIRED := Fired(k)
  |   | LASTMODRULE := LastModRule(k)
  |   | T_WM(e) := ValInWM(e, k)  ∀ e ∈ ENTITIES
  | until k = n
End
```

The Fired function calls the Candidate function. As we previously said, a separate sub-routine is created to compute this function. The generic *cliché* used to remove the recursion deals with tail-recursive functions. Indeed, after the unfolding of the "Firable" and "Satisfied" functions, and granted that the "ValInWM" function is yet represented by a one-dimensioned array (which is a global variable), the definition of Candidate is the following:

```
. Candidate (r1,r2,n): RULES^2 × TIME →  RULES ∪ {⊥}
   If r1 = r2  Then  ⊥
   Else If And ( PremValue(p) = T_WM(PremEntity(p)) )
        ∀ p ∈ Premises(r1)
          Then  r1
          Else  Candidate(Successor(r1), r2 , n)
```

This is a tail-recursive definition. The pattern is: f(x) = **If** a(x) **Then** b(x) **Else** f(θ(x)). The principle of the iterative translation of such functions is well known. It is the inverse of McCarthy's transformation:

```
Begin
  | While not a(x) do
  |   | x := θ(x)
  | EndWhile
  | result := b(x)
End
```

The iterative pattern that Cogito uses is based on the same principle but is, for technical reasons, slightly different. Indeed, the *cliché* that it uses does not require to identify explicitly the a, b and θ functions, which would need to make a costly unification. The *cliché* is the following:

```
Begin
  | Finished := False
  | While not Finished Do
  |   | If a(x) Then
  |   |   | result := b(x)
  |   |   | Finished := True
  |   | Else
  |   |   | x := θ(x)
  |   | EndIf
  | EndWhile
End
```

The interest is that inside the "While" loop, we have the whole body of the function. The unique difference comes from the fact that each time there is a stop condition (i.e., each time there is no recursive call), the system adds an instruction which updates the boolean variable "Finished" and allows to end the loop. It is therefore sufficient, instead of

explicitely identifying the a, b and θ functions, to detect where are the stop conditions and where are the recursive calls, which is very easy. The same method can be used for non tail-recursive functions (see [Ginoux, 1991]).

Therefore, the generated "Candidate" subroutine is the following (the "n" argument is no longer useful):

```
Begin        /* Candidate: parameter: r1, r2 */
  Finished := False
  While not Finished Do
  |   If r1 = r2  Then      result := ⊥
  |                         Finished := True
  |     Else
  |      If And (PremValue(p) = TWM(PremEntity(p)) )
  |        ∀ p ∈ Premises(r1)
  |                         Then result := r1
  |                              Finished := True
  |         Else
  |            r1 := Successor(r1)
  |      EndIf
  |   EndIf
  EndWhile
End
```

Now, the problem which remains to be solved is the determination of the computation domain.

COMPUTE R = { ValInWM (e,k)/ e ∈ ENTITIES and k = MU n/ Fired(n) = ⊥, n ∈ ℕ }

Indeed, the ValInWM function is to be computed at a very particular instant. The system must determine the smallest integer "k" such as Fired(k) = ⊥. The difficulty comes from the fact that the "Fired" function is recursively crossed with the "ValInWM function" which is precisely the function to compute. This means that computing the instant means computing the function to compute. Therefore, an "a priori" computation of the instant is not possible.

However, as we previously said, the ValInWM function, which is recursive, is going to be computed by using an iterative pattern which consists of a loop starting from n=0 and climbing up by applying θ^{-1} = λn. n + 1 until the initial argument is reached. This initial argument is precisely the instant that we are looking for. Granted that the loop increments the successive values of the current instant, it is sure that the first "k" which verifies Fired(k) = ⊥ is the smallest. It is therefore sufficient to take the condition "Fired(k) = ⊥", that is "FIRED = ⊥" as the stop condition of the *repeat* loop.

The algorithm can now be written. Some useless assignments have already been deleted and a syntactic simplification has been made which are not detailed here. Besides, for clarity reasons, the "ModAction" function has not been unfolded:

```
Begin   /* Main */
  |  r := Last(RULES, Successor)
  |  LASTMODRULE:= r
  |  FIRED := r
  |  ForEach e ∈ ENTITIES Do        /* initializations */
  |     T_WM(e)  :=  InitialValue(e)
  |  EndFor
  |  k := 0
  |  Repeat
  |     k := k + 1
  |     /* Computation of Fired(k) */
  |     FIRED := Candidate(Successor(FIRED),LASTMODRULE)
  |     /* Computation of LastModRule(k) */
  |     If FIRED ≠ ⊥ and ∃ a ∈ Actions(FIRED) /
  |     |                    ModAction(ActEntity(a), a)
  |     |  Then   LASTMODRULE:= FIRED
  |     EndIf
  |     /*Computation of ValInWM(e,k) ∀ e ∈ ENTITIES*/
  |     ForEach e ∈ ENTITIES Do
  |       If FIRED ≠ ⊥ and ∃ a ∈ Actions(FIRED) /ModAction(e, a)
  |       |  Then  T_WM(e)  :=  ActValue(a)
  |       EndIf
  |     EndFor
  |  until FIRED = ⊥
End
```

3.6 Current investigations

There remains now some very interesting simplifications to do which are not implemented yet, because they need new sophisticated mechanisms of reasoning.

At first, the "k" variable is no longer useful since the computation inside the *repeat* iteration doesn't need its value. Indeed, there 's no need for the precise value of "time". What is needed is the way to move from an instant to the next one, which is handled by the iteration itself, and the values at the previous instant, which are stored in the data structures. So, the "k " variable can be suppressed.

Second, the two tests on the value of FIRED can be merged because the value of FIRED is independant of the loop on the ENTITIES set. Then, the idea comes down to determining more precisely the entities which are going to be concerned by the "$T_{WM}(e)$:= ActValue(a)" assignment at each step of the *repeat* iteration. In fact, these entities are those for which there exists an action of the fired rule which modifies their values. Consequently, it is absolutely useless to scan the whole ENTITIES set. It is sufficient to consider the sub-set of the entities modified by the actions of the fired rule. Indeed, an action modifies only one entity and an entity is modified by only one action inside a rule.

To go further, we can notice that instead of performing a loop on the subset of the modified entities, it is equivalent to do a loop on the modifying actions and to consider for each action the modified entity. The interest is that this sub-set of actions is also useful for the computation of "LastModRule".

Indeed, the two computations of "LastModRule" and "ValInWM" can be made within the same loop. Moreover, this loop iterates on the sub-set of actions of the fired rule which is generally very small compared to the whole ENTITIES set. So, this would save a large amount of useless computation when compared to the initial algorithm. Indeed, in the initial algorithm, there was a first loop on the actions of the fired rule in order to determine if there was at least one modifying action, and also two nested loops: the first one on the whole ENTITIES set, and the second one on the actions of the fired rule again! The final algorithm is then:

```
Begin   /* Main */
  r := Last(RULES, Successor)
  LASTMODRULE:= r
  FIRED := r
  ForEach e ∈ ENTITIES Do
    T_WM(e) := InitialValue(e)
  EndFor
  Repeat
    FIRED := Candidate(Successor(FIRED),LASTMODRULE)
    If FIRED ≠ ⊥
      ForEach a ∈ Actions(FIRED) /
                        ModAction(ActEntity(a), a) Do
        LASTMODRULE:= FIRED
        T_WM(ActEntity(a)) := ActValue(a)
      EndFor
    EndIf
  until FIRED = ⊥
End
```

The only problem that remains is that the "LASTMODRULE:= FIRED" assignment is performed several times. But obviously, this is not significant compared to the huge gain in efficiency that comes from the optimization of the loops.

Conclusion

This paper aimed at showing how to express a program specification in the Cogito language and how the Cogito system could automatically generate an efficient imperative algorithm, starting from this abstract declarative specification of a concrete non-trivial example

Although the example of the inference engine does not require all the capabilities of the system, it allows to show some of them. In particular, the major transformation module, which is recursion removal, has been described at work. As we showed, removing recursion without using any stack, even for non tail-recursive functions, has deeply improved the generated algorithm. This is something that, as far as we know, classical compilers of functional languages cannot currently do.

Future work includes taking into account the kind of optimizations described in the last paragraph, which are very interesting in terms of efficiency, but require sophisticated knowledge and reasoning power. We also project to raise the level of the input specification. We aim at starting from an actual problem specification as some algorithm designers (for example, KIDS [Smith, 1990]) do. Cogito will therefore have to learn from these systems some knowledge in the field of problem solving.

References

[Arsac, 1983] J. Arsac. *Les bases de la Programmation.* Dunod. Paris. 1983.

[Barstow, 1979] D.R. Barstow. *Knowledge-based program construction.* Elsevier North Holland, New-York. 1979.

[Burstall and Darlington, 1976] R.M. Burstall and J. Darlington. *A system which automatically improves programs.* Acta Inf. 6. pp 41, 60. 1976.

[Burstall and Darlington, 1977] R.M. Burstall and J. Darlington. A *transformation system for developing recursive programs.* J; ACM 24, 1. Janvier. pp 44, 67. 1977.

[Garijo, 1978] F.J. Garijo. *GPFAR 2: un système d'écriture automatique de programmes pour le calcul optimisé des fonctions récursives.* Paris VI. Third cycle thesis. 1978.

[Ginoux, 1991] B. Ginoux. *Génération automatique d'algorithmes par système expert à partir de spécifications déclaratives de très haut niveau: Le système COGITO.* PhD thesis. Paris IX University. 1991.

[Ginoux and Lagrange, 1989a] B. Ginoux and J.P. Lagrange. *An Expert System Approach To Program Synthesis.* AAAI Spring Symposium, Series 1989: Artificial Intelligence and Software Engineering, Standford, Ca. USA. 1989.

[Ginoux and Lagrange, 1989b] B. Ginoux and J.P. Lagrange. *Synthesis of Simple Programs which handle Complex Data.* IJCAI'89 Workshop on Automating Software Design, Detroit, Mich. USA. 1989.

[Kant and Barstow, 1981] E. Kant and D.R. Barstow. *The refinement paradigm : The interaction of coding and efficiency knowledge in program synthesis.* IEEE Trans. Softw. Eng. 7, 458-471. 1981.

[Smith, 1990] D.R. Smith. *KIDS: A Semiautomatic Program Development System.* IEEE transactions on software engineering. Vol. 16. No. 9. Septembre 1990.

[Steier and Anderson, 1989] D.M. Steier and A.P. Anderson. *Algorithm Synthesis : A comparative Study.* Springer - Verlag. New-York. 1989.

Case-Based Reasoning for the Verification and Validation of Complex Devices' Models

Michel P. Féret and J. I. Glasgow

Department of Computing & Information Science,
Queen's University, Kingston,
Ontario, Canada, K7L 3N6*
michelf@bnr.ca, janice@qucis.queensu.ca

Abstract

This paper presents an approach that considers CBR for the verification and validation of knowledge-based systems. It concentrates on model-based diagnostic systems by identifying practical problems with models of complex devices that leads to diagnostic errors. The paper shows that case-based reasoning, used to account for errors in models for complex devices, can be integrated with other diagnosis techniques and applied at different stages of the spiral model for software development.

1 Introduction

While researchers have studied the verification and validation of knowledge based systems (KBSs) [O'L87, OBS87, Gup91], and more specifically of case-based reasoning (CBR) systems [O'L93], they have so far ignored CBR as a tool for verification and validation of knowledge bases. This paper presents an approach that uses CBR as a means to account for errors in complex device models used for diagnostic applications. CBR uses experience to avoid repeating the same mistakes twice, to improve the quality of the successful results and to reach conclusions faster.

This paper presents examples from our previous research in automated diagnosis [FGLJ90, FG91, FG92, FG93], which involved the design and implementation of a generic model-based diagnostic expert system that integrates model-based diagnosis and CBR. This system was based on a new model for diagnosis, called Explanation-Aided Diagnosis (EAD), that accounts for the potential incompleteness and incorrectness of the device models [Fé93]. This system, the Automated Data Management System (ADMS), has been fully implemented and has previously been described and compared to other diagnostic systems in [FG91]. It was applied to two real-world devices, a robotic system called the Fairing Servicing Subsystem, and a Reactor Building Ventilation System [FG91, FG92]. Extensive experimental results showed the effectiveness of the method [Fé93].

This paper focuses on insights into verification and validation acquired during the design and the development of the EAD model. Section 2 provides a brief introduction to model-based diagnosis (MBD) and characterizes verification and validation problems in the context of MBD. Sections 3 and 4 describe the verification problem in the context of MBD and motivates the need for on-line, run-time verification and validation tools. Sections 5 and 6 introduce the CBR paradigm and overview the hybrid CBR component of the ADMS. Section 7 describes how CBR can solve the problems presented in Sections 3 and 4. A final discussion summarizes the contributions of this paper and generalizes them to other application domains. The experimental results shown in Section 7 are extracted from [Fé93]. The contribution of this paper stems from the novel interpretation of these results from the point of view of the verification and validation of knowledge-based systems.

2 Model-Based Diagnosis

The main idea underlying research in model-based diagnosis is that a device model can be used as the basis for diagnosis. These models describe the correct, expected behavior of the device. The search for components in an abnormal state is guided by the discrepancies between what is predicted by the model and what is observed in the device. A diagnostic session is triggered when initial symptoms do not match with the predictions of the model.

Model-based diagnosis is abductive by nature. Abduction is often viewed as inference to the "best" explanation. After abnormal symptoms have been detected, an

*This research was supported through a contract from the Canadian Space Agency (STEAR program), a scholarship and an operating grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada. We also would like to thank Spectrum Engineering Corporation Ltd. (Peterborough, Ontario) and Bell-Northern Research Ltd. (Ottawa, Ontario).

abductive diagnoser constructs one or more explanatory hypotheses that would resolve the anomaly and explain the situation. In model-based diagnosis, this is done by chaining together causal inferences. Abduction in model-based diagnosis consists of "backtracking" from symptoms to components by uncovering the discrepancies between what is observed and what is expected (i.e. what is described by the device model). Abduction is said to "explain away" symptoms.

One problem with model-based diagnosis is the difficulty of implementing models of complex devices. Analyzing and compiling human diagnostic problem-solving capabilities is difficult. Misunderstandings, incorrect specifications, typos, etc. typically lead to partially incorrect models which are difficult to debug, especially when there is no realistic simulation program available for the device. These errors have been described for general knowledge-based systems [O'L93]. They typically result in problems of inconsistency, redundancy, incompleteness, and lack of correctness. Moreover, device models are not always the most natural or efficient representation for diagnosing faulty components [SC85]. These knowledge acquisition, verification and validation problems clearly weaken the reliability of model-based systems such as the ADMS and need to be addressed before the system can be used for critical, real-world applications.

The process of human diagnostic problem-solving is often suboptimal [GS88, YH88]. The following biases are the most relevant causes of suboptimal diagnostic performance:

- Not all possible causes are taken into account.
- Disconfirming evidence is ignored.
- Absence of symptoms is ignored.
- The probability of causes is estimated incorrectly.
- Only confirming actions are performed.

Resulting shortcomings are likely to be found in any model designed and implemented by humans. In our experience, we have found such mistakes in the experts' explanations and reasoning processes. This leads to device models that are either incomplete or inconsistent because they incorporate human limitations. An aggravating factor with these biases is that they occur at the knowledge level and therefore are often only detectable at run-time when diagnostic errors are produced. They represent, to some extent, the worst possible case for knowledge validation.

Because of the problems mentioned above, complex device models need rigorous verification and validation, in order to field systems with usable models. However, given the difficulty of the task and the state of the art in verification and validation, it is also likely that most model-based diagnostic systems will be fielded with an incomplete or incorrect device model. This justified, at the level of models for diagnosis, the development of the EAD model [Fó93]. At the software management level, it justifies the further study of potential verification and validation techniques for device models.

3 Verification for MBD

Verification is the process of ensuring that the knowledge in the system is represented correctly [ABC82]. This includes that the knowledge is consistent, complete and correct. Sometimes, verification also includes issues related to redundancy when multiple versions of the same information are present in the system.

Failure mode descriptions, structural knowledge, symptom specifications, component interactions and/or normal behaviors descriptions can be missing, incomplete or incorrectly described in the device model. In this context, it is difficult to list all possible redundancy, consistency and completeness errors that could be present in device models. However, it is simpler to list diagnostic failures that occur because the model is incomplete, incorrect or inconsistent. In [Fó93], we have characterized diagnostic failures using the EAD model and related them to errors types in the device models.

Consistency errors occur when 1) two failure modes "fire" at the same time for the same component, 2) given a certain set of symptoms, it is impossible that a specific component be faulty and yet is still considered as a potential cause, or 3) a potential diagnosis produced by the system cannot produce the described symptoms.

Completeness errors occur when a diagnosis is overlooked or when descriptions of failure modes or specifications of interactions among components are missing in the device model. Incorrect knowledge can also result in missing diagnoses.

Redundancy errors occur when multiple versions of the same knowledge exist and do not match anymore. In the context of diagnosis, this often happens when the system (or the human) performs hypothetical reasoning and mixes known symptoms with hypothetical ones and hypothetical conclusions with known facts.

The errors mentioned above lead to systems that fail to produce all relevant potential diagnoses, that produce irrelevant and therefore incorrect potential diagnoses, or that produce correct diagnoses (i.e. identify the cause of the problem) but fail to explain them properly (e.g. wrong failure mode).

4 Validation for MBD

Validation is the process which ensures that the system make correct decisions and achieve acceptable performance, usually specified by requirements defined prior to the development of the system. In the context of automated diagnosis, these requirements vary greatly. A

critical application may require a zero diagnostic failure rate. Less critical applications might only accept that the correct diagnosis be produced along with some irrelevant diagnoses. Expert systems used for teaching might insist on the quality of their explanations and not be as stringent about the quality of the diagnostic results themselves. Nevertheless, the overall competence of any diagnostic system increases with the proportion of correct diagnoses relative to incorrect or badly explained ones.

Another problem specific to automated diagnosis is that human reasoning errors often permeate into the design of the system (see Section 2). These errors are only detectable at run time when diagnostic failures occur. The requirements imposed on diagnostic systems often include the need for performing better than human experts, or the need for helping human experts to detect and characterize their own mistakes.

Finally, performance requirements can also include speed requirements. It is usually difficult to estimate how close the resulting expert system will be to the requirements. If the system does not meet them, it is usually difficult to speedup the system without affecting its design and current implementation.

The intrinsic complexity of devices and of device models makes the tasks of verification and validation of the knowledge contained in device models extremely difficult. The previous sections have emphasized that, at least at this point, reaching perfect device models can only be done through a fielded phase. Detection of some errors is only possible during this phase when the model is actually being used for diagnosis. This calls for on-line, run-time verification and validation techniques that account for imperfect device models and, if possible, help the system automatically deal with detected errors. The rest of this paper establishes CBR as a paradigm for such techniques.

5 Case-Based Reasoning

CBR has traditionally been used as a stand-alone problem-solving method. A CBR system stores past experiences in the form of cases. When a new problem arises, the system retrieves the cases most similar to the current problem, then combines and adapts them to derive and criticize a solution. If the solution is not satisfactory, new cases are retrieved to further adapt it in the light of additional constraints (expressed from the non-satisfactory parts of the proposed solution) until it is acceptable. After a problem is solved, a new case can be created and stored in the case base.

Notable CBR systems include MEDIATOR [KS89] and PERSUADER [Syc87] for dispute resolution, JULIA [Kol87, Hin88] and KRITIK [GC89] for design, CHEF [Ham89] for planning, HYPO [AR87] for legal reasoning, CASEY [Kot88], PROTOS [BP87] and CELIA [Red89] for diagnosis, and LADIES [BBGD92] for decision support.

6 CBR and Model-Based Diagnosis

The ADMS - an existing implementation for the EAD model - uses fault models which only specify incorrect behavior modes. These fault models are organized hierarchically along the actual structure of the monitored device. The general diagnostic algorithm described in Section 2 is adapted accordingly to fault models. Each node in the model represents a subpart of the device and is associated with necessary conditions for this subpart to be faulty. If these necessary conditions are not satisfied, the components belonging to the corresponding subpart can be safely pruned away from the search space of potential diagnoses. The algorithm proceeds in a top-down manner through the decomposition hierarchy. At the bottom of the hierarchy, known failure modes are tested against the current symptoms and "fired" in a rule-based fashion. In the ADMS, this abductive algorithm, known as the structural isolation process, is associated with a CBR system. More details can be found in [FG93] and in [Fó93]. Following is a brief description of the relevant features of this hybrid system.

The CBR component of the ADMS is used for critiquing the results of the model-based approach in the light of past experience and provides the human operator with a means for exploring alternative hypotheses. The integration of CBR with the structural isolation process allows for a simple and effective indexing schema as well as a computationally inexpensive similarity measure for cases.

Cases for EAD store past diagnostic scenarios, each consisting of a description of the fault that occurred (fault type, fault time, detecting sensor), the series of pruning steps used to produce a list of potential diagnoses (i.e. the tests performed during diagnosis and their values), the list of potential diagnoses produced by the structural isolation process and the correct diagnosis selected by the operator. A successful case is a case where the correct diagnosis was produced by the structural isolation process and confirmed by the human operator. A failure case is a case where the diagnosis failed to find the correct diagnosis, and for which the operator chose a component that was not in the list of proposed diagnoses.

The structural isolation process can be seen as a rough estimate of the location of a component whose failure explains the observed symptoms. The list of potential diagnoses is used as a means of indexing the case base, leaving the values of the associated sensor functions for the matching step which is a finer judgement of similarity. For either a successful or a failure case, we use

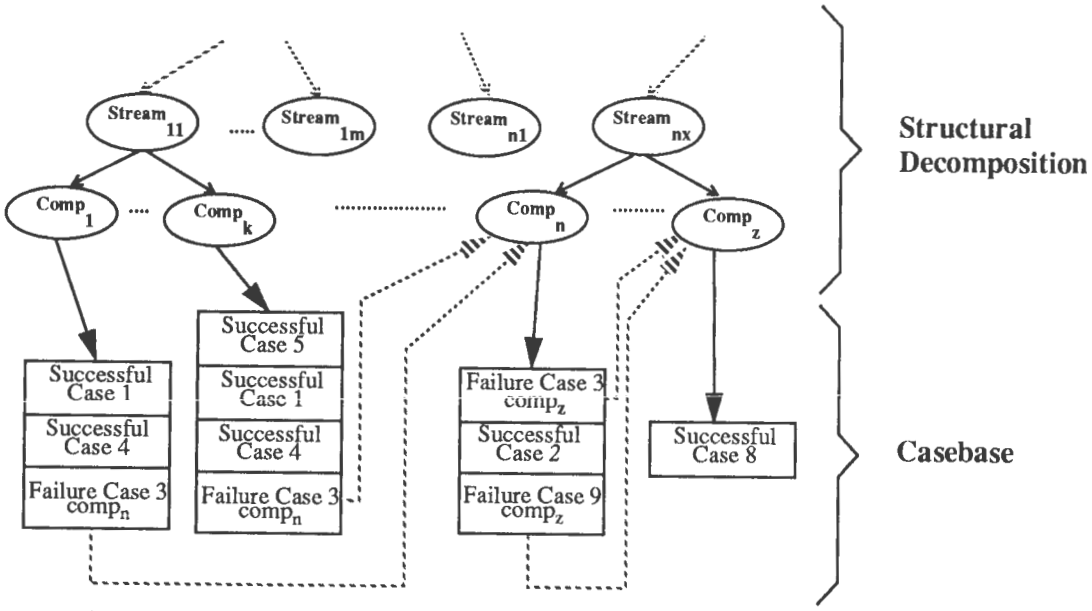


Figure 1: The case base and the structural decomposition.

each potential diagnosis produced by the structural isolation process as an index for the case. Figure 1 illustrates this indexing schema. Each case is stored at the bottom of the structural decomposition under the basic components it contains as potential diagnoses. The dashed arrows originate from failure cases and point to the components representing the correct diagnoses for those cases.

This indexing schema is satisfying because of the interdependencies that exist among "neighboring" components. Such components often share the same characteristics and are likely to appear in each other's lists of potential diagnoses. They will likely share the same cases, or cases that are very similar to each other, except for failure cases. This ensures a useful grouping of similar cases with "bridges" from one grouping to the next provided by failure cases. This indexing schema, based on the information generated by the structural isolation process, is therefore both simple and effective.

Cases are retrieved from the case base to evaluate and criticize the current list of potential diagnoses. This can be done automatically or left to the operators' control. In the latter case, the operator can ask the ADMS to explore its case base and to criticize or confirm a potential diagnosis or to suggest new diagnoses that were not generated by the structural isolation process. If the current potential diagnosis is supported by a previous successful case, the level of confidence in this potential diagnosis can be raised. If the correct diagnosis for the most similar case disagrees with all the suggested diagnoses, and points towards a failure case that matches sufficiently well with the current situation, the validity of the current diagnosis is lowered. The diagnosis stored in the failure case is extracted from the case base and presented to the user as a new potential diagnosis that can, in turn, be evaluated.

Because of the indexing method described above, components at the bottom of the hierarchy serve as pointers to cases that represent diagnostic sessions caused by similar or related failures. The matching is effective because the knowledge contained in those cases is relevant in both the current and the past cases. The matching algorithm is focused on the part of the system that is the most relevant to the current situation.

7 CBR for Verification and Validation

The problems described in Sections 2 to 4 are verification and validation problems. Some of them are general to all knowledge-based systems. Others are typical of model-based diagnostic applications. Not all of them are solvable at design or compile time. This conclusion uncovers the need for run-time techniques that either correct these problems "on the fly" or, at least, make the system account for them. The CBR paradigm provides such techniques.

Verification problems, emerging as completeness, consistency and correctness problems, result in diagnostic systems that output irrelevant diagnoses, diagnoses incorrectly justified, or that fail to produce some relevant diagnoses. The EAD model, presented in [F93], accounts for these problems. The ADMS, an implementation for the EAD model, learns from its mistakes and from its successes. As it gains experience - experience directly

related to the monitored device - its outputs become more focused and of better quality. Extensive experimental results, supporting these claims can be found in [F93] and in [FG93].

Validation problems, emerging at run-time after the system is fielded, are usually due to either human deficiencies that have been incorporated into the system, or to speed requirements that are not being met. CBR, used as a speed up learning tool, can help the system to acquire speed as it gains experience. The work done by Koton on CASEY is a good illustration of the use of CBR as a speedup learning mechanism [Kot88], where CBR is tried first as an attempt to reason from analogy. The cases are directly derived from the MBD system. If the CBR system fails to reach an appropriate solution, the MBD system is call to solve the problem. Its outputs are then stored as a new case in the case base. CASEY attempts to use CBR first, falling back on a traditional search method if CBR fails. The EAD approach uses CBR only *after* the abductive phase is completed. In this sense, the two approaches are opposite. However, they both implement hybrid approaches which combine the advantages of two separate problem-solving methodologies.

8 CBR in the Spiral Development Model

A simulator was built for the Fairing Servicing Subsystem (one of the two devices we applied the ADMS to), capable of simulating correct behavior as well as single faults. The ADMS was applied to the data generated by the simulator with progressively degraded device models. The goal was to see how the CBR component allows the system to learn from its mistakes. We degraded the model by "failing" sensor functions. Sensor functions are functions that extracted a qualitative description of the device from real-time sensor data gathered from the device (or the simulator). Sensor functions were used to encode the necessary conditions and the failure modes used during the structural isolation process. Degraded models are equivalent to imperfect models: the higher the number of failed sensor functions, the less perfect the device model is. The maximum number of failed sensor functions corresponds to a model which is 75% wrong. The failure rate of the system was measured in relation to the case base size for different levels of model degradation (see [F93] for a complete description of the experiment[1]). Figure 2 illustrates a typical graph for such an experiment (we also made other parameters vary, such as the similarity threshold, the matching algorithm, etc).

[1]The experimental setup for this experiment is accessible through anonymous ftp from ftp.qucis.queensu.ca. The code is located in the /pub/feret/archives/exp4.

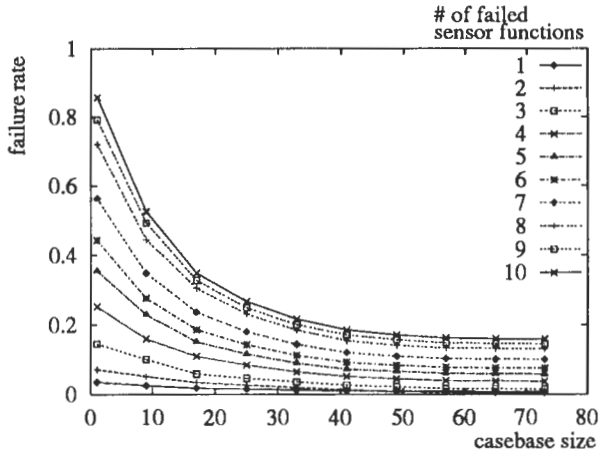


Figure 2: Failure Rates Relative to Case Base Size

The graphs in Figure 2 show that the larger the case base is, the better the diagnostic performance becomes. This suggests that, at least for non critical applications, such a hybrid diagnostic system can indeed be fielded, even though the device model it uses is imperfect. If an error is made, the CBR system stores a case registering the conditions under which it was made. When the same conditions occur, the abductive diagnostic system makes the same mistake but the CBR system recognizes a potential problem and retrieves the case.

The shape of the curves in Figure 2 also suggest that a fielded system could be fine-tuned progressively by knowledge engineers. The process would involve analyzing the cases in the case base, potentially replaying them with a simulator, to isolate the nodes where errors were made. Preliminary research indicates that this process could partly be automated.

For critical systems, the development method described above can still be useful, assuming a simulator is available. Batch tests could be run and resulting cases would provide valuable insights about whether the system is ready for fielding and, if not, where are the remaining problems.

The main advantage of this method is that is allows the systems to make mistakes, to recover and to learn from them. The learning is achieved by mere accumulation of cases which, in turn, can be processed at regular intervals (or after a batch test) by knowledge engineers or by an automated machine learning algorithm.

This method can also be applied at different stages of the development process: a prototype can use the error cases for fine tuning, or for early verification and validation of a prototype model. A fielded system can use it to keep track of its mistakes so that knowledge engineers can periodically evaluate the system (and fine-tune it).

Hybrid CBR systems have been applied to diagnosis

[FG93], to planning [GC92] and to natural language understanding [Car93]. These systems can all learn from their mistakes, even if their CBR component is not perfect (as illustrated in Figure 2, the failure rate tends to level off, indicating the limits of the CBR system).

This paper has established CBR as an on-line, run-time tool for verification and validation of knowledge-based systems. When used in association with an existing problem-solving method (e.g. abductive diagnosis, hierarchical planning), CBR provides a simple and effective way to learn from mistakes. CBR can also be used to learn from success and for speed-up learning.

References

[ABC82] W. Adrion, M. Branstad, and J. Cherniavsky. Validation and verification of computer sotware. *ACM Computing Surveys*, 14(2):159–192, 1982.

[AR87] K.D. Ashley and E.L. Rissland. Compare and contrast: A test of expertise. In *Proceedings of 5th National Conference on Artificial Intelligence*. Morgan-Kaufman, 1987.

[BBGD92] E. Blevis, R. Burke, J. I. Glasgow, and N. Duncan. The life analysis and depreciation integrated exemplar system (ladies). *International Journal of Expert Systems, Special issue on Case-based Reasoning*, 4(2):141–156, 1992.

[BP87] E. R. Bareiss and B. W. Porter. Protos: An examplar-based learning apprentice. In *Proceedings of the 4th International Workshop on Machine Learning*, pages 12–23, June 1987.

[Car93] C. Cardie. A case-based approach to knowledge acquisition for domain-specific sentence analysis. In *Proceedings of 11th National Conference on Artificial Intelligence (AAAI-93)*. AAAI Press/MIT Press, 1993.

[F93] M. P. Féret. *Explanation-Aided Diagnosis: Combining Case-Based and Model-Based Reasoning for the Diagnosis of Complex Devices.* PhD thesis, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, November 1993.

[FG91] M. P. Féret and J. I. Glasgow. Generic diagnosis for mechanical devices. In *Proceedings of the 6th International Conference on Applications of Artificial Intelligence in Engineering*, pages 753–768, Oxford, UK, July 1991. Computational Mechanics Publications, Elsevier Applied Science.

[FG92] M. P. Féret and J. I. Glasgow. Case-based reasoning in model-based diagnosis. In *Proceedings of the 7th International Conference on Applications of Artificial Intelligence in Engineering*, pages 679–692, Waterloo, Canada, July 1992. Computational Mechanics Publications, Elsevier Applied Science.

[FG93] M. P. Féret and J. I. Glasgow. Hybrid case-based reasoning for the diagnosis of complex devices. In *Proceedings of AAAI-93*, pages 168–175, Washington, D.C., July 1993.

[FGLJ90] M. P. Féret, J. I. Glasgow, D. Lawson, and M. A. Jenkins. An architecture for real-time diagnosis systems. In *Proceedings of the Third International Conference on Industrial and Engineering Applications and Expert Systems*, pages 9–15, Charleston, SC, July 1990.

[GC89] A. Goel and B. Chandrasakeran. Use of device models in adaptation of device cases. In Hammond K., editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning, Volume 2*, pages 100–109, 1989.

[GC92] A. Goel and T. Callantine. An experience-based approach to navigational route planning. In *Proceedings of the IEEE/RSJ Conference on' Intelligent Robots and Systems*, pages 705–710, Raleigh, NC, 1992.

[GS88] T. Govindaraj and Y. L. Su. A model of fault diagnosis performance of expert marine engineers. *International Journal on Man Machine Studies*, 29:1–20, 1988.

[Gup91] U. Gupta, editor. *Validating and Verifying Knowledge-Based Systems.* IEEE Computer Society Press, Los Alamitos, 1991.

[Ham89] K. J. Hammond. *Case-Based Plannning: Viewing Planning as a Memory Task.* Academic, Boston, 1989.

[Hin88] T. R. Hinrichs. Towards an architecture for open-world problem-solving. In Kolodner J., editor, *Proceedings of the DARPA Workshop on Case-Based Reasoning, Volume 1*, pages 182–189, 1988.

[Kol87] J. L. Kolodner. Extending problem solver capabilities through case-based inference. In *Proceedings of the 4th International Workshop on Machine Learning*, pages 167–178, June 1987.

[Kot88] P. Koton. Reasoning about evidence in causal explanations. In *Proceedings of AAAI-88*, pages 256–261, 1988.

[KS89] J. L. Kolodner and R.L. Simpson. The mediator: Analysis of an early case-based problem solver. *Cognitive Science*, 13(4):507–549, 1989.

[OBS87] R. O'Keefe, O. Balci, and E Smith. Validating expert system performance. *IEEE Expert*, 2(4), 1987.

[O'L87] D. O'Leary. Validation of expert systems – with applications to auditing and accounting expert systems. *Decision Sciences*, 18(3), 1987.

[O'L93] Daniel E. O'Leary. Verification and validation of case-based systems. *Expert Systems with Applications*, 6(1):57–66, 1993.

[Red89] M. Redmond. Combining case-based reasoning, explanation-based learning and learning from instruction. In *Proceedings of the 6th International Workshop on Machine Learning*, Ithaca, New York, 1989. Morgan Kaufmann.

[SC85] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem-solving systems. Technical Report Tech. Rep., Ohio State University, Colombus, Ohio, 1985.

[Syc87] E. P. Sycara. *Resolving Adversarial Conflicts: An Approach to Integrating Case-Based Reasoning and Analytic Methods.* PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[YH88] W. C. Yoon and J. M. Hammer. Deep-reasoning fault diagnosis: An aid and a model. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(4):659–675, 1988.

A simple approach to Bayesian network computations

Nevin Lianwen Zhang
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

David Poole
Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z2, Canada

Abstract

The general problem of computing posterior probabilities in Bayesian networks is NP-hard (Cooper 1990). However efficient algorithms are often possible for particular applications by exploiting problem structures. It is well understood that the key to the materialization of such a possibility is to make use of conditional independence and work with factorizations of joint probabilities rather than joint probabilities themselves. Different exact approaches can be characterized in terms of their choices of factorizations. We propose a new approach which adopts a straightforward way for factorizing joint probabilities. In comparison with the clique tree propagation approach, our approach is very simple. It allows the pruning of irrelevant variables, it accommodates changes to the knowledge base more easily. it is easier to implement. More importantly, it can be adapted to utilize both intercausal independence and conditional independence in one uniform framework. On the other hand, clique tree propagation is better in terms of facilitating precomputations.

Keywords: reasoning under uncertainty, Bayesian networks, algorithm

1 Introduction

Several exact approaches to the computing of posterior probabilities in Bayesian networks have been proposed, studied, and some of them implemented. The one that is most well known is clique tree propagation, which has been developed over the past few years by Pearl (1988), Lauritzen and Spiegehalter (1988), Shafer and Shenoy (1988), and Jensen *et al* (1990). Other approaches include Shachter's arc reversal node reduction approach (Shachter 1988), symbolic probabilistic inference first proposed by D'Ambrosio (Shachter *et al* 1990), recursive decomposition by Cooper (1992), and component tree propagation by Zhang and Poole (1992).

This paper grew out of an attempt to understand those approaches and the relationships among them. We asked ourselves: are there any common principles that underlie all those approaches? If yes, what are the choices that render them different from one another? What are the advantages and disadvantages of these choices? Are there any better choices and/or any better combinations of choices?

Shachter *et al* (1992) has demonstrated the similarities among the various approaches. In this paper, we are more interested in the differences among them.

Cooper (1990) has proved that the general problem of computing posterior probabilities in Bayesian networks is NP-hard. In particular applications, however, it is often possible to compute them efficiently by exploiting the problem structures. The key technique that enables the materialization of such a possibility, as pointed out by Shafer and Shenoy (1988), is to work with factorizations of joint probabilities rather than the joint probabilities themselves. What all the exact approaches have in common is that they all adopt this technique, while they differ in their own choices of factorizations.

These understandings lead to a new approach that chooses a straightforward factorization for joint probabilities. Though very simple, the new approach has several advantages over clique tree propagation

in terms of pruning irrelevant variables, accommodating changes to the knowledge base and easiness of implementation. It also leads to a uniform framework for utilizing both conditional and intercausal independence. The only disadvantage we can think of is that it does not facilitate precomputation.

The organization of the paper is as follows. Preliminary definitions are given in section 2. Section 3 reviews results concerning the irrelevance of variables to a query. After the removal of irrelevant variables, queries about posterior probabilities can be transformed into a standard form, i.e queries about marginal probabilities. For technical convenience, further exposition will be carried out in terms of potentials rather the probabilities (section 4). In section 5, we illustrate the technique of working with factorizations of joint potentials. The subproblem of data management is identified in section 6. Clique tree propagation is one solution to this subproblem. A new and very simple solution is proposed in section 7, which is based on a simple way for factorizing joint potentials. In section 8, we compare the solution to clique tree propagation. Some conclusions are provided in section 9.

2 Preliminaries

We begin by giving a definition of Bayesian networks.

A *Bayesian network* $\mathcal{N}$ is a triplet $(V, A, \mathcal{P})$, where

1. V is a set of variables.
2. A is a set of arcs, which together with V constitutes a directed acyclic graph $G = (V, A)$.
3. $\mathcal{P} = \{P(v|\pi_v) : v \in V\}$, where π_v stands for the set of parents of v. In words, $\mathcal{P}$ is the set the conditional probabilities of the all variables given their respective parents [1].

Figure 1 show a simple Bayesian network `net1` with seven variables a, b, c, d, e, f, and g. The network contains the following prior and conditional probabilities: $P(a)$, $P(f|a)$, $P(b|a)$, $P(c|b)$, $P(d|b)$, $P(e|c,d)$, and $P(g|f,e)$.

Note that variables in a Bayesian network will be referred as nodes when they are viewed as members of the underlying graph. Also note that the graphical structure of a Bayesian network can be read from the set of the prior and conditional probabilities. So, we can use the symbol $\mathcal{N}$ to refer to the set of prior and conditional probabilities $\mathcal{P}$ without causing any confusion.

The *prior joint probability* $P_{\mathcal{N}}$ of a Bayesian network $\mathcal{N}$ is defined by

[1] Note that when v is a root, π_v is empty. In such a case, the expression $P(v|\pi_v)$ simply stands for the prior probability of v.

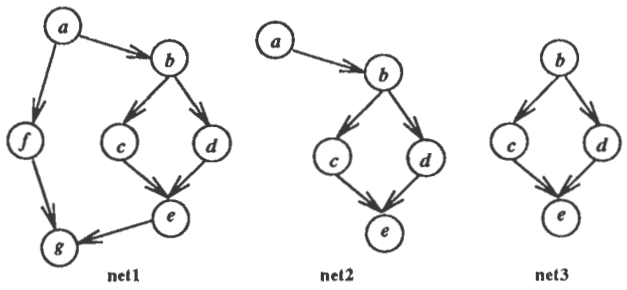


Figure 1: Bayesian network and irrelevant variables.

$$P_{\mathcal{N}}(V) = \prod_{v \in V} P(v|\pi_v). \quad (1)$$

For example, the prior joint probability P_{net1} of `net1` is given by

$$P_{net1}(a,b,c,d,e,f,g) = P(a)P(f|a)P(b|a)P(c|b)P(d|b)P(e|c,d)P(g|f,e).$$

For any subset X of V, the *marginal probability* $P_{\mathcal{N}}(X)$ is defined by

$$P_{\mathcal{N}}(X) = \sum_{V-X} P_{\mathcal{N}}(V).$$

Some variables may be observed to have specific values. For example, the variable b in `net1` may be observed to be a specific value b_0. Let $Y \subseteq V$ be the set of variables observed and Y_0 be the corresponding set of values. Let $X \subseteq V$ be the set of variables of interest. The *posterior probability* $P_{\mathcal{N}}(X|Y = Y_0)$ of X is defined by

$$P_{\mathcal{N}}(X|Y = Y_0) = \frac{P_{\mathcal{N}}(X, Y = Y_0)}{P_{\mathcal{N}}(Y = Y_0)}. \quad (2)$$

The problem of concern to this paper is how to compute $P_{\mathcal{N}}(X|Y = Y_0)$?

3 Irrelevant variables and standard queries

Given a query to a Bayesian network $\mathcal{N}$, it is often possible to graphically identify certain variables being irrelevant to the query. This issue is addressed in Geiger *et al* (1990), Lauritzen *et al* (1990), and Baker and Boult (1990). The materials in this section are extracted from those papers.

To *remove* a node v from a Bayesian network $\mathcal{N} = (V, A, \mathcal{P})$ is to: (1) remove v from V, (2) remove from A all the arcs that contain v, (3) remove from $\mathcal{P}$ all the items that involve v, and (4) set the prior probabilities

for all the nodes, if any, that become roots[2] because of the removal to be the uniform distribution.

A node in a Bayesian network $\mathcal{N}$ is *leaf* if it has no children. A node is *barren* w.r.t a query $P_{\mathcal{N}}(X|Y = Y_0)$, if it is a leaf and it is not in $X \cup Y$. In **net1**, g is barren w.r.t $P_{net1}(e|b = b_0)$.

Theorem 1 *Suppose $\mathcal{N}$ is a Bayesian network, and v is a leaf node. Let $\mathcal{N}'$ be the Bayesian network obtained from $\mathcal{N}$ by removing v. If v is barren w.r.t to the query $P_{\mathcal{N}}(X|Y = Y_0)$, then*

$$P_{\mathcal{N}}(X|Y = Y_0) = P_{\mathcal{N}'}(X|Y = Y_0). \quad (3)$$

Consider computing $P_{net1}(e|b = b_0)$. The node g is barren w.r.t the query and hence irrelevant. According to Theorem 1, g can be harmlessly removed. This creates a new barren node f. After the removal of g and f, **net1** becomes **net2**. Thus the query $P_{net1}(e|b = b_0)$ is reduced to the query $P_{net2}(e|b = b_0)$.

Let $An(X \cup Y)$ be the *ancestral set* of $X \cup Y$, i.e the set of nodes in $X \cup Y$ and of the ancestors of those nodes. By repeatedly applying Theorem 1, one can easily show that

Corollary 1 *All the nodes outside $An(X \cup Y)$ are irrelevant to the query $P(X|Y = Y_0)$.*

The *moral* graph $m(G)$ (Lauritzen and Spiegehalter 1988) of the an directed graph $G = (V, A)$ is the undirected graph obtained from G be marrying the parents of each node (i.e adding an edge between each pair of parents), and then dropping all directions. If two nodes x and y are separated by a set B in $m(G)$, we say that x and y are *m-separated* by B in G. The term m-separation is new, but the concept itself was used Lauritzen *et al* (1990).

Theorem 2 *Suppose $\mathcal{N} = (V, A, \mathcal{P})$ is a Bayesian network. Given a query $P_{\mathcal{N}}(X|Y = Y_0)$, let $\mathcal{N}'$ be the Bayesian network obtained from $\mathcal{N}$ by removing all the nodes that are m-separated from X by Y. Then*

$$P_{\mathcal{N}}(X|Y = Y_0) = P_{\mathcal{N}'}(X|Y = Y_0). \quad (4)$$

In our example, since a is m-separated from e by b in **net2**, the query can be further reduced to $P_{net3}(e|b = b_0)$ Note that a is not m-separated from e by b in **net1**.

It can be proved (Lauritzen *et al* 1990 and Geiger *et al* 1990) that, given a query, all the irrelevant nodes that are graphically recognizable can be recognized by applying those two theorems.

From equation (2), we see that $P_{\mathcal{N}}(X|Y = Y_0)$ can be obtained from $P_{\mathcal{N}}(X, Y = Y_0)$ by multiplying a renormalization constant. However, the two queries are different in terms of irrelevant variables. For example, a is irrelevant to the query $P_{net2}(e|b = b_0)$, but relevant to the query $P_{net2}(e, b = b_0)$.

From now on, we will assume that all the irrelevant variables have been removed unless otherwise indicated. Under this assumption, we can replace the query $P_{\mathcal{N}}(X|Y = Y_0)$ with the query $P_{\mathcal{N}}(X, Y = Y_0)$. We call the latter a *standard query*. The rest of the paper will only be dealing with standard queries.

4 Potentials

A *potential* is a non-negative function which takes a positive value at at least one point. Here are some example potentials. The probability $P(X)$ is a potential of X, the conditional probability $P(X|Y)$ is a potential of X and Y, and $P(X, Y = Y_0)$ is a potential of X.

Let S be a set of potentials over a set of variables V. The *marginal potential* $P_S(X)$ is defined as follows: Multiply all the potentials in S together to get the *joint potential* $P_S(V)$, and $P_S(X)$ is obtained from $P_S(V)$ by summing out all the variables outside X. It is obvious that marginal probability is a special case of marginal potentials. For technical convenience, we shall be talking about marginal potential $P_S(X, Y = Y_0)$ instead of marginal probability $P_{\mathcal{N}}(X, Y = Y_0)$ from now on.

5 The key technique

Let S be a set of potentials over the set V of variables. A naive way to compute $P_S(X, Y = Y_0)$ is first to explicitly compute and store the joint potential $P_S(V)$, and then compute $P_S(X, Y = Y_0)$ from $P_S(V)$. This method is not efficient.

Even though the general problem of computing posterior probabilities in Bayesian networks is NP-hard, efficient algorithms often exist for particular applications due to the underlying structures. The purpose of this section is to describe a key technique that allows us to make use one aspect of problem structure, namely conditional independencies. The technique is to work with factorizations of joint potentials (probabilities) rather than joint potentials (probabilities) themselves.

We say that a set S_1 of potentials is a *factorization of the joint potential* $P_S(V)$ if $P_S(V)$ is the result of multiplying the potentials in S_1. The set S itself is certainly a factorization of $P_S(V)$, and it is the most straightforward one because it is what one has to begin with. We call S the *primary factorization* of $P_S(V)$.

We will see later that clique tree propagation does not directly adopts the primary factorization. Rather it first performs some pre-organizations and precomputations on S and then proceeds with the resulting more organized factorization.

[2] Nodes that do not have parents.

Exponential explosion can be in terms of both storage space and time. It is quite easy to see why factorization is able to help us to save space. For the sake of illustration, consider the Bayesian network **net1** in Figure 1. If all the variables are binary, to store the set of potentials of **net1**, i.e the prior and conditional probabilities, one needs to store $2+4*4+2*8=34$ numbers. On the other hand, to explicitly store the joint potential (probability) $P_{net1}(a,b,c,d,e,f,g)$ itself, one needs to store $2^7=128$ numbers.

To see how factorizations of joint potentials enable us to save time, we assume that the summing-out-variables-one-by-one strategy is adopted for computing marginal potentials.[3] We also assume that an ordering has been given for this purpose. This ordering will be referred as the *elimination ordering*.

Since we choose to work with the a factorization, which is a set of potentials, we need to define how to sum out one variable from a set of potentials. *To sum out a variable v from a set of potentials S* is to: (1) remove from S all the potentials that contain v, (2) multiply all those potentials together, (3) sum out v from the product, and (4) add the resulting potential to S.

For example, to sum a out of **net1**, we first remove $P(a)$, $P(b|a)$ and $P(f|a)$ from **net1**, then compute

$$\psi_a(b,f)=\sum_a P(a)P(f|a)P(b|a), \tag{5}$$

and finally add $\psi_a(b,f)$ to **net1**. After all these operations, **net1** becomes $\{P(c|b),\ P(d|b),\ P(e|c,d),\ P(g|f,e),\ \psi_a(b,f)\}$.

Usually, it takes much less arithmetic calculations to sum out one variable from a factorization of a joint potential than from the joint potential itself. For example, equation (5) denote all the arithmetic calculations needed to sum out a from **net1**. It involves only three variables: a, b, and f. On the other hand, to sum out a explicitly from $P_{net1}(a,b,c,d,e,f,g)$ itself, one needs to perform the following calculations,

$$\sum_a P_{net1}(a,b,c,d,e,f,g),$$

which involves all the seven variables in the network. This is the exactly why working the factorizations of joint potentials enables us to reduce time complexity.

6 Three components

In implementing Bayesian networks, if one adopts the key technique outlined in the previous section, then the resulting system can be divided into three components.

[3]It must be noted that they are exact approaches that do not adopt this strategy. See Poole and Neufeld (1991) and Poole (1992) for examples.

The first component finds an elimination ordering. We call it the *ordering determination* component. Roughly speaking, an elimination ordering is good if the arithmetic calculations needed to sum out each variable, from the primary factorization, involve only a small number of other variables. Even with a clear and crisp definition, the ordering determination problem proves to be a difficult one. See Kjærulff (1990) and Klein *et al* (1990) for research progresses on the problem. In this paper, we shall not discuss it any further.

The third component is the *arithmetic calculation* component. It takes a bunch of potentials, multiply them together, sum out a certain variable from the product, and return the result. A major goal in designing Bayesian network inference algorithms is to minimize the total number of arithmetic calculations.

In between the first and the third components lies the *data management* component. It determines, from the elimination ordering produced by the first component, what arithmetic calculations the third component is going to perform, and in which order. The component also hides the design decisions as to how to store the potentials, how to retrieve a potential when it is needed, and how to update the set of potentials after a variable has been summed out. We call a design of the data management component a *data management scheme*.

Among the existing exact approaches to Bayesian network computations, clique tree propagation (Jensen *et al* 1990), the arc reversal node reduction approach (Shachter 1988), and symbolic probabilistic inference (Shachter *et al* 1990) are data management schemes; while recursive decomposition (Cooper 1990) and component tree propagation (Zhang and Poole 1992) are mixtured of data management schemes and ordering determination methods.

In the remainder of the paper, we shall first propose a very simple data management scheme (section 7), and we shall compare this scheme with clique tree propagation and the arc reversal and node reduction approach (sections 8 and 9).

7 A simple data management scheme

The following algorithms describes our design of the data management component. It takes, as input, a set of potential S, a standard query $(X,\ Y=Y_0)$ and an elimination ordering *Ordering*. The output is $P_S(X,Y=Y_0)$.

PROCEDURE **P**(S, $(X,\ Y=Y_0)$, *Ordering*)

1. Set Y to Y_0 in the potentials in S, resulting in S_1.
2. Associate each potential ψ of S_1 with the variable that appears earliest in

Ordering among all the variables of ψ,

3. **Repeat** till *Ordering* becomes empty,
   - Remove the first variable on *Ordering*. Denote this variable by v. Call subroutine **Arithemetic-Calculation** to multiply all the potentials associated v together and to sum out v from the product, resulting in ψ_v,
   - Associate ψ_v with the variable that appear earliest in *Ordering* among all the variables of ψ_v, and
4. Return the potential produced from the removal of the last variable in *Ordering*, which is $P_S(X, Y = Y_0)$.

As an example, let us consider computing $P_{net1}(g, a = a_0)$. Suppose the elimination ordering is (b, c, d, e, f). Then, the initial variable-potential association scheme is as follows:

$b:$	$c:$	$d:$	$e:$	$f:$
$P(b\|a_0)$	$P(e\|c,d)$		$P(g\|e,f)$	$P(f\|a_0)$
$P(c\|b)$				
$P(d\|b)$				

Let $\psi_b(c,d) = \sum_b P(b|a_0)P(c|b)P(d|b)$. Then after the removal of b, the association scheme becomes:

$c:$	$d:$	$e:$	$f:$
$P(e\|c,d)$		$P(g\|e,f)$	$P(f\|a_0)$
$\psi_b(c,d)$			

Let $\psi_c(d,e) = \sum_c P(e|c,d)\psi_b(c,d)$. After the removal of c, we get

$d:$	$e:$	$f:$
$\psi_c(d,e)$	$P(g\|e,f)$	$P(f\|a_0)$

Let $\psi_d(e) = \sum_d \psi_c(d,e)$. After the removal of d, we get

$e:$	$f:$
$P(g\|e,f)$	$P(f\|a_0)$
$\psi_d(e)$	

Let $\psi_e(f,g) = \sum_e P(g|e,f)\psi_d(e)$. After the removal of d, we get

$f:$
$P(f\|a_0)$
$\psi_e(f,g)$

Finally, let $\psi_f(g) = \sum_f P(g|e,f)\psi_e(f,g)$. The potential $\psi_f(g)$ is $P_{net1}(g, a = a_0)$.

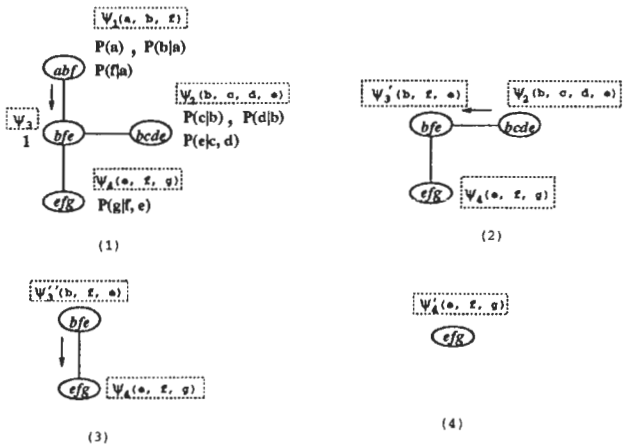


Figure 2: Clique tree propagation.

8 Comparing with clique tree propagation

We begin this section with a brief review of clique tree propagation. For this paper, a *clique* can be simply understood as a subset of variables. A *clique tree* is a tree whose nodes are cliques such that if a variable appear on two different nodes, then it also appears in all the nodes in the path between the two nodes. A *clique tree for a set of potentials* is a clique tree such that for each potential in the set, there is at least one clique in the tree that contains all its variables.

Like our approach, clique tree propagation reduces time and space complexities by working with factorizations of joint potentials. Unlike in our approach, which begins with the primary factorization, clique tree propagation associates the potentials in the primary factorization with the cliques in a clique tree, such that each potential is associated with one and only one clique that contains all its variables. All the potentials associated with one clique are multiplied together, resulting one single potential. If there is no potential associated with a clique, the constant potential 1 is stuck in. Thus the initial factorization for clique tree propagation consists of one and only one potential for each clique.

Figure 2 (1) shows a clique tree for the Bayesian network **net1** in Figure 1, together with a grouping of its prior and conditional probabilities (potentials). The initial factorization is $\{\psi_1(a,b,f), \psi_2(b,c,d,e), \psi_3(b,f,e), \psi_4(e,f,g)\}$, where $\psi_1(a,b,f) =_{def} P(a)P(b|a)P(f|a)$, $\psi_2(b,c,d,e) =_{def} P(c|b)P(d|b)P(e|c,d)$, $\psi_3(b,f,e) =_{def} 1$, and $\psi_4(e,f,g) =_{def} P(g|f,e)$.

Clique tree propagation computes a marginal potential by message passing in the clique tree. To pass a message from one node C (a clique) to one of its neighbors D, one sums out, from the potential associated with C, the variables in $C - D$, send the resulting potential to

D, and update the potential currently associated with D by multiplying it with the potential from C. Figure 2 show the message passing process for computing $P_{net1}(g, a = a_0)$.

In Jensen *et al* (1990), the prior marginal probability of each clique (node) in the clique tree is precomputed and stored at the node. To compute $P_{net1}(g|a = a_0)$ in this scheme, one only needs to pass proper messages from node (abf) to (bfe), and then to (efg). No message from node $bcde$ to node (bfe) is necessary. See the cited paper for details.

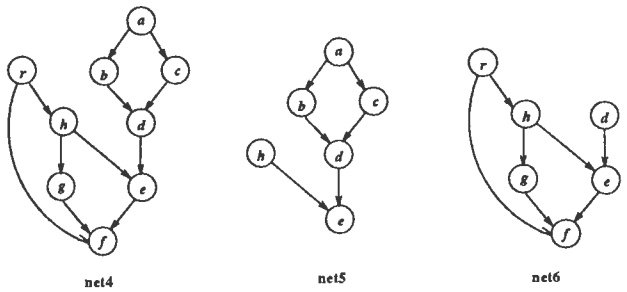


Figure 3: Pruning vs. precomputing.

8.1 Comparisons

Before commencing the comparisons, let us point out the both our approach and clique tree propagation have the same starting point. Our approach begin with an elimination ordering, and clique tree propagation begins with a clique tree. The availability of a clique tree is equivalent to the availability of an elimination ordering for the empty query $P_S(\emptyset)$. There are linear time algorithms to obtain an elimination ordering from a clique tree and to get back the clique tree from the ordering (see, for example, Zhang 1991).

To compare our data management scheme with clique tree propagation, we notice that our approach handles changes to the knowledge base more easily. Clique tree is a secondary structure. Any topology changes to the original network, like adding or deleting variable, or adding or deleting an arc, require recomputing the clique tree and the potential associated with each clique. In the Jensen *et al* (1990) scheme, one has to recompute the marginal probabilities for all the cliques even when there are only numerical adjustments to the conditional probabilities.

Secondly, if in a query $P_{\mathcal{N}}(X|Y = Y_0)$, X is not contained in any clique of the clique tree, then the secondary structure has to be modified, at least temporarily. This is even more cumbersome in the Jensen *et al* (1990) Scheme.

Two major issues in comparing our approach with clique tree propagation are pruning and precomputing, to which we devote the next subsection.

8.2 Pruning vs. precomputing

Pruning irrelevant variables and precomputing the prior probabilities of some variables are two techniques to cut down computations. In this section, we shall illustrate those two techniques through an example and discuss some related issues.

Consider the query about posterior probability $P_{net4}(e|h = h_0)$, where **net4** is given in Figure 3. One can first prune f because it is barren. Thereafter, g and r can also be pruned because g becomes barren after the removal of f and r becomes m-separated from e by h. Thus, pruning enables one to transform the original query into $P_{net5}(e|h = h_0)$, a query to a Bayesian network with three variables less.

One the other hand, it is easy to see that summing out variables a, b, and c in **net4** results in **net6**, where $P(d) = \sum_{a,b,c} P(a)P(b|a)P(c|a)P(d|b,c)$. So, if $P(d)$ is precomputed, the query $P_{net4}(e|h = h_0)$ can be immediately transformed into $P_{net6}(e|h = h_0)$, a query to a Bayesian network with three variables less.

So, both pruning and precomputing enable us to cut down computations. However, they both have prices to pay as well.

Pruning irrelevant variables implies that we will be working with a potentially different sub-network for each query. This usually means that an elimination ordering is to be found for each particular query from scratch, instead of deriving it from an ordering for the empty query.

We argue that this is not a serious drawback for two reasons. First, if a query only involve a few variables, pruning may give us a sub-network which is only a small portion of the original network. After all, only those variables who are ancestors of variables in the query could be relevant to the query. Thus pruning makes finding a good elimination ordering much easier. Second, some existing hueristics for finding elimination, like maximal cardinality search and lexicographic search (Rose 1970, Kjæruff 1990), are quite simple. It does not take much more time to find an ordering from scratch.

As far as precomputing is concerned, it is difficult to decide what to precompute. For example, precompute $P(g)$ is not very helpful in the previous example. One solution is to compute the prior probabilities for all the combinations of variables. But this implies an exponential number of precomputations.

Clique tree propagation works on the secondary structure of clique tree, which is kept static. This makes precomputing possible (Jensen *at al* 1990). As we pointed out early, however, there is a price to pay. The approach does not prune irrelevant variables, and it is hard for it to accommodate changes to the knowledge base.

8.3 Intercausal independence

A major reason for us to come up with a new data management scheme is that it leads to a uniform framework for utilizing conditional independence as well as intercausal independence.

In Zhang and Poole (1994), we give a constructive definition of intercausal independence. Noisy OR-gates and noisy adders satisfy our definition. A nice property of our definition is that it relates intercausal independence with factorization of conditional probabilities, in a way very similar to that conditional independence is related factorization of joint probabilities. The only difference lies in the way the factors are combined. While conditional independence implies that a joint can be factorized as a *multiplication* of several factors, intercausal independence implies that a conditional probability can be factorized as a *certain combination* of several factors, where combination is usually not multiplication.

The concept of heterogeneous factrization is proposed. A heterogeneous factorization is one where different factors can be combined in different ways. We have adapted the data management scheme proposed in this paper to handle heterogeneous factorizations.

9 Conclusions

A key technique to reduce time and space complexities in Bayesian networks is to work factorizations of joint potentials rather than joint potentials themselves. Different exact approaches can be characterized by their choices of factorizations. We have proposed a new approach which begins with the primary factorization. Our approach is simpler than clique tree propagation. Yet it is advantageous in terms of pruning irrelevant variables and accommodating changes to the knowledge base. More importantly, our approach leads to a uniform framework for dealing with both conditional and intercausal independence. However, our approach does not support precomputation as clique tree propagation does.

Acknowledegment:

We wish to thank Mike Horsch for his valuable comments on a draft of this paper and Runping Qi for useful discussions. Research is supported by NSERC Grant OGPOO44121 and travel grants from Hong Kong University of Science and Technology.

References:

M. Baker and T. E. Boult (1990), Pruning Bayesian networks for efficient computation, in *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, July, Cambridge, Mass. , pp. 257 - 264.

G. F. Cooper (1990) The computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence*, **42**, pp. 393-405.

G. F. Cooper (1990), Bayesian belief-network inference using recursive decomposition, Report No. KSL 90-05, Knowledge Systems Laboratory, Medical Computer Science, Standford University.

D. Geiger, T. Verma, and J. Pearl (1990), *d*-separation: From theorems to algorithms, in *Uncertainty in Artificial Intelligence* **5**, pp. 139-148.

F. V. Jensen, K. G. Olesen, and K. Anderson (1990), An algebra of Bayesian belief universes for knowledge-based systems, *Networks*, 20, pp. 637 - 659.

U. Kjærulff (1990), Triangulation of Graphs - Algorithms giving small total state space, R 90-09, Institute for Electronic Systems, Department of Mathematics and Computer Science, Strandvejen, DK 9000 Aalborg, Denmark.

P. Klein, A. Agrawal, A. Ravi, and S. Rao (1990), Approximation through multicommodity flow, in Proceedings of 31st Symposium on Foundations of Computer Science, pp. 726-737.

S. L. Lauritzen and D. J. Spiegehalter (1988), Local computations with probabilities on graphical structures and their applications to expert systems, *Journal of Royal Statistical Society B*, 50: 2, pp. 157 - 224.

S. L. Lauritzen, A. P. Dawid, B. N. Larsen, and H. G. Leimer (1990), Independence Properties of Directed Markov Fields, *Networks*, 20, pp. 491-506.

J. Pearl (1988), *Probabilistic Reasoning in Intelligence Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, Los Altos, CA.

D. Poole and E. Neufeld (1991), Sound probabilistic inference in Prolog: An executable specification of Bayesian networks, Department of Computer Science, University of British Columbia, Vancouver, B. C., V6T 1Z2, Canada.

D. Poole (1992), Search for Computing posterior probabilities in Bayesian networks, Technical Report 92-24, Department of Computer Science, University of British Columbia, Vancouver, Canada.

D. J. Rose (1970), Triangulated graphs and the elimination process, *Journal of Mathematical Analysis and Applications*, **32**, pp 597-609.

R. Shachter (1986), Evaluating Influence Diagrams, *Operations Research*, 34, pp. 871-882.

R. Shachter (1988), Probabilistic Inference and Influence Diagrams, *Operations Research*, 36, pp. 589-605.

R. D Shachter, S. K. Andersen, and P. Szolovits (1992), The equivalence of exact methods for probabilistic inference in belief networks, Department of

Engineering-Economic Systems, Stanford University.

R. D. Shachter, B. D'Ambrosio, and B. A. Del Favero (1990), Symbolic Probabilistic Inference in Belief Networks, in *AAAI-90*, pp. 126-131.

G. Shafer and P. Shenoy (1988), Local computation in hypertrees, Working Paper No. 201, Business School, University of Kansas.

L. Zhang (1991), Studies on hypergraphs (I): Hyperforests, accepted for publication on *Discrete Applied Mathematics*.

L. Zhang and D. Poole (1992) Sidestepping the triangulation problem in Bayesian net computations, in *Proc. of 8th Conference on Uncertainty in Artificial Intelligence*, July 17-19, Standford University, pp. 360-367.

L. Zhang and D. Poole (1994) Intercausal independence and heterogeneous factorizations, submitted to *The Tenth Conference on Uncertainty in Artificial Intelligence*

A Polynomial-time Hypothetical Reasoning employing an Approximate Solution Method of 0-1 Integer Programming for Computing Near-optimal Solution

Mitsuru Ishizuka and **Tomoki Okamoto**+
Dept. of Information & Communication. Eng.
Faculty of Engineering, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo 113, Japan

+Presently with Tokyo Electric Power Co.

Abstract

A hypothetical reasoning is an important knowledge system's framework because of its theoretical basis and its usefulness for practical problems including diagnosis, design, etc. One crucial problem with the hypothetical reasoning is, however, its slow inference speed. In order to achieve practical or tractable inference speed, we apply an approximate solution method of 0-1 integer programming to a weight-based hypothetical reasoning, where a numerical weight is assigned to each possible element hypothesis and the optimal solution hypothesis set with minimal sum of its element hypotheses' weights is searched. In this method, we regard all described knowledge as constraints. To narrow down the search space, we first extract restricted knowledge relevant to the proof of a given goal. Then, we transform the restricted knowledge into inequatlities to apply 0-1 integer programming. While the computational complexity of the hypothetical reasoning is NP-complete or NP-hard, an approximate solution method of 0-1 integer programming allows polynomial inference time for finding a near-optimal solution hypothesis.

1. Introduction

A hypothetical reasoning is an important framework for advanced knowledge-based systems because of its theoretical basis and its usefulness for many practical problems including diagnosis, design, etc. [Poole 87, 88, Ishizuka 90, Makino 90]. It is an abductive inference mechanism for finding consistent solution hypotheses satisfying given constraints. One crucial problem with the hypothetical reasoning is its slow inference speed because a non-monotonic inference is needed due to the use of hypothetical or defeasible knowledge.

In order to overcome this problem, the authors' group has developed so far several fast hypothetical reasoning methods, e.g., 1) fast hypothetical reasoning using inference-path network [Ito 91, Ishizuka 91] which includes ATMS mechanism [deKleer 86], 2) fast hypothetical reasoning for predicate-logic knowledge-base [Kondo 91] which employs a deductive database technique, 3) fast hypothetical reasoning using analogy [Ishizuka 93], and 4) knowledge-base compilation method [Tsuruta 91, 92]. Since the computational complexity of the hypothetical reasoning is NP-complete or NP-hard [Kautz 89, Bylander 89, Stillman 90], we cannot overcome the wall of exponential inference time with respect to problem size as long as we use ordinary inference methods. The above methods 2) and 3), for example, use analogical reasoning and knowledge-base compilation, respectively, to overcome this inference speed limit.

In this paper, we present another efficient hypothetical reasoning method based on an approximate solution method of 0-1 integer programming. Here, we consider a weight-based or cost-based hypothetical reasoning [Charniak 90], where a numerical weight is assigned to each possible element hypothesis and an optimal solution hypothesis set is searched which has the minimum weight sum of the element hypotheses. This framework is useful, for example, for finding the most possible diagnosis or the least expensive design satisfying given constraints.

We regard the described logical knowledge as constraints and transform them into inequalities to apply the 0-1 integer programming method. Before this application, we find the restricted portion of knowledge relevant to the proof of a given goal so as to narrow down the search space. As a result, we show that the approximate solution method of 0-1 integer programming is very effective to find a near-optimal solution hypothesis in polynomial time.

Recently, it is recognized that the combination of mathematical programming techniques with knowledge-based processing is useful to achieve an efficient inference [Hooker 88, Dhar 90, Charniak 92]. However, the use of an approximate solution method was not considered in [Chaniak 92]. While its use was considered in [Dhar 90], a preprocessing of restricting the scope of knowledge being introduced in this paper had not been incorporated. Since the computational complexity of 0-1 integer programming is still NP-complete, it is necessary to incorporate a preprocessing of reducing the number of variables for 0-1 integer programming and to apply an approximate solution method, for achieving an efficient and tractable inference of the hypothetical reasoning.

Probabilistic search methods based on simulated annealing(SA) [Kirkpatrik 83] or genetic algorithm(GA) [Goldberg 89] are also exploited recently in AI area for efficiently finding near-optimal solutions. The search of our inference method is different from these probabilistic methods; our method uses guiding information obtained by the efficient simplex method for a corresponding problem relaxed from 0-1 integer domain to real domain, and excecutes a local search for a 0-1 optimal solution around the optimal solutuion in real domain.

Also, the efficiency of the local search for CSP (constraint satisfaction problem) or SAT (satisfiability testing) is recently recognized such as in a heuristic repair method [Minton 92] and GSAT [Selman 93a, 93b]. Our method differs from these methods in that the efficient simplex method is used to obtain a good initial guess and analog-value points between (or sometimes outside) 0-1 vertex points are considered in the local search process. In addition, our method can compute a near-optimal solution rather than a simple single solution satisfying the constraints. The use of unconstrained nonlinear programming for CSP or SAT is shown in [Gu 93]. This local search process is conceptually similar to our method; however, it does not provide any mechanism to determine a good initial search point. Thus our method, while it is described in the context of the hypothetical reasoning in this paper, may indicate a new efficient search for wider problem solving under declarative knowledge.

2. Logic-based Hypothetical Reasoning

The hypothetical reasoning in this paper is a logic-based one [Poole 87, 88, Ishizuka 91, Ito 91], where knowledge is divided into two categories, i.e., background knowledge (or fact in [Poole 87, 88]) and hypothesis. Background knowledge denoted by Σ has no possibility of inconsistency, whereas the hypothesis denoted by H has the possibility of contradiction with other hypotheses, and thus is defeasible knowledge.

As illustrated in Fig.1, the basic behavior of this hypothetical reasoning is as follows. When a goal G is given, the system first tries to prove this goal from background knowledge. If it fails, then the system selects a subset of the hypotheses so that the given goal is proved from the union of bacground knowledge and this hypothesis subset, which should be consistent with background knowledge. This consistent subset of hypotheses becomes a solution for the given goal in the hypothetical reasoning system. The generation of this consistent hypothesis subset can be viewed as abduction.

The structure of above hypothetical reasoning can be summarized to find a solution h satisfying

- $h \subseteq H$ (h is a subset of H)
- $\Sigma \cup h \vdash G$ (G can be derived from $\Sigma \cup h$), and
- $\Sigma \cup h \not\vdash \square$ ($F \cup h$ is consistent, $\square$: empty clauses),

where Σ, H and G are background knowledge, possible hypotheses and a given goal, respectively.

In this paper, we restrict the knowledge representation to propositinal Horn clauses, since our main concern here is an efficient inference mechanism. Since the logical negation of an atom cannot be expressed with Horn clauses, we introduce an atom called "inc" to denote inconsistency among hypotheses, such as,

$$inc \leftarrow h_1, h_2.,$$

which says that h_1 and h_2 cannot be coexist in an environment.

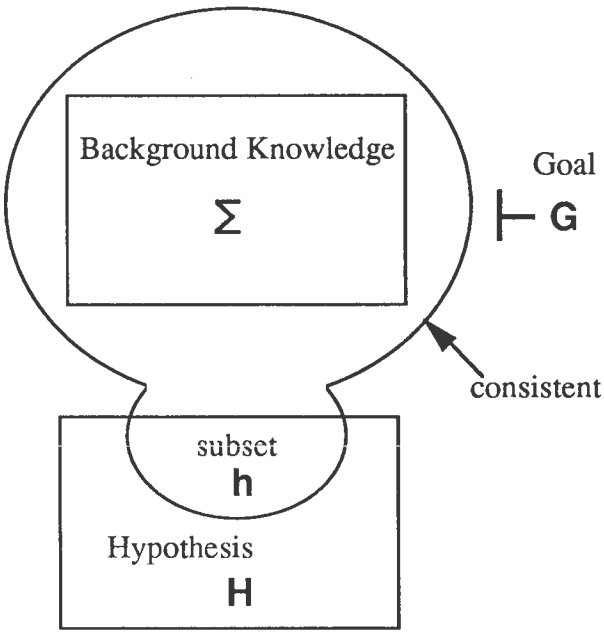


Fig. 1 Basic structure of logic-based hypothetical reasoning.

Rule-type hypotheses are allowed in general in our hypothetical reasoning system. They are, however, transformed by a preprocessing into newly introduced single-atom hypotheses and modified background knowledge. For example, a rule-type hypothesis 'p←q.' will be transformed by introducing a new atom 'r' into,

background knowledge $p \leftarrow q, r.$, and
hypothesis r.

In this case, the hypothesis 'p←q.' can be interpreted as being included in a solution hypothesis set if 'r' is included in the solution hypothesis set. With this preprocessing, all the hypotheses become unit clauses (single atoms).

There are often cases that a goal with a non-Horn clause such as,

$$s_1 \,\&\, \text{---} \,\&\, s_m \leftarrow t_1, \text{---}, t_n.$$

is given to the system; t_1,---, t_n and s_1, ---, s_m may be, for example, input and output observations, respectively, in a fault diagnosis problem, or an input-output specification in a circuit design problem. In these cases, by introducing an atom 'g' indicating an inference goal, we add

$$g \leftarrow s_1, \text{---}, s_m$$
$$t_1.$$
$$\vdots$$
$$t_n.$$

into the knowledge-base as background knowledge, and then try to prove 'g'.

There exist plural solution hypotheses sets in general. In many cases, an optimal solution hypothesis set with the minimum weight sum of its element hypotheses is required as the solution. We consider this type of a weight-based or cost-based hypothetical reasoning in this paper. A weight for each element hypothesis is defined in our system , for example, as,

hyp (h_1, 2),

where 2 is a numerical weight assigned to hypothesis h_1.

Figure 2 dipicts the functions of the hypothetical reasoning system described in this paper.

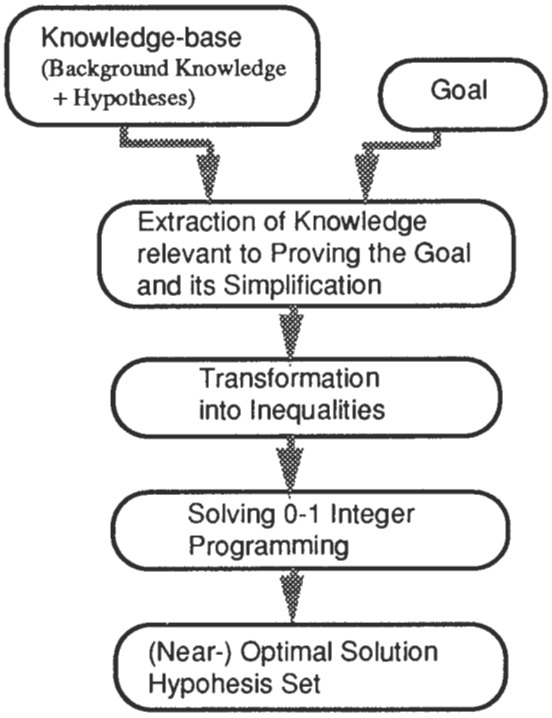


Fig. 2 Structure of the hypothetical reasoning of this paper.

3. Transformation of Knowledge into Inequalities for Applying 0-1 Integer Programming

In order to apply 0-1 integer programming for solving a hypothetical reasoning problem, we have to transform described knowledge into linear inequalities. We first describe this transformation method employed in our system.

We assume closed world assumption (CWA) [Clark 78] for the knowledge-base; i.e., we regard knowledge not explicitly described in the knowledge-base as false. In this situation, we can interpret a fact unable to be deductively proved from the knowledge-base as false; this is called 'negation as failure'. For example, suppose that

$a \leftarrow b, c.,\quad a \leftarrow d.,\quad e \leftarrow f.,\quad b.,\quad f.,$

are described in a knowledge-base. In this case, since 'a' cannot be proved to be true from this knowledge-base, we interpret '$\neg$a' is true.

To give a model theoretic semantics to the negation as failure, we introduce the concept of completion [Clark 78, Lloyd 84]. Simply speaking, we rewrite 'e $\leftarrow$ f.' into 'e $\leftrightarrow$ f.' ('e' is true iff 'f' is true). Also, if an atom 'd' appears in the body of a Horn clause and there exist no Horn clause with the head of 'd', we add '$\neg$d' to the knowledge-base. By this completion procedure, the above-mentioned illustrative knowledge-base becomes

$a \leftrightarrow ((b \wedge c) \vee d),\quad b,\quad \neg c,$

$\neg d,\quad e \leftrightarrow f,\quad f.$

In the case of propositional Horn clauses, it is known that the completion and the closed world assumption are equivalent.

In the framework of our hypothetical reasoning, we can assume the closed world assumption for background knowledge and apply the completion. Since it is unknown whether a hypothesis is true or not in a certain environment, we regard it as a variable to be determined for satisfying a given goal under the background knowledge. That is, we regard the hypothesis as a 0-1 variable in 0-1 integer programming.

Generally speaking, logic formulae in the knowledge-base can be regarded as constraints. Transforming these constraints into inequalities, we can apply 0-1 integer programming for obtaining a solution. Here, we consider a method of transforming the completed knowledge-base into inequalities. (Another transformation method for uncompleted knowledge-base is shown in [Hooker 88].)

Firstly, modeling the propositional logic formulae by Boolean algebraic equations, we rewrite the truth value (true/false) of an atom into 1/0, and equivalence symbol ($\leftrightarrow$) into equality symbol (=). Then, we can reduce all the knowledge except hypotheses into one of the following forms.

(1) $p = q_1 \vee \cdots \vee q_n$,

(2) $p = (q_1 \wedge \cdots \wedge q_n) \vee r$

where $p, q_i, r \in \{0,1\}$, $i = 1,2, \cdots, n$,

(3) $p = 1$ if p is defined,

(4) $p = 0$ if $\neg p$ is defined.

Next, we transform above Boolean equations of (1) and (2) into equivalent linear inequalities. These equivalent linear inequalities are not unique; however, we adopt here,

$$(1')\ \frac{q_1 + \cdots + q_n}{n} \leq p \leq q_1 + \cdots + q_n$$

$$(2')\ \frac{q_1 + \cdots + q_n + nr - (n-1)}{2n} \leq p \leq \frac{q_1 + \cdots + q_n + nr}{n}$$

for (1) and (2), respectively. All the rule-type knowledge can be expressed by these linear inequalities, if necessary, by introducing supplementary variables. The atom 'inc' indicating inconsistency is transformed into 'inc=0', and the atom indicating the goal is assigned 1 since it is a constraint to be satisfed.

With above procedures, the hypothetical reasoning with the propositional logic expression can be reformulated into 0-1 integer programming; among 0-1 integer solutions satisfying all the constraints, the variables with 1 become to represent a solution hypothesis set. Setting the weight sum of element

hypotheses to the objective function of the 0-1 integer programming, we can obtain the optimal solution hypothesis set by calculating the optimal solution of the 0-1 integer programming.

4. Extraction of Knowledge Relevant to Proving a Goal and its Simplification

With above-mentioned method, the optimal solution hypothesis set in the hypothetical reasoning can be computed in principle by the 0-1 integer programming method. However, if we transform all knowledge in the knowledge-base into inequalities and apply 0-1 integer programming, it becomes quite inefficient because the number of 0-1 variables becomes large. Practical performance cannot be attained by this simple application of 0-1 integer programming for practical-scale knowledge-bases, since, different from linear programming in real domain, the speed of integer programming is not sufficient.

For the hypothetical reasoning with propositional Horn clauses, it is possible to efficiently extract limited knowledge relevant to proving a given goal, while leaving the synthesis of necessary hypotheses to a later process, by the same method as one used in a fast hypothetical reasoning using inference-path network [Ito 91, Ishizuka 91]. The extracted knowledge can be further simplified or compiled efficiently. These procedures can be constructed on the basis of a linear-time algorithm of satisfiability testing for propositional Horn clauses [Dowling 84]. Since the computational complexity of 0-1 integer programming is NP-complete and its computational time increases exponentially with respect to the number of variables, this type of preprocessing for reducing the variables is necessary for achieving a highly efficient computation.

Our preprocessing consists of three processes. The first process is the formation of a goal-directed initial inference-path, and the second is its simplification. The third one is the extraction of relevant constraint knowledge indicating inconsistent combinations among hypotheses (hereinafter, inconsistency knowledge). For illustration purpose, we consider a knowledge-base including hypotheses h_1~h_{13} with numerical weights and a goal (a, b) shown in Fig. 3.

An initial inference-path network can be formed by a backward inference originated from the given goal. This inference-path network becomes to contain all relevant knowledge possibly to contribute to the proof of the goal. The unit clauses of background knowledge and element hypotheses are placed at the leaf nodes of this network. For the knowledge-base and the goal of Fig.3, the initial inference-path network of Fig.4 can be formed, for example. Knowledge such as p←e,h_{11}., q←k, l. ----, and the element hypotheses of h_9~h_{13} are not included in this network, since they are irrelevant to proving the goal (a, b) in this case.

In the simplification process of the inference-path network, 'true' state at the nodes corresponding to unit-clause background knowledge, and 'false' state at the nodes with no possibility of turning into 'true' because of lacking their child hypothesis nodes are propagated upward. That is, we assign 'true-by-hypothesis' state to all the intermediate nodes in the inference-path network except ones corresponding to the unit clauses of background knowledge. We also assign 'true-by-hypothesis' state to the hypothesis nodes. Then, the 'true' and 'false' states are propagated upward by changing the 'true-by-hypothesis' state of an AND node to 'true' state if all its AND-connected child nodes are in 'true' state and to false' state if one of them is in 'false' state, and by changing the 'true-by-hypothesis' state of an OR node to 'true' state if one of its OR-connected child nodes is in 'true' state and to 'false' state if all of them are in 'false' state.

In the illustrative initial inference-path network of Fig.4, since node 'f' is in 'true' state, node 'c' having this 'f' as its OR-connected child node turns into 'true' state. As a result, at node 'a' having this node 'c' as its AND-connected child node, we don't need to consider this node 'c' and its child nodes any more in the synthesis process of necessary hypotheses and need to consider only child node 'd'. Furthermore, node 'n' in Fig.4 can be identified as 'false' state, since it doesn't have any child hypothesis nodes and thus has no possibility of turning into 'true' state. Then, node

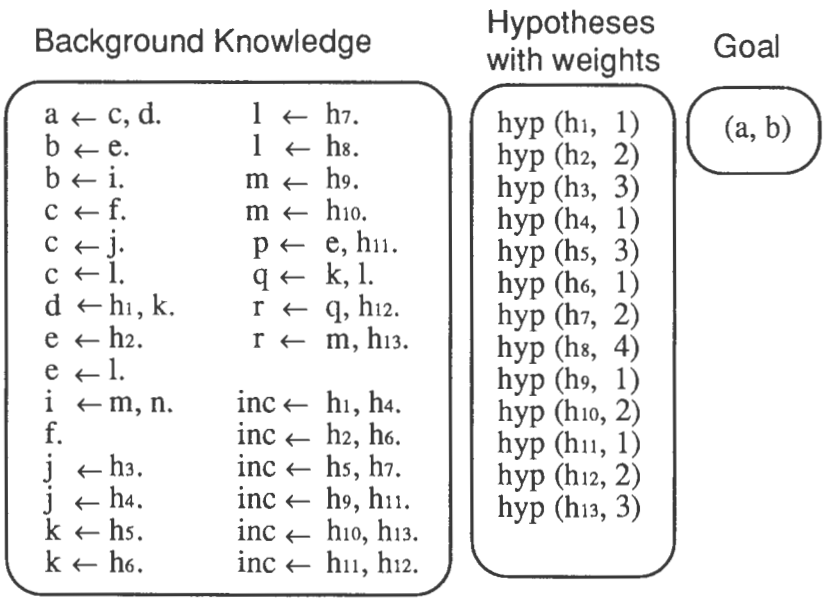


Fig. 3 An example of knowledge-base with hypotheses and an inference goal to be satisfied.

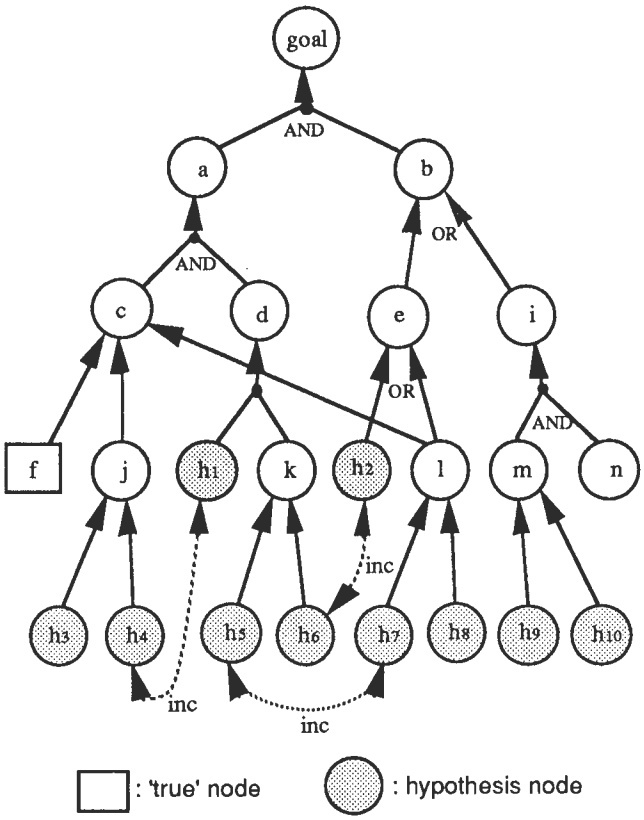


Fig.4 Agoal-directed initial inference-path network.

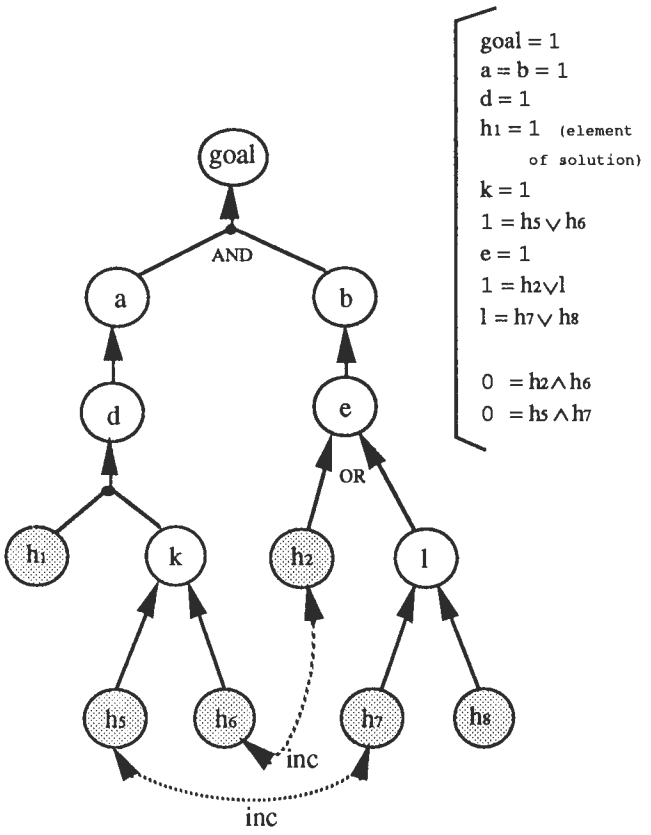


Fig. 5 A simplified inference-path network and extracted constraint Boolean equations (right side).

'i' also becomes 'false' state because it has this node 'n' as its AND-connected child node. Thus, at node 'b' having this node 'i' as its OR-connected child node, we don't need to consider node 'i' any more in the synthesis process of necessary hypotheses and need to consider only child node 'e'. As a result, a simplified inference-path network shown in Fig.5 is obtained.

Moreover, since it is a constraint that the goal becomes 'true' if a solution exists, we can determine the state of AND-connected child nodes by propagating this constraint downward. For example, in the inference-path network of Fig.5, nodes 'a' and 'b' are required to be 'true'; as a consequence, node 'd', node 'k' and hypothesis node 'h_1' are also required to be 'true'. In the same manner, it is necessary for node 'e' to be 'true'. Thus we can obtain simplified constraint equations shown in the right side of Fig.5. Here, 'h_1=1' means that the element hypothesis 'h_1' should be adopted to satisfy the given goal; otherwise, the goal cannot be satisfied. We can thus reduce the number of variables for 0-1 integer programming.

The extraction of relevant inconsistency knowledge can be performed by selecting only inconsistency knowledge in which every body atom is appeared in the simplified inference-path network as hypothesis or intermediate node. We can ignore other inconsistency knowledge in the case of the given goal.

By the above procedures, the simplified constraint (Boolean) equations shown in the right side of Fig.5 are obtained for the case of Fig.3. Excluding the variables already determined to 0 or 1, we can have simplified constraint equations with only five variables as,

$1 = h_5 \vee h_6$,
$1 = h_2 \vee h_7 \vee h_8$,
$0 = h_2 \wedge h_6$,
$0 = h_5 \wedge h_7$.

We transform these constraint equations into linear inequalities according to the method described in section 3. The objective function z for integer programming can be set based on the weights of the element hypotheses, for example, in this case as,

$z = h_1 + 2h_2 + 3h_5 + h_6 + 2h_7 + 4h_8$
$= 1 + 2h_2 + 3h_5 + h_6 + 2h_7 + 4h_8$,

Then, we can apply 0-1 integer programming for solving the weight-based hypothetical reasoning.

5. Applying Exact and Approximate Solution Methods of 0-1 Integer Programming and their Evaluation

For solving 0-1 integer programming, we have applied two exact methods and one approximate method, i.e., 1) all integer method, 2) implicit enumeration method [Balas

65], and 3) pivot and complement method [Balas 80]. See [Greenberg 71, Garfinkel 72, Konno 81], for example, for general discussion on integer programming methods.

There are two main approaches in the exact solution methods for 0-1 integer programming; they are cutting plane method and branch-and-bound method. The all integer method is a variant of the cutting plane method. The implicit enumeration method is based on the branch-and-bound method. Both methods partially employ an efficient linear programming method, i.e., simplex method in their processes. It is recognized in general that the implicit enumeration method is the most efficient among currently available exact solution methods.

On the other hand, the pivot and complement (P&C) method is an efficient approximate solution method for finding a near-optimal solution in polynomial time. Integer constraint is first relaxed in this method to find an optimal solution in real domain by employing the simplex method. Then, by repeating the change of bases (pivot operation) so as to decrease the degree of non-integer index and by rounding the assignments to variables into 0 or 1 while allowing the slight increase of the objective function value, the P&C method finds a feasible integer solution. In the next step, the P&C method executes a local search around this feasible integer solution for finding a better 0-1 integer solution (complement operation).

For example, the problem illustrated in section 4 and in Figs.3-5 becomes to the following 0-1 integer programming problem after removing unnecessary equations because of already determined variables.

$$h_5 + h_6 \geq 1$$
$$h_2 + h_7 + h_8 \geq 1$$
$$-h_2 - h_6 \geq -1$$
$$-h_5 - h_7 \geq -1$$

where

$$h_2, h_5, h_6, h_7, h_8 \in \{0,1\},$$

and the objective function

$$z = 1 + 2h_2 + 3h_5 + h_6 + 2h_7 + 4h_8$$
$$\Rightarrow \text{ minimize.}$$

In this example, together with already determined element hypothesis h_1, a final solution will be obtained as (h_1, h_6, h_7) with the minimum value 4 of the objective function.

Hypothetical reasoning systems employing above-described 0-1 integer programming methods have been implemented in C language and their performance was evaluated. Fault diagnosis problems of digital circuits were used as examples in experiments. The CPU time on Sun4/370 was measured with respect to several sizes of the digital circuits. The CPU time here includes the extraction of relevant knowledge, its simplification, transformation into inequalities, and 0-1 integer programming. Table 1 shows the experimental results.

The time expressed as simplification in Table 1 is the time spent for the extraction of knowledge relevant to a given goal and its simplification; this is within 0.1 sec in the used examples and is very fast. Figure 6 depicts these measurements against the number of possible

Table 1 Experimental performance of hypothetical reasoning systems employing 0-1 integer programming methods. (- indicates the cases that a solution was not obtained in a pre-determined time limit.)

Example No.	Number of Element Hypotheses Before Simplificatin → After Simplification	CPU time [sec]				Value of Objective Function	
		Simpli-fication	All Integer Method	Implicit Enumeration Method	Pivot and Complement Method	Optimal Value	Value by Pivot and Complement Method
(1)	15→10	0.01	0.02	0.01	0.04	9	9
(2)	30→24	0.02	75.53	9.37	2.25	14	14
(3)	45→38	0.03	-	1301.04	12.01	19	19
(4)	60→52	0.06	-	-	35.77	?	24
(5)	75→66	0.09	-	-	87.57	?	29
(6)	15→7	0.01	0.01	0.01	0.02	10	11
(7)	30→24	0.02	9.73	40.60	1.63	15	15
(8)	45→38	0.04	13094.15	8172.21	11.86	20	20
(9)	60→52	0.05	-	-	35.27	?	25
(11)	15→11	0.01	0.02	0.10	0.08	9	9
(12)	30→25	0.02	83.75	55.03	1.98	14	15
(13)	45→39	0.03	-	8598.74	11.96	19	21
(14)	60→53	0.07	-	-	50.18	?	25
(16)	15→11	0.01	0.03	0.05	0.12	4	7
(17)	30→25	0.02	2.56	18.15	2.16	8	15
(18)	45→39	0.04	1850.41	1877.5	12.01	12	17
(19)	60→53	0.06	-	-	39.67	?	25

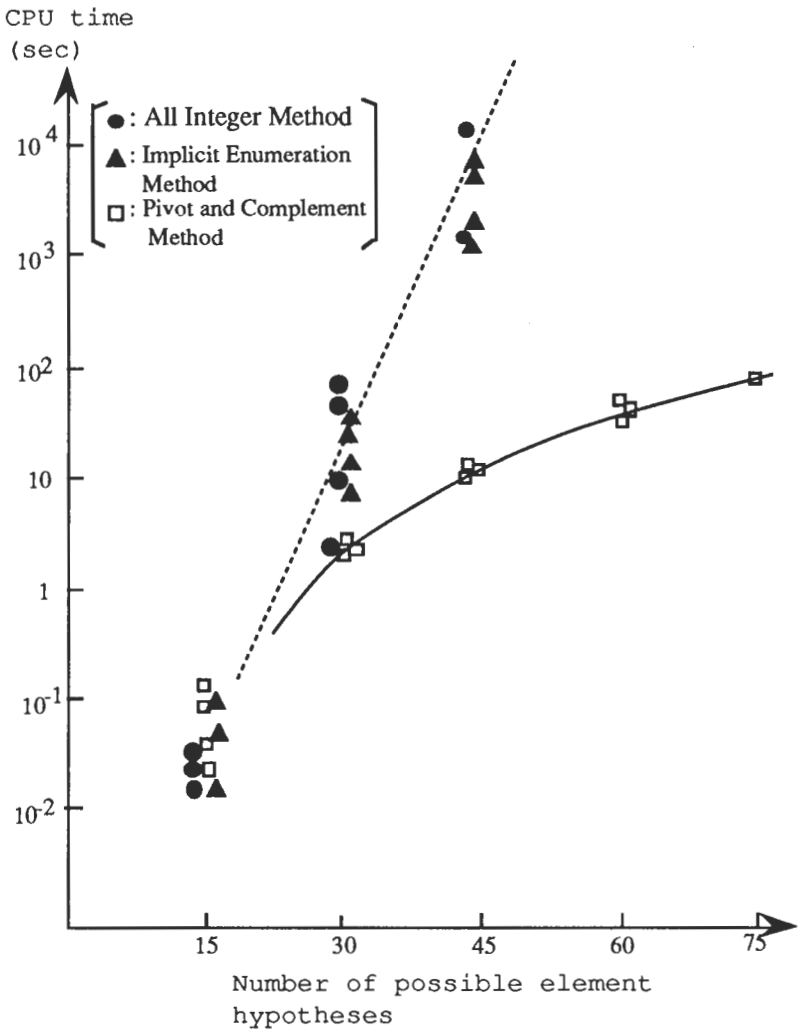


Fig. 6 Inference time (CPU time) of hypothetical reasoning systems employing 0-1 integer programming methods.

element hypotheses which indicates the scale of knowledge-base. Since the 0-1 integer programming is NP-complete problem, the all integer method and the implicit enumeration method which are exact solution methods show exponential-time performance against the scale of knowledge-base, though these methods were applied after the preprocessing including the extraction of relevant knowledge, etc.

On the other hand, Table 1 and Fig.6 show that the pivot and compliment (P&C) method which is an efficient approximate solution method can find a near-optimal solution in polynomial time. The regression analysis of the experimental data reveals that the CPU time is approximately 4.7th power of the number of possible element hypotheses in the knowledge-base. The obtained solutions coincide with the optimal solutions in many cases, or are good near-optimal solutions as seen in the original paper [Ballas 80] and Table 1. (It appears that the solution by the P&C method for example No.17 in Table 1 is not good, since its objective function value is 8 whereas the value of the optimal solution is 15. It becomes clear, however, by a detailed analysis that this approximate solution is the second best solution in this case.)

As seen in the above experiments, the pivot and complement (P&C) method allows a practical polynomial-time hypothetical reasoning. It is, however, not necessarily suitable from its algorithm for the following cases.

- The constraints in 0-1 integer programming is strong.
- The optimal real-number solution and the optimal integersolution are located far away to each other.

In other words, since this method emphasizes the near-optimality of the solution rather than reliably finding a feasible solution, there are cases that the method fails to find a 0-1 solution even if the solution exists. There may be possibilities of improving its performance by considering each problem structure particularly in the hypothetical reasoning problem.

6. Conclusions

This paper has presented a polynomial-time hypothetical reasoning which employs an approximate solution method of 0-1 integer programming. A preprocessing including the extraction of restricted knowledge relevant to a given goal and its simplification is incorporated to effectively reduce the number of variables of 0-1 integer programming. Unlike existing probabilistic search methods such as simulated annealing(SA) and genetic algorithm(GA), and other local search methods such as the heuristic repair method [Minton 92], GSAT [Selman 93a, 93b] and Gu's method based on the unconstrained nonlinear programming [Gu 93], the salient feature of our method is the efficient local search arround the optimal real-domain solution obtained by the efficient simplex method. A further improvement may be possible if we take account of a specific knowledge structure of the hypothetical reasoning.

References

[Balas 65] E. Balas: An Additive Algorithm for Solving Linear Programs with Zero-One Variables, Opsearch, Vol.13, pp.517-546 (1965)

[Balas 80] E. Balas and C. Martin: Pivot and Complement -- A Heuristic for 0-1 Programming, Management Science, Vol.26, pp.86-96 (1980).

[Bylander 89] T. Bylander, D. Allemang, et al.: Some Results Concerning the Complexity of Abduction, Proc. Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR'89), (1989).

[Charniak 90] E. Charniak and S. Shimony: Probabilistic Semantics for Cost Based Abduction, Proc. AAAI'90 (1990).

[Charniak 92] E. Charniak and E. Santos Jr.: Dynamic MAP Calculation for Abduction, Proc. AAAI'92 (1992).

[Clark 78] K. L. Clark: Negation as Failure, in Logic and Databases (H. Gallaire and J. Minker (eds.)), Plenum Press, N.Y., pp.293-322 (1978).

[deKleer 86] J. deKleer: An Assumption-based TMS, Artifi. Intelli., Vol.28, pp.127-167 (1986).

[Dhar 90] V. Dhar and N. Ranganathan: Integer Programming vs. Expert Systems: An Experimental Comparison, Comm. ACM, Vol.33, No.3, pp.323-336 (1990).

[Dowling 84] W. F. Dowling and J. H. Gallier: Linear-time Algorithm for Testing the Satisfiability of Propositional Horn Formulae, Jour. of Logic Programming, Vol.3, pp.267-284 (1984).

[Garfinkel 72] R. Garfinkel and G. L. Nemhauser: Integer Programming, Jon Wiley and Sons (1972)

[Goldberg 89] D. E. Goldberg: Genetic Algorithm in Search, Optimization & Machine Learning, Addison-Wesley (1989).

[Greenberg 71] H. Greenberg: Integer Programming, Jon Wiley and Sons (1971)

[Gu 93] J. Gu: Local Search for Satisfiability (SAT) Problem, IEEE Tran. SMC, Vol.23, No.4, pp.1108-1129 (1993).

[Hooker 88] J. N. Hooker: A Quantitative Approach to Logic Inference, Decision Support Systems, Vol.4, No.1, pp.45-69 (1988).

[Ishizuka 90] M. Ishizuka and T. Matsuda: Knowledge Acquisition Mechanisms for a Logical Knowledge Base including Hypotheses, Knowledge-Based Systems, Vol.3, No.2, pp.77-86 (1990).

[Ishizuka 91] M. Ishizuka and F. Ito: Fast Hypothetical Reasoning System using Inference-Path Network, Proc. Int'l Conf. on Tools for AI (TAI'91), San Jose (1991).

[Ishizuka 93] M. Ishizuka and A. Abe: Fast Hypothetical Reasoning using Analogy on Inference-path Networks, Proc. Int'l Conf. on Tools with AI (TAI'93), Boston (1993).

[Ito 91] F. Ito and M. Ishizuka: Fast Hypothetical Reasoning System using Inference-Path Network (in Japanese), Jour. Japanese Soc. for AI, Vol.6, No.4, pp.501-509 (1991).

[Kirkpatrik 83] S. Kirkpatrik, et al.: Optimization by Simulated Annealing, Science, No.220, pp.671-681 (1983).

[Kondo 91] A. Kondo, T. Makino and M. Ishizuka: An Efficient Hypothetical Reasoning System for Predicate-logic Knowledge-base, Proc. Int'l Conf. on Tools for AI (TAI'91), San Jose (1991).

[Konno 81] H. Konno: Integer Programming (in Japanese), Sangyo-Tosho (1986)

[Makino 90] T. Makino and M. Ishizuka: A Hypothetical Reasoning System with Constraint Handling Mechanism and its Application to Circuit-Block Synthesis, Proc. PRICAI'90, Nagoya (1990).

[Minton 92] S. Minton, et. al.: Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, Artif. Intelli., Vol.58, pp.161-205 (1992).

[Poole 87] D. Poole, R. Aleliunas and R. Goebel: Theorist: A Logical Reasoning System for Defaults and Diagnosis, in The Knowledge Frontier: Essays in The Knowledge Representation (N. J. Cercone and G. McCalla (eds.)), Springer-Verlag, N.Y. (1987).

[Poole 88] D. Poole: A Logical Framework for Default Reasoning, Artif. Intelli., Vol.36, pp.27-47 (1988).

[Selman 93a] B. Selman and H. Kautz: An Empirical Study of Greedy Search for Satisfiability Testing, Proc. AAAI-93 (1993).

[Selman 93b] B. Selman and H. Kautz: Domain-Independent Extensions to GSAT: Solving Large Stractured Satisfiability Problems, Proc. IJCAI-93 (1993).

[Tsuruta 91] S. Tsuruta and M. Ishizuka: A Compiling Method of Propositional Knowledge Base for Abductive Generation of Lacked Knowledge (in Japanese), Jour. Japanese Soc. for AI, Vol.6, No.1, pp.117-123 (1991).

[Tsuruta 92] S. Tsuruta and M. Ishizuka: A Compiling Method of Predicate Knowledge Base for Efficient Abductive Hypothesis Synthesis (in Japanese), Jour. Japanese Soc. for AI, Vol.7, No.1, pp.130-137 (1992).

A Logical Language for Natural Language Processing

Syed S. Ali
Department of Computer Science
Southwest Missouri State University
901 South National Avenue
Springfield, MO 65804
ssa231f@cnas.smsu.edu

Abstract

We present a formal description of a logical language that is based on a propositional semantic network. Variables in this language are not atomic and have potentially complex structure. We start from the individual components of a semantic network system, atomic nodes and relations that connect nodes, and provide a complete specification for the structure of nodes and a subsumption procedure between nodes. We differ from other work in subsumption in that the representation language is uniform and based on an extended first-order predicate logic. The language is particularly suitable for addressing some problems associated with natural language processing, namely the representation of complex natural language descriptions and inference associated with description subsumption.

1 Introduction

We present a formal description of a propositional semantic-network-based knowledge representation system. Variables in this representation are not atomic and have potentially complex structure. We start from the individual components of a semantic network system, atomic nodes and relations that connect nodes, and provide a complete specification for the structure of nodes and a subsumption procedure between nodes. We differ from other work in subsumption in that the representation language is uniform and based on a first-order predicate logic. The language is particularly suitable for addressing some problems associated with natural language processing, namely the representation of complex natural language descriptions and inference associated with description subsumption.

Subsumption is a partial ordering on related concepts (nodes in a semantic network) that relates more general concepts to more specific instances of those concepts. The manner in which "more general" is determined characterizes the type of subsumption. Woods has classified subsumption into *extensional, structural, recorded, axiomatic,* and *deduced* subsumption [Woods, 1991]. In general, the more complex the structure of the concepts (and their associated semantics) the more difficult subsumption becomes.

Our formalism embeds a procedure for determining *structural* and *deduced* subsumption between concept nodes in a propositional semantic network representation (one in which propositions are represented by nodes and not arcs of the network). We specify the structure of the nodes of the propositional semantic network system in terms of its components, nodes and relations between nodes, and specify the subsumption procedure in terms of these components. This subsumption procedure can do the type of subsumption inference associated with KL-ONE classification [Brachman and Schmolze, 1985] and its successors (most notablely KRYPTON [Brachman *et al.*, 1985]). This subsumption procedure does this without a distinction between an assertional and terminological component (and their associated difficulties [Beierle *et al.*, 1992]).

Sections 2 and 3 summarize the formal specification of the logical language. Section 4 describes the subsumption procedure in detail. Sections 5 and 6 describe the concept matcher and inference mechanism. These sections provide a framework in which the natural language processing examples of Section 7 can be best understood.

2 Syntax and Semantics of the Logic

We specify the syntax and semantics of a logic whose variables are not atomic and have structure. We call these variables *structured variables.* The syntax of the logic is specified by a complete definition of a propositional semantic network representation formalism (an augmentation of [Morgado, 1986, Shapiro, 1991]). By a propositional semantic network, we mean that all information, including propositions, "facts", etc., is represented by nodes. The implemented system, ANALOG, is used here, for convenience, to refer to the logical system.

2.1 Semantics

As a propositional semantic network formalism, any theory of semantics that ascribes propositional meaning to nodes can be the semantics used in ANALOG. In this paper, examples and representations are used that follow the case frame semantics of [Shapiro and Rapaport,

1987] which provide a collection of propositional case frames and their associated semantics based on an extended first-order predicate logic. We augment that logic further with arbitrary individuals (for the semantics of structured variables) in a manner similar to the semantic theory of [Fine, 1985]. For a more complete specification of the syntax and semantics of ANALOG and its suitability for NLP, see [Ali, 1994].

2.2 The Domain of Interpretation

ANALOG nodes are terms of a formal language. The interpretation of a node is an object in the domain of interpretation, called an entity. Every ANALOG node denotes an entity, and if n is an ANALOG node, then $[\![n]\!]$ denotes the entity represented by n. Nodes are atomic or structured. In the latter case, they are connected to other nodes by labelled arcs. The labels on arcs are called "relations". It is useful, for discussing the semantics of ANALOG networks, to present them in terms of an "agent". Said agent has beliefs and performs actions, and is actually a model of a cognitive agent. In the rest of this explication, the term "node" will be used for ANALOG node, and "relation" for ANALOG relation.

2.3 Metapredicates

To help formalize this description we introduce the metapredicates *Conceive*, *Believe*, and $=$. If n, n_1, n_2 are metavariables ranging over nodes, and p is a metavariable ranging over proposition nodes, the semantics of the metapredicates listed above are:

- *Conceive*(n) Means that the node is actually constructed in the network. *Conceive*(n) may be true without $[\![n]\!]$ being known to be true or false. Also note that n need not be a proposition.
- *Believe*(p) Means that the agent believes the proposition $[\![p]\!]$.
- $n_1 = n_2$ Means that n_1 and n_2 are the same, identical, node.

Belief implies conception, as specified in Axiom 1.

Axiom 1: $Believe(p) \Rightarrow Conceive(p)$

In practical terms, this means that for a proposition to have a belief status it must be a node in the semantic network. This is a reasonable assumption since for an agent to believe something the agent must have some conceptualization for it.

2.4 Definition of Nodes

Informally, a node consists of a set of labeled (by relations) directed arcs to one or more nodes. Additionally, a node may be labeled by a "name" (e.g., `BILL`, `M1`, `V1`) as a useful (but extra-theoretic) way to refer to the node. This naming of a proposition node is of the form `M`n, where n is some integer. A "!" is appended to the name to show that the proposition represented by the node is believed to be true (*Believe*(`M1`)). However, the "!" does not affect the identity of the node or the proposition it represents. Similarly, variable nodes are labeled `V`n, where n is some integer, and base nodes are named `B`n, where n is some integer (additionally, base nodes may be named for the concept they represent, e. g., `man`). More formally a node is defined as follows:

Definition 1: There is a non-empty collection of labelled atomic nodes called **base nodes**. Typically, base nodes are mnemonically labelled to indicate the entity they denote. *Example:* `bill` is a base node.

Definition 2: A **wire** is an ordered pair $\langle r, n \rangle$, where r is a relation, and n is a node. Metavariables $w, w_1, w_2, \ldots$ range over wires. *Example:* <`member`, `john`> is a wire.

Definition 3: A **nodeset** is a set of nodes, $\{n_1, \ldots, n_k\}$. Meta-variables $ns, ns_1, ns_2, \ldots$ range over nodesets. *Example:* {`john, bill`} is a nodeset if `john` and `bill` are nodes.

Definition 4: A **cable** is an ordered pair $\langle r, ns \rangle$, where r is a relation, and ns is a non-empty nodeset. Meta-variables $c, c_1, c_2, \ldots$ range over cables. *Example:* <`member`, {`john, bill`}> is a cable.

Definition 5: A **cableset** is a non-empty set of cables, $\{\langle r_1, ns_1 \rangle, \ldots, \langle r_k, ns_k \rangle\}$, such that $r_i = r_j \iff i = j$. Meta-variables $cs, cs_1, cs_2, \ldots$ range over cablesets. *Example:* {<`member`, {`john, bill`}>, <`class`, {`man`}>} is a cableset.

Definition 6: Every node is either a base node or a cableset. *Example:* `bill` is a base node, {<`member`, {`john, bill`}>, <`class`, {`man`}>} is a cableset.

Definition 7: We overload the **membership** relation "$\in$" so that $x \in s$ holds just under the following conditions:

1. If x is a node and s is a nodeset, $x \in s \iff \exists y\, [y \in s \wedge Subsume(y, x).]$
   Example: `M1` $\in$ {`M1, M2, M3`}
2. If x is a wire such that $x = \langle r_1, n \rangle$, and s is a cable such that $s = \langle r_2, ns \rangle$, then
   $x \in s \iff r_1 = r_2 \wedge n \in ns$.
   Example:
   <`member`, `john`> $\in$ <`member`, {`john, bill`}>
3. If x is a wire and s is a cableset, then $x \in s \iff \exists c[c \in s \wedge x \in c]$.
   Example: <`member`, `john`> $\in$ {<`member`, {`john, bill`}>, <`class`, {`man`}>}

Because we need more definitions before *Subsume* can be defined, we defer its definition to Figure 2.

Definition 8: An **nrn-path** from the node n_1 to the node n_{k+1} is a sequence, $n_1, r_1, \ldots, n_k, r_k, n_{k+1}$, for $k \geq 1$ where the n_i are nodes, the r_i are relations, and for each i, $\langle r_i, n_{i+1} \rangle \in n_i$. *Example:* If `M1` = {<`member`, {`john, bill`}>, <`class`, {`man`}>}, then

In the following model (A, B, M, R, U, E, Γ):

A ={relation, arg}, B ={bill, john, ted}, M ={M1, M2, M3}, Γ ={M1, M2, M3}

where:

M1 = {<relation, {brothers}>, <arg, {bill, john, ted}>}
M2 = {<relation, {brothers}>, <arg, {john, ted}>}
M3 = {<relation, {brothers}>, <arg, {bill, john}>}

Some reductions: *Reduce*(M2, M1), and *Reduce*(M3, M1)

Figure 1: Example of Subsumption by Reduction for a Particular Model

M1,member,john and M1,class,man are some nrn-paths.

Definition 9: A node n_1 **dominates** a node n_2 just in case there is an nrn-path from n_1 to n_2. The predicate *dominate*(n_1, n_2) is true if and only if n_1 dominates n_2. *Example:* If M1 = {<member, {john, bill}>, <class, {man}>}, then M1 dominates john, bill, and man.

Definition 10: A **variable** node is a cableset of the form {<*any*, *ns*>} (**universal variable node**) or {<*some*, ns_1>, <*depends*, ns_2>} (**existential variable node**). Additionally, a node is a variable node only if:

1. If it has the form {<*any*, *ns*>}, then every $n \in ns$ *must* dominate it.
2. If it has the form {<*some*, ns_1>, <*depends*, ns_2>}, then every $n \in ns_1$ *must* dominate it *and* every $n \in ns_2$ *must* be a universal variable node.

Example: V1 = {<any, {{<member, {V1}>, <class, {man}>}}>} is the variable node corresponding to *every man*. The variable label V1 is just a convenient extra-theoretic method of referring to the variable.

We define two selectors for variable nodes:

$$rest(v) = \begin{cases} ns & \text{if } v = \{<any, ns>\} \\ ns_1 & \text{if } v = \{<some, ns_1>, <depends, ns_2>\} \end{cases}$$

$$depends(v) = ns_2 \quad \text{if } v = \{<some, ns_1>, <depends, ns_2>\}$$

Informally, *rest*(v) is the set of restriction propositions on the types of things that may be bound to the variable node v. *Depend*(v) is the set of universal variable nodes on which the existential variable node, v, is scope-dependent.

Definition 11: A **molecular** node is a cableset that is *not* a variable node. *Example:* {<member, {john, bill}>, <class, {man}>} is a molecular node, since it is a cableset but not a variable node.

Definition 12: A **rule** node is a molecular node that dominates a variable node that does not, in turn, dominate it. *Example:*
Given the labelled nodes below:
V1 = {<any, {M1}>}
M1 = {<member, {V1}>, <class, {man}>}
M2 = {<member, {V1}>, <class, {mortal}>}
M2 is a rule node since M2 dominates V1, which does not, in turn, dominate M2. M1 is *not* a rule node because, while it dominates V1, it is also dominated by V1. The non-rule nodes that dominate variable nodes correspond to restrictions on binders of those same variable nodes.

Definition 13: A **wireset** of a node n is defined as $\{w | w \in n\}$. *Example:* If M2 = {<member, {john, bill}>, <class, {man}>} then: *wireset*(M2) = {<member, john>, <member, bill>, <class, man>}.

2.5 The ANALOG model

Definition 14: An **ANALOG model** is a tuple (A, B, M, R, U, E, Γ) where A is a set of relations, B is a set of base nodes, M is a set of non-rule molecular nodes, R is a set of rule nodes, U is a set of universal variable nodes, and E is a set of existential variable nodes, and $\Gamma \subseteq M \cup R$. B, M, R, U, and, E are disjoint. Γ consists of the set of asserted nodes.

2.6 Reduction

We follow [Shapiro, 1986, Shapiro, 1991] in arguing for a form of reduction inference (defined in Axioms 2 and 3 below) as being useful. This is a form of structural subsumption [Woods, 1991], peculiar to semantic network formalisms, which allows a proposition to "reduce" to (logically imply) propositions whose wires are a subset of the wires of the original proposition. Figure 1 gives an example of a proposition expressing a brotherhood relation among a group of men. Node M1 represents the proposition that bill, john, ted, and joe are brothers. By reduction subsumption, all proposition nodes (such as M2 and M3) involving fewer brothers follow.

However, we must restrict the use of reduction inference to precisely those propositions and rules which are

reducible through the use of the *IsReducible* metapredicate.

Axiom 2: $Reduce(cs_1, cs_2) \iff (\forall w[w \in cs_2 \Rightarrow w \in cs_1] \wedge IsReducible(cs_1, cs_2))$.

Note that the semantics of the metapredicate *IsReducible* will be specified in terms of the particular case frames used in a representation language. Propositions like `M1` are clearly reducible, but not all propositional case frames are reducible. For example,

$$\forall x((\text{man}(x) \wedge \text{rich}(x)) \Rightarrow \text{happy}(x)) \quad (1)$$

should not allow the reduced proposition:

$$\forall x(\text{man}(x) \Rightarrow \text{happy}(x)) \quad (2)$$

which involves fewer constraints on x than than proposition (1), as the latter does not follow from the former. New propositions derived by reduction should be implied by the propositions from which they are derived. For example note that reduction to proposition (2) is appropriate when the constraints in the antecedent of the rule are disjunctive as in:

$$\forall x((\text{man}(x) \vee \text{rich}(x)) \Rightarrow \text{happy}(x)) \quad (3)$$

IsReducible should be define appropriately for the particular representation language to allow (or disallow) these reductions. In Section 3.3 we specify some of the reducible case frames we use in this paper.

A proposition that is a reducible reduction of a believed proposition is also a believed proposition. Since nodes are, by definition, cablesets, we state this as in Axiom 3.

Axiom 3: $(Reduce(n_1, n_2) \wedge Believe(n_1)) \Rightarrow Believe(n_2)$

2.7 Types of Nodes

We have defined four types of nodes: base, molecular, rule, and variable nodes. Informally, base nodes correspond to individual constants in a standard predicate logic, molecular nodes to sentences and functional terms, rule nodes to closed sentences with variables, and variable nodes to variables. Note that syntactically all are terms in ANALOG, however.

3 Semantic Issues

ANALOG is specified in terms of a propositional semantic network. By this is meant that all information, including propositions, "facts", etc., is represented by nodes. Labelled arcs are purely structural and carry no assertional import. The benefit of representing propositions by nodes is that propositions about other propositions may be represented. This means that ANALOG is not strictly first-order as variables may quantify over nodes that correspond to propositions or predicates rather than individuals. This has useful consequences when processing natural language, particularly questions whose answers are propositions, in that *any* node (including non-base nodes) can be bound to the variable associated with a question.

3.1 Intensional Representation

ANALOG nodes correspond to intensional entities. What is being represented, in ANALOG, is an agent's mind. Objects in that mind need not represent any extensional object in the world. To connect objects of mind to extensional objects in the world, ANALOG uses a case frame with a `lex` arc to the object of thought denoting its extension. Additionally, sensory nodes can also make this connection to the external world.

3.2 The Uniqueness Principle

No two nodes in the network represent the same individual, proposition, or rule.

Axiom 4: $n_1 = n_2 \iff [\![n_1]\!] = [\![n_2]\!]$

This is a consequence of the intensional semantics, since nodes correspond to objects of thought in an agent's mind. The objects of thought are intensional: a mind can have two or more objects of thought that correspond to only one extensional object, or no extensional object. The classic example of this is the Morning Star and the Evening Star which might be distinct objects of thought, but have a single extensional referent. Thus the nodes representing the Morning Star and the Evening Star are distinct, and any node can denote only *one* intensional object which no other node can denote [Maida and Shapiro, 1982].

A benefit of this is a high degree of structure-sharing in large networks. Additionally, the network representation of some types of sentences can reflect the re-use of natural language terms expressed by pronouns and other reduced forms.

3.3 Case Frame Semantics

ANALOG can support any propositional representations that have a consistent syntax and semantics. In this paper, examples of representations used will follow the syntax and semantics of [Shapiro and Rapaport, 1987]. Here we describe only two case frames (those used in the brothers example of Figure 1), due to space limitations. We specify their syntax, semantics, and status for reduction.

1. $\{<member, ns_1>, <class, ns_2>\}$: For any $n_1 \in ns_1$, and any $n_2 \in ns_2$, $[\![n_1]\!]$ is a member of class $[\![n_2]\!]$. Valid reductions are specified by:

   $IsReducible(\{<member, ns_1 \cup ns_3>, <class, ns_2 \cup ns_4>\}, \{<member, ns_1>, <class, ns_2>\})$

   where $ns_1 \neq \{\}$, and $ns_2 \neq \{\}$.

2. $\{<relation, ns_1>, <arg, ns_2>\}$: For every $n_1 \in ns_1$, and every $n_2, n_3 \in ns_2, n_2 \neq n_3$, $[\![n_2]\!]$ is in relation $[\![n_1]\!]$ to $[\![n_3]\!]$, and $[\![n_3]\!]$ is in relation $[\![n_1]\!]$ to $[\![n_2]\!]$. Valid reductions are specified by:

   $IsReducible(\{<relation, ns_1 \cup ns_3>, <arg, ns_2 \cup ns_4>\}, \{<relation, ns_1>, <arg, ns_2>\})$

   where $ns_1 \neq \{\}$, and $|ns_2| \geq 2$

Subsume(x, y) in a model $(A,\ B,\ M,\ R,\ U,\ E,\ \Gamma)$ if and only if, one of:

1. $x = y$.
2. *Reduce*(x, y)
3. For $x \in U$ and $y \in B \cup M \cup R$, if not *occurs-in*$(x,\ y)$ and there exists a substitution S such that

$$\forall r[r \in rest(x),\ \Gamma \vdash r\{y/x\} \cdot S].$$

Logical derivation is here denoted by "$\vdash$."

4. For $x \in U$ and $y \in U \cup E$, if

$$\forall r[r \in rest(x) \Rightarrow \exists s[s \in rest(y) \wedge Subsume(r, s)]]$$

5. For $x, y \in E$, if all of the following hold:

$$\forall s[s \in rest(y) \Rightarrow \exists r[r \in rest(x) \wedge Subsume(r, s)]]$$
$$\forall r[r \in rest(x) \Rightarrow \exists s[s \in rest(y) \wedge Subsume(r, s)]]$$
$$\forall d[d \in depends(y) \Rightarrow \exists c[c \in depends(x) \wedge Subsume(c, d)]]$$

Figure 2: Subsumption Procedure

Note that most of the valid reductions of these propositional case frames follow directly from their semantics. There are numerous other case frames that are useful and necessary (for a complete dictionary of them, see [Shapiro *et al.*, 1993]).

4 Subsumption

Semantic network formalisms provide "links" that relate more general concepts to more specific concepts; this is called a *taxonomy*. It allows information about concepts to be associated with their most general concept, and it allows information to filter down to more specific concepts in the taxonomy via inheritance. More general concepts in such a taxonomy *subsume* more specific concepts, the subsumee inheriting information from its subsumers. For atomic concepts, subsumption relations between concepts are specified by the links of the taxonomy. We specify subsumption for non-atomic concepts, below.

Definition 15: A **binding** is a pair v/u, where either u is a universal SV and v is any node or u and v are both existential nodes. *Examples:* `V1/V2`, `JOHN/V1`.

Definition 16: A **substitution** is a (possibly empty) set of bindings, $\{t_1/v_1, \ldots, t_n/v_n\}$. *Examples:* {`V1/V2, JOHN/V3`}, {`B1/V1, M1/V2`}.

Definition 17: The result of **applying** a substitution, $\theta = \{t_1/v_1, \ldots, t_m/v_m\}$, to a node n is the instance $n\theta$ of n obtained by simultaneously replacing each of the v_i dominated by n with t_i. If $\theta = \{\}$, then $n\theta = n$. *Example:* If `M1 = {<member, {V1}>, <class, {MAN}>}` then: `M1{JOHN/V1} = {<member, {JOHN}>, <class, {MAN}>}`

Definition 18: Let $\theta = \{s_1/u_1, \ldots, s_n/u_n\}$ and $\rho = \{t_1/v_1, \ldots, t_m/v_m\}$ be substitutions. Then the **composition** $\theta \cdot \rho$ of θ and ρ is the substitution obtained from the set: $\{s_1\rho/u_1, \ldots, s_n\rho/u_n, t_1/v_1, \ldots, t_m/v_m\}$ by deleting any binding $s_i\rho/u_i$ for which $u_i = s_i\rho$. *Example:* If $\theta =$ {`V1/V2, V4/V3`}, and $\rho =$ {`V2/V1, JOHN/V4`} then $\theta \cdot \rho =$ {`JOHN/V3, JOHN/V4`}.

Definition 19: A substitution, θ, is **consistent** iff neither of the following hold:

$$\exists u, t, s[t/u \in \theta \wedge s/u \in \theta \wedge s \neq t]$$
$$\exists u, v, t[t/u \in \theta \wedge t/v \in \theta \wedge u \neq v]$$

We note that this is different from the standard definition of consistency for substitutions. A substitution that is not consistent is termed **inconsistent**. The motivation for the second constraint (called the unique variable binding rule, UVBR) is that in natural language, users seldom want different variables in the same sentence to bind identical objects [Shapiro, 1986]. For example, *Every elephant hates every elephant* has a different interpretation from *Every elephant hates himself*. Typically, the most acceptable interpretation of the former sentence requires that it not be interpreted as the latter. UVBR requires that within an individual sentence that is a rule (has bound variables), any rule use (binding of variables) must involve different terms for each variable in the rule to be acceptable. *Examples:*

{`JOHN/V2, BILL/V2`} is inconsistent.
{`JOHN/V1, JOHN/V2`} is inconsistent.
{`JOHN/V1, BILL/V2`} is consistent.

Definition 20: The predicate *occurs-in*(x, y) where x is a variable is defined: *occurs-in*$(x, y) \iff$ *dominate*(y, x). *Occurs-in* is used for the standard occurs check of the unification algorithm (and is just a more perspicacious naming of *dominate*).

In the following model (A, B, M, R, U, E, Γ):

A ={member, class, any}, B ={man, mortal, Socrates}, M ={M1, M3, M4}, R ={M2}, U ={V1}

where:

M1 = {<member, {V1}>,<class, {man}>}, M2 = {<member, {V1}>,<class, {mortal}>}
M3 = {<member, {Socrates}>,<class, {man}>}, M4 = {<member, {Socrates}>,<class, {mortal}>}
V1 = {<any, {M1}>}

The resulting subsumption: *Subsume*(M2, M4).

Figure 3: Example of Subsumption for a Particular Model

In ANALOG, we specify subsumption as a binary relation between arbitrary nodes in the network. We define subsumption between two nodes x and y in Figure 2. This definition of subsumption includes subsumption mechanisms that Woods classifies as *structural, recorded, axiomatic,* and *deduced* subsumption [Woods, 1991]. In Figure 2, case (1) corresponds to identical nodes (a node, obviously, subsumes itself). Case (2) is the reduction inference case discussed in section 2.6. Case (3) applies when a universal structured variable node subsumes another node. This corresponds to a description like *any rich man* subsuming *John* if *John* is known to be a man and rich. Such a variable will subsume another node if and only if every restriction on the variable can be derived (in the current model) for the node being subsumed. Subsumption, consequently, requires derivation. For the examples shown here, standard first-order logical derivation may be assumed. Case (4) allows a more general universal variable node to subsume a less general existential variable node. For this to happen, for every restriction in the universal variable node there must be a restriction in the existential variable node, and the former restriction must subsume the latter restriction. For example, the variable node corresponding to *every rich girl* would subsume *some rich happy girl* (but not *some girl*). Case (5) allows one existential variable node to subsume another. The requirement for this case is, essentially, that the variables be notational variants of each other, except for those universal structured variables they are scope dependent upon. This is because it is not, in general, possible for any existential variable to subsume another except when they are structurally identical. The reason this case is needed (rather than just excluding it entirely) is that for a rule node corresponding to *every boy loves some girl* to subsume *every rich boy loves some girl*, the existential variable node corresponding to the *some girl* in the first rule node must subsume the existential variable node corresponding to the *some girl* in the second rule node.

In Figure 3, a more detailed example for a particular model is given. Node M2 represents the proposition that *all men are mortal*, M3 the proposition that Socrates is a man, and M4 the proposition that Socrates is mortal. V1 is the structured variable representing any man. It then follows that M4 is a special case of M2 directly by subsumption, since V1 subsumes Socrates. Note that the restrictions on subsumption involving variables is stricter than *Reduce*, which only requires that the wires of one node be a subset of the other.

As with reduction (Axiom 3), a proposition that is subsumed by a believed proposition is also a believed proposition. This can be stated as a more general form of Axiom 3.

Axiom 5: $(Subsume(n_1, n_2) \wedge Believe(n_1)) \Rightarrow Believe(n_2)$

Axiom 5 allows the sorts of commonsense description subsumption evident in natural language.

5 Node Matching

Matching in ANALOG is the process of determining the most general common instance (MGI) of two nodes such that one instance subsumes the other instance. Matching is specified in terms of nodes (and the arcs connecting them) and sets of substitutions (*substitutionsets*) for variables in the nodes.

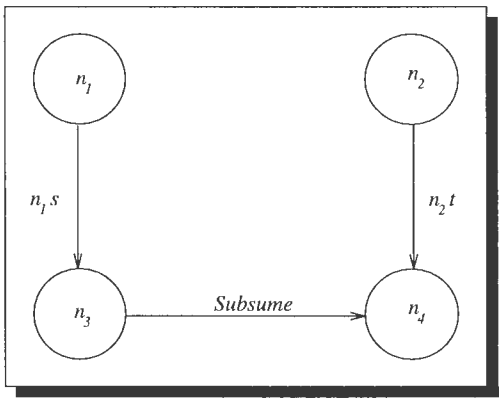


Figure 4: Pictorial View of Node Matching Process

The process looks like Figure 4. When two nodes are being considered to determine a subsumption relationship (n_1 and n_2), the matcher attempts to find two sub-

Algorithm 1 *Match(i, j, S)* in a model $(A,\ B,\ M,\ R,\ U,\ E,\ \Gamma)$

IF $i = j$ THEN (1)
  RETURN S
ELSEIF $i \in U$ and $Subsume(ia_k, jb_k)$ THEN (2)
  RETURN $\{(a_k \cdot \{j/i\}, b_k) \mid (a_k, b_k) \in S \wedge a_k \cdot \{j/i\}$ is consistent.$\}$
ELSEIF $j \in U$ and $Subsume(jb_k, ia_k)$ THEN (3)
  RETURN $\{(a_k, b_k \cdot \{i/j\}) \mid (a_k, b_k) \in S \wedge b_k \cdot \{j/i\}$ is consistent.$\}$
ELSEIF $i, j \in E$ THEN
  IF $Subsume(ia_k, jb_k)$ THEN (4)
    RETURN $\{(a_k \cdot \{j/i\}, b_k) \mid (a_k, b_k) \in S \wedge a_k \cdot \{j/i\}$ is consistent.$\}$
  ELSEIF $Subsume(jb_k, ia_k)$ THEN (5)
    RETURN $\{(a_k, b_k \cdot \{i/j\}) \mid (a_k, b_k) \in S \wedge b_k \cdot \{j/i\}$ is consistent.$\}$
  END
ELSEIF $i, j \in M \cup R$ THEN (6)
  RETURN *matchwires(wireset(i), wireset(j), S)*
ELSE
  RETURN **fail**
END

Algorithm 2 *Matchwires(ws_1, ws_2, S)*

IF $S =$ fail or $ws_1 = \{\}$ THEN
  RETURN **fail**
ELSEIF $ws_2 = \{\}$ THEN
  RETURN S
ELSE

$$S \leftarrow \bigcup_{\substack{<r,n> \in ws_1 \\ <r,m> \in ws_2}} \left[\begin{array}{ll} matchwires(& ws_1 - \{ < r, n > \}, \\ & ws_2 - \{ < r, m > \}, \\ & match(n, m, s)) \end{array} \right] - \{\mathbf{fail}\}$$

  IF $S = \{\}$ THEN
    RETURN **fail**
  ELSE
    RETURN S
  END
END

Figure 5: Match and Matchwires Algorithms

stitutions s and t (called the source and target substitutions, respectively), such that if s is applied to n_1 to produce n_3, and if t is applied to n_2 to produce n_4, then n_3 will subsume n_4. Given two nodes, the matcher returns a set of (s_i, s_j) pairs that are possible substitutions for the two nodes that result in the subsumption relationship. If none is possible, the empty set is returned.

5.1 Match Algorithm

The two-way matching procedure takes a pair of nodes i, j and returns a substitution set consisting of source-target substitutions (s_i, t_i) such that $Subsume(is_i,\ jt_i)$. To account for the possibility of multiple source-target substitutions, match also takes a substitution set which constrains the possible matches (based on previous recursive matchings). *Match* is defined in Algorithm 1 of Figure 5 and works by recursively generating all possible consistent source-target substitutions on a case by case basis. Each case of the match procedure potentially adds a new binding to the source or target substitutions in the substitution set that is the result of the matching.

In Algorithm 1, case (1) will succeed only if the two nodes are identically the same node. This follows from the uniqueness principle. Since this match involves no variables, no new bindings are added in the event of a successful match. This case occurs when matching two terms that share a subterm. This is particularly likely to occur for base nodes, since there will only be one base node for any intensional individual in the network. Cases (2) and (3) deal with attempted matches where one node is a universal variable. If the universal variable node subsumes the other node (subject to the current source and target substitutions), a new binding is added to all the source-target substitution sets and the consistent source-target substitutions become the resulting substitution set. Cases (4) and (5) deal with attempted matchings where both nodes are existential variable nodes. Depending on whether the source subsumes the target or the target subsumes the source (subject to the current

```
(parse -1)
ATN parser initialization...
Input sentences in normal English orthographic convention.
Sentences may go beyond a line by having a space followed by a <CR>
To exit the parser, write ^end.
: Every man owns a car
I understand that every man owns some car.
: Every young man owns a car
I understand that every young man owns some car.
: Every young man that loves a girl owns a car that is sporty
I understand that every young man that loves any girl owns some sporty car.
: Every young man that loves a girl that owns a dog owns a red car that is sporty
I understand that every young man that loves any girl that owns any dog owns some red
sporty car.
: Every young man that loves a girl and that is happy owns a red sporty car that wastes gas
I understand that every young happy man that loves any girl owns some sporty red car that
wastes gas.
: ^end
ATN Parser exits...
```

Figure 6: Examples of complex noun phrase use that correspond to structured variables

source and target substitutions), a new binding is added to all the source or target substitutions and the consistent source-target substitutions become the resulting substitution set. Case (6) is the general case and deals with attempts to match two molecular or rule nodes. Both the source and target nodes are converted into their equivalent wiresets, and all possible matchings of wires are attempted to determine if the source node can match the target node. This is done using the *matchwires* function in Algorithm 2. *Matchwires* works by attempting to find consistent source-target substitutions such that all wires of a source instance's wireset are in the target instance's wireset.

6 Node Inference

The primary form of inference in ANALOG is instantiation, that is, replacing more statements with more specific statements by applying substitutions generated by the matcher. The metarule for this is:

$$\frac{\begin{array}{l}Believe(n_1)\\ Conceive(n_2)\\ (s,t) \in match(n_1, n_2, \{\})\end{array}}{Believe(n_2 t)}$$

In this metarule, n_1 is a node that represents a statement that is believed, and n_2 is a node that is merely conceived, typically corresponding to a question. If match returns a substitution pair that allows the belief of an instance of n_2 this rule allows the system to believe that instance. Because syntactically similar natural language statements are mapped into structurely similar representations (including questions) this rule allows general inference, as illustrated in the next section.

7 ANALOG for NLP

So far, we have motivated some aspects of the logic underlying the ANALOG knowledge representation and reasoning system and formalized some important concepts, such as subsumption, associated with the logical system. At this point, we will attempt to illustrate the utility of the system for natural language processing with specific examples the system modelling natural language use.

ANALOG includes a generalized augmented transition network (GATN) natural language parser and generation component linked up to the knowledge base (based on [Shapiro, 1982]). A GATN grammar specifies the translation/generation of sentences involving complex noun phrases into/from ANALOG structured variable representations.

We present two demonstrations of the NLP component of ANALOG. The first illustrates the representation and use of complex noun phrases, the second is a demonstration that illustrates the use of description subsumption and inference in providing useful answers to questions.

7.1 Representation of Complex Noun Phrases

An apparent advantage of the use of structured variables lies in the representation and generation of complex noun phrases that involve restrictive relative clause complements. The restriction set of a structured variable typically consists of a type constraint along with property constraints (adjectives) and other more complex constraints (restrictive relative clause complements). So, when parsing a noun phrase all processing is localized

```
: Every man is mortal                                                    (1)
I understand that every man is mortal.                                   (2)
: Who is mortal                                                          (3)
Every man is mortal.                                                     (4)
: Is any rich man mortal                                                 (5)
Yes, every rich man is mortal.                                           (6)
: John is a man                                                          (7)
I understand that John is a man.                                         (8)
: Is John mortal                                                         (9)
Yes, John is mortal.                                                     (10)
: Who is mortal                                                          (11)
John is mortal and every rich man is mortal and every man is mortal.     (12)
: Are all rich young men that own some car mortal                        (13)
Yes, every young rich man that owns some car is mortal.                  (14)
: Any rich young man that owns any car is happy                          (15)
I understand that every young rich man that owns any car is happy.       (16)
: Is John happy                                                          (17)
I don't know.                                                            (18)
: Young rich John owns a car                                             (19)
I understand that mortal rich young John owns some car.                  (20)
: Who owns a car                                                         (21)
Mortal rich young John owns some car.                                    (22)
: Is John happy                                                          (23)
Yes, mortal rich young John is happy.                                    (24)
```

Figure 7: Deduced Subsumption with Complex Descriptions.

and associated with building its structured variable representation. When generating a surface noun phrase corresponding to the structured variable, all constraints associated with the variable are part of its structure and can be collected and processed easily. This is in contrast to non-structured variable representations (such as first-order predicate logic) where the restrictions on variables are disassociated from the variables themselves, in the antecedents of rules.

In Figure 6, user input is italicized, the text at the beginning is a standard message and will be omitted from the remaining figure. Figure 6 shows example sentences with progressively more complex noun phrases being used. These noun phrases are uniformly represented using structured variables. Parsing and generation of these noun phrases is simplified because structured variables collect all relevant restrictions on a variable into one unit, a structured variable. The parser parses the user's sentence and builds an ANALOG representation for the user input. The resulting representation is then passed to the generation component, which generates the output response (sometimes prefixed by the canned phrase `I understand that`). Notice how, in Figure 6, the descriptions in the sentences get progressively more complex. Because structured variables collect all constraints on a variable into one term, it is relatively simple to parse complex natural language noun phrases, as in the examples. Similarly, for generation of language from the representation, it is simple to generate complex natural language descriptions from structured variables. If constraints on variables corresponding to the complex noun phrases were represented using first-order predicate logic, then it would be difficult to generate natural language noun phrases corresponding to these variables. This is because the constraints on variables would, likely, be well-separated from the variables in the antecedents of rules involving these variables. This is not the case in a structured variable representation.

7.2 Description Subsumption and Question Answering

Because the structure of the representation of rules is "flat", that is, there is not the artificial antecedent-consequent structure associated with first-order logic-based representations, it is possible to frame questions whose answers are rules and not just ground formulas. Since the structure of the question will mirror the structure of the rule, any rule that is subsumed by a question is an answer to that question. Figure 7 gives a sample dialog involving questions whose answers are ground propositions (e.g., *Is John mortal*) as well as questions whose answers are rules (e.g., *Who is mortal*). This dialog also illustrates the uses of subsumption. Since we told the system *Every man is mortal*, it follows that any more specifically constrained man (e.g., *Every rich young man that owns some car*) must also be mortal. Note that this answer (a rule) follows directly by subsumption from a rule previously told to the system. This is another way in which rules may be answers to questions.

We examine the subsumption inference in Figure 7 in detail. All references to sentences will be to the numbered sentences in Figure 7. After sentence (1) is processed, sentence (3) then asks who is mortal. In a standard first-order-predicate-logic-based system, no answer could be given because there are, as yet, no instances of men in the knowledge base. This is contrary to the commonsense answer of sentence (4), which reiterates the rule of sentence (1). This is possible in ANALOG because the structure of the representation of the question (*Who is mortal*) is similar to that of any of its answers. Thus, any asserted proposition that is subsumed by the question is a valid answer (including rules).

Sentence (5) is an example of a question about a rule. Since *every man is mortal* is believed (the system was told this in sentence (1)) it follows that any more restricted sort of man is also mortal. The subsumption procedure specifies this explicitly. The representation of sentence (5) in is a less general form of sentence (1), since *any man* subsumes *any rich man*. Since the rule of sentence (5) is subsumed by a believed node (that of sentence (1)), it follows by Axiom 5 that sentence (5) is believed (thus, the representation of the question itself is a believed proposition) and the system answers yes to the question. Sentence (7) asserts that *John is a man*. At this point, the system knows *all men are mortal* and *John is a man*. When the question of sentence (9) is asked, the system finds the rule of sentence (1) and determines that it subsumes sentence (9) because John is a man and, again by axiom 5, the result follows. However, note that in this derivation the result is a ground formula rather than a rule. Sentence (11) illustrates the retrieval of the system's information about who is mortal; note the additional believed propositions. Sentence (13) is an example of a more complex noun phrase in a rule. The representation of (13) is subsumed by that of sentence (1) or (5) leading to the yes answer. In sentence (15), a new rule about rich young car-owning men being happy is introduced. The question of sentence (17) (*is John happy*) cannot be answered, because the system cannot determine that any rich young car-owning man subsumes John. This is because, while John is a man, he is not known to be young, rich, and owning a car (requirements for this subsumption). Sentence (19) informs the system of these requirements the systems understanding is verified by question (21). Note that the structure of the question involving two variables (*Who* and *a car*) is identical to that of the structure of its answer, which would not be the case if constraints were separated from variables in the antecedents of rules (as is done in typical logics). The question is asked again and, because the subsumption relationship can be determined, is answered in the affirmative.

8 Summary

We have described a logical language designed for natural language processing. We have shown how the primary means of automated deduction is a form of subsumption, that models reasoning methods used in natural language. We have illustrated the suitability of our work for natural language processing.

References

[Ali, 1994] Syed S. Ali. *A "Natural Logic" for Natural Language Processing and Knowledge Representation.* PhD thesis, State University of New York at Buffalo, Computer Science, January 1994.

[Beierle *et al.*, 1992] C. Beierle, U. Hedtstuck, U. Pletat, P. H. Schmitt, and J. Siekmann. An Order-sorted Logic for Knowledge Representation Systems. *Artificial Intelligence*, 55(2-3):149–191, June 1992.

[Brachman and Schmolze, 1985] Ronald J. Brachman and J. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[Brachman *et al.*, 1985] Ronald J. Brachman, Victoria Pigman Gilbert, and Hector J. Levesque. An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON. *Proceedings IJCAI-85*, 1:532–539, 1985.

[Fine, 1985] Kit Fine. *Reasoning with Arbitrary Objects.* Basil Blackwell, Oxford, 1985.

[Maida and Shapiro, 1982] A. S. Maida and S. C. Shapiro. Intensional Concepts in Propositional Semantic Networks. *Cognitive Science*, 6(4):291–330, 1982. Reprinted in *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque (eds.), Morgan Kaufmann, San Mateo, CA, 1985, 170–189.

[Morgado, 1986] E. J. M. Morgado. Semantic Networks as Abstract Data Types. Technical Report 86–19, Department of Computer Science, SUNY at Buffalo, 1986.

[Shapiro and Rapaport, 1987] S. C. Shapiro and W. J. Rapaport. SNePS Considered as a Fully Intensional Propositional Semantic Network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer–Verlag, New York, 1987.

[Shapiro *et al.*, 1993] S. C. Shapiro, W. J. Rapaport, and the SNePS Research Group. A Dictionary of Case Frames, 1993. In preparation.

[Shapiro, 1982] S. C. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *The American Journal of Computational Linguistics*, 8(1):12–25, 1982.

[Shapiro, 1986] S. C. Shapiro. Symmetric relations, intensional individuals, and variable binding. *Proceedings of the IEEE*, 74(10):1354–1363, 1986.

[Shapiro, 1991] S. C. Shapiro. Cables, Paths, and "Subconscious" Reasoning in Propositional Semantic Networks. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 137–156. Morgan Kaufmann, 1991.

[Woods, 1991] William A. Woods. Understanding Subsumption and Taxonomy: A Framework for Progress. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 45–94. Morgan Kaufmann, 1991.

Goo: A Database for Temporal Uncertainty Management

Keiji Kanazawa
Computer Science Division
University of California
Berkeley, CA 94720
kanazawa@cs.berkeley.edu

Abstract

This paper introduces Goo, a system for reasoning about temporal uncertainty. Goo is a system for maintaining a picture of the likelihood of facts and events over time. Goo features a flexible knowledge representation language based on a continuous-time probabilistic temporal logic. It constructs and maintains probabilistic networks models from logical knowledge in response to queries and assertions. Unlike most approaches, Goo is not confined to the representation of state transition models. It can model continuous change, and it can answer a rich class of relational queries. Goo offers a simple and expressive framework for reasoning about time and probability in a practical manner.

1 Introduction

We are interested in the design of robust inference systems for supporting activity in dynamic domains. In most domains, things cannot always be predicted accurately in advance. Thus, the capability to reason about change in uncertain environments is an important component of robust performance. Our goal is to develop theory and implementations of temporal reasoning under uncertainty that are well suited to a variety of domains. To this end, in past work, we presented an approach to representing knowledge about time and probability using temporal logic and Bayesian networks [Kanazawa, 1991]. In this paper, we present an implementation of these ideas in Goo, a *probabilistic temporal database* (PTDB).

Goo is a system for modeling the change in the likelihood of facts and events continuously over time. In contrast to state transition models such as Markov processes that focus on modeling what is true in successive states, Goo simplifies reasoning about the lifetimes of facts and the relationships between facts and events.

Goo was developed with an eye to applications in planning, scheduling, and medical decision making. An example domain is planning a trip: in order to form a good trip plan, it is useful to estimate the likely time of arrival at different locations. Suppose that we are trying to determine the best way to reach the Statue of Liberty from Lincoln Center in New York City. One way is to drive through Times Square. We know that there are often traffic jams in Times Square. To assess the impact of driving through Times Square, we would need to determine the likely time of arrival in Times Square, the likelihood of traffic jams at that time, and how they affect the overall plan. To take another example, in medicine, it is often critical to know the likely changes in a patient's condition over time in order to form a treatment plan.

In Goo, users enter facts and rules about the world, and Goo makes inferences drawing on that knowledge. It is based on the *logic of lifetimes*, a logic for representing knowledge about time and probability, and *time nets*, a type of Bayesian network for reasoning about time. Users can query the probability of facts and events, complex relational queries, and "what if?" queries based on hypothetical facts and events. Goo incrementally constructs and maintains time nets from logical knowledge in response to queries and assertions. It updates probability distributions as necessary in response to changes asserted by users. In its combination of a declarative knowledge representation with probabilistic networks, Goo is similar to work by Breese [Breese, 1987], Wellman [Wellman, 1990], and Goldman and Charniak [Goldman, 1990]. Goo is distinguished by its ability to represent and reason about time.

In the next section, we present the basic concepts underlying Goo. Then, we consider in more detail the operation of Goo, and the language used to program and interact with Goo. Finally, we present an example of its use.

2 Basic Concepts

In this section, we outline the basic concepts underlying a logic for declaratively expressing knowledge about time and uncertainty, and a graph representation that simplifies reasoning from such knowledge.

In our framework, we distinguish between *facts* and *events*. A fact is something that once it becomes true, stays true for some time.[1] By contrast, an event occurs

[1] A fact is a *fluent* [McCarthy and Hayes, 1969]; each fact

instantaneously; it is true over an infinitesimally small interval of time. Each fact has a *range*, the maximal interval $[u, v]$ over which the fact holds true uninterruptedly. The *lifetime* of a fact is $v - u$. Each event has a single *date* d at which it occurs.

For each fact φ, we automatically consider the events `beg`(φ) and `end`(φ), corresponding to the dates at which the fact becomes true and false. These events are known, respectively, as the *enabling* and *clipping* events of the fact. For the fact "Sally is here", its enabling event corresponds to the instant when Sally begins to be here, and its clipping event corresponds to the instant when she ceases to be here. There are also other types of events, called *point* events, such as "Sally arrives here", which are not themselves the enabling or clipping event of any fact, but which may trigger the enabling or clipping of a fact such as "Sally is here".

With each event, we may associate a probability density. For instance, if Sally is scheduled to arrive around 1 p.m., then we might model the likely time of Sally's arrival by a normal density with the mean at 1 p.m..

The *logic of lifetimes* (LL) [Kanazawa, 1992] is a continuous-time logic for representing knowledge about facts and events as outlined above. It is temporally-quantified and has real-number functions for representing probability distributions. LL is otherwise propositional as far as facts and events are concerned. The semantics of LL is based on a probability measure over possible worlds. For details on the syntax and semantics of LL, the reader is referred to [Kanazawa, 1991; Kanazawa, 1992] (see also [Haddawy, 1991] for a similar logic).

In LL, there is a set of propositional symbols Φ representing facts and events. From these propositions, it is possible to build up sentences associating the propositions with time points and probabilities. This is done with the `holds`, `occ`, and `P` sentence formers. `holds`(u, v, φ) means that fact φ is true throughout the time interval $[u, v]$[2], `occ`(u, φ) means that event φ occurs at time point u, and `P`(φ) is the probability of a wff φ. For instance, consider the fact `here(Sally)` representing "Sally is here". `holds(1pm, 2pm, here(Sally))` means that Sally is here between 1 and 2 p.m., `occ(1pm, beg(here(Sally)))` means that Sally begins to be here at 1 p.m., and `P(holds(2pm, ∞, here(Sally)))` is the probability that Sally never leaves here after 2 p.m.. Such sentences can be combined with logical connectives such as `and` and `not`. Conditional probability statements encode knowledge about how different facts and events affect one another.

As we see in ensuing sections, in `Goo`, we express knowledge about facts and events in a language essentially the same as LL called `bayesl`. Before we present `bayesl`, we introduce the graph representation used for reasoning about facts and events.

The *time net* belongs to the popular class of directed graph representation of knowledge about probability often known as the *Bayesian network* [Charniak, 1991; Pearl, 1988]. A Bayesian network is a directed acyclic graph $G = (N, A)$, consisting of a set of *nodes* N, and a set of *arcs* A. Each node represents a random variable of interest, which may be discrete or continuous-valued. Arcs are directed edges that represent dependencies between the random variables represented by nodes. *Parents* and *children* of nodes, on the basis of arc direction, are defined in the usual fashion, as are *root* and *leaf* nodes. Let Ω_n be the set of values (or continuous range) of a node $n \in N$. There is a probability distribution (density for continuous variables) $\Pr(n{=}\omega, \omega \in \Omega_n)$ for each node $n \in N$. If the node is a root node then this is its marginal probability distribution; otherwise, it is a conditional probability distribution dependent on the states of the parents of n in G. The text by Pearl [Pearl, 1988] provides a comprehensive treatment of Bayesian networks.

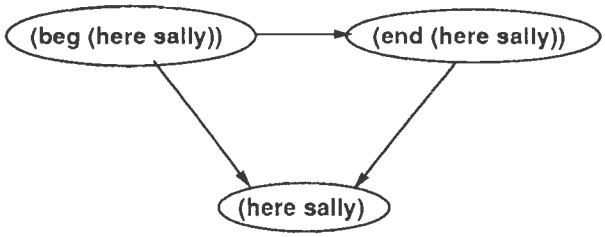


Figure 1: The time net for a fact.

In a time net, facts and events are represented as nodes, and conditional dependence between facts and events (on the basis of conditional distributions) are represented as arcs. For each point event of interest, a time net contains one node representing the date of the event. For each fact of interest, a time net contains three nodes, one node representing the range of the fact, and one node each representing the dates of the enabling and clipping events of the fact (see Figure 1). All fact and event nodes have continuous densities associated with them. For instance, a node representing the event "Sally arrives" might have associated with it a normal density with the mean at 1 p.m. as before.

A time net may also include nodes that correspond to compound sentences formed with logical connectives and temporal relations. For example, given two nodes representing the dates of two events a and b, we can add a node giving the probability that a precedes b, $P(a < b)$.

The time net is used for answering queries about the probabilities of facts and events, as well as compound and relational queries like the above. When a time net is created, only the marginal distributions of root nodes are known, perhaps along with some evidence about the value of some other random variables. Distributions for all other nodes must be computed on the basis of the known marginal distributions, evidence, and conditional distributions. Computing the distributions in a time net can be performed by a number of standard methods of *evaluating* Bayesian networks [Pearl, 1988]. `Goo` uses an approximation algorithms based on sampling [Shachter

must remain true throughout an interval of time, and it corresponds to the *liquid* proposition of [Shoham, 1988].

[2] Let $[b, e]$ be the range of φ. `holds`(u, v, φ) is true iff $[u, v]$ is a subset of the interval $[b, e]$. Note that this does not imply $b = u$ or $e = v$.

and Peot, 1989]. In such *simulation* algorithms, in each trial, the nodes in a network are visited in some order, and values for the random variables represented by the nodes are sampled from their probability distributions. Over repeated trials, the (weighted) averages of the samples converge approximately to the true distributions. Goo typically discretizes each density into a vector of "buckets" for recording the samples.

3 Goo

Goo is a system for combining declarative knowledge about time and probability with time nets for reasoning. In Goo, users assert knowledge in a language based on LL called bayesl. bayesl differs from LL chiefly by restricting the type of rules that can be asserted.

The ontology of bayesl is basically the same as that of LL. bayesl extends LL by the addition of *simple facts*. A simple fact is a variable that is either true or false. It is normally used to record background context. For example, a simple fact may assert that there is a plan to drive to the Statue of Liberty.

Goo combines a simple assertional database as in Prolog with time nets for reasoning involving probabilities. Simple facts, user assertions, and probability rules are all stored in the assertional database. In the current version of Goo, there is one time net that is being evaluated, updated, and modified at any given time. This time net represents the events and facts of interest. Goo incorporates an inference module that determines the time net that is applicable at any given time. Goo is *event driven* in the sense that it does nothing without a user command such as recording of information or queries (note that a "user" might be another program). Goo is implemented in CMU Common Lisp on Sun Sparcstations. The language bayesl is implemented directly in Lisp. Knowledge about time and probability is given to Goo as bayesl statements which are in turn just Lisp statements. In the following sections, we present assertions, queries, rules, and inference in Goo and bayesl.

3.1 Assertions in Goo

To assert facts and events in Goo, the record form is used. For example, to record that Sally left home at noon, we might use the form (record (occ #@"12:00" (leave Sally home))).[3] In general, a record expression takes the form (record <assertion>) where <assertion> is a ground sentence of the following type:

- (true <fact>): Simple <fact> is true.
- (false <fact>): Simple <fact> is false.
- (occ <t> <event>): <event> occurs at time <t>.
- (extent <t1> <t2> <fact>): <fact> becomes true at <t1> and becomes false at <t2>.

[3] In bayesl, a date constant is represented with a #@"date string" expression such as #@"noon" or #@"2:34pm". A time constant is represented with a #!"time string" expression such as #!"0:05" and #!"3:00", meaning, respectively, 5 minutes and 3 hours.

In Goo, knowledge about how events affect other events or facts is represented by conditional probability rules. record is also used to record such rules. (record (pdf <event> <antecs> <fun>)) asserts that the conditional density of <event> given <antecs> is given by the function <fun>. <antecs> is a list of facts and events that <event> is directly dependent on. If <antecs> is empty, then <fun> is the prior density of <event>. <antecs> are the *antecedents* of the rule, and <event> is the *consequent* of the rule. A Goo rule may contain variables, which start with the letter ?. Variables in rules are assumed to be universally quantified.

In Goo, each event probability density function is represented with a Lisp function. Goo uses sampling for assessment. Therefore, it is convenient to let each Lisp function sample from the density that it represents, based on the sample values of the random variables (typically other events) that the event depends on, if any. For example, to represent the fact that arriving in Times Square usually occurs about half an hour after leaving home, we might use the following rule (Rule (1) in the ensuing text):

```
(pdf (arrive TimesSquare)
     ((leave home) (drive home TimesSquare))
     (lambda ()
       (norm (+ (date (leave home)) #!"0:30")
                #!"0:15")))
```

where norm is assumed to be a function that samples from a normal density, in this case with mean at 30 minutes after the time of leaving home and a variance of 15 minutes. The bayesl (date <event>) construct is translated at run-time into an expression that returns the current sample or evidence date of the event named <event>.

Each time norm is called in the function above, it returns a sample value for the arrival time at Times Square. A sample might be "11:45 a.m.". Over repeated calls, norm returns different samples with different frequencies depending on the mean and the variance. In the above example, assuming that the time of leaving home is 11 a.m., the samples for the time of arrival at Times Square would be concentrated around 11:30 a.m. (the mean) and approximate a normal density.

The lambda body for a density can contain any Lisp form. Goo implements a family of common densities and other functions useful in such forms [Kanazawa, 1992]. More examples of pdf forms are given in a later section.

Goo does not currently allow the specification of the distribution of a fact. It assumes that the probability density function of a fact can be recovered from the probability density functions of its enabling and clipping events. This is not a severe restriction; given the distribution of a fact, it is relatively simple to recover the densities of its enabling and clipping events.

3.2 Queries in Goo

A query in Goo takes one of two forms. The first, used for simple facts, looks like (true? <fact>) and returns whether or not simple fact <fact> is true. The second, used for probability estimates, looks like (P

<wff>) and returns the probability of a ground sentence <wff>. For example, (P (occ #@"noon" #@"1pm" (beg traffic-jam)))) is a query for the probability that the traffic jam begins between noon and 1 o'clock.[4] Goo appears to be the first system for probabilistic temporal reasoning that features a rich class of queries including logical and temporal relations.

The basic wffs in probability queries are the following:

- (occ <t1> <t2> <event>): <event> occurs between <t1> and <t2>.
- (occ <t> <event>): <event> occurs at time <t>.[5]
- (holds <t1> <t2> <fact>): <fact> holds true throughout the interval from <t1> to <t2>.
- (holds <t> <fact>): <fact> holds true at time <t>.

In addition, we may recursively form compound queries involving the logical connectives **and**, **or**, and **not**, as well as temporal relations:

- (<point-rel> <date1> <date2>): <point-rel> is a point temporal relation in $\{<, <=, >, >=, =, /=\}$, the last of which is the inequality relation. <date1> and <date2> are either constants or expressions of form (date <event>). Thus we can compare two events with each other, or the time of one event in relation to a fixed point in time.

3.3 Rules in Goo

In Goo, knowledge about how facts and events affect other facts and events are represented by pdf rules (e.g., rule (1) on page 3.1). Goo applies rules to:

- Determine the probability function of a newly added node.
- Determine if any other nodes should be added when adding a node.
- Determine if the probability function of any existing nodes need to be updated to reflect the changes in the model.

The conditions under which the rule applies are given by its antecedents. The action that the rule specifies is either to add its consequent as a node (and arcs from the nodes for the antecedents to the consequent node), or if the node already exists, to assign the node its density function.

A rule is *potentially applicable* whenever all of the antecedents of the rule are present in the database. We only perform the action specified in a potentially applicable rule when the rule has not yet been applied, and there is no other rule that applies better in the situation. In a given situation, any number of rules may apply. Two potentially applicable rules whose consequents are the same and whose antecedents have a non-null intersection are said to *compete*.

As an example, suppose that A and B are in the database. The two rules (pdf C (A) <fn1>) and (pdf C (A B) <fn2>) compete. Both rules specify that a node named C is to be added, but which probability density function is to be selected?[6]

In general, the second rule corresponds to the existence of more conditioning information and is to be preferred. In Goo, the rule that applies in such situations is the rule with the maximal antecedent set with respect to set inclusion. Goo currently requires that there be a total order with respect to antecedent set inclusion over each set of competing rules. This is because there is no general way to "complete" an ambiguous set of rules.

In [Kanazawa, 1992], we present a polynomial-time algorithm for rule application in Goo. The algorithm is a forward chaining algorithm that is recursively invoked whenever a previously unrecorded fact or event is asserted. The algorithm provably terminates for rule sets which are ground and acyclical. Rule set cyclicity is defined by regarding each rule as a hyperedge from its antecedents to its consequent. If there is a cycle in the hypergraph consisting of all the rules in a set, then the rule set is cyclic. With a ground rule set, acyclicity is a sufficient condition to avoid creating cycles in time nets. It is not a necessary condition; one reason is that a rule set may contain redundant information. It appears to be non-trivial to unfold such spurious cycles from a rule set.

Goo allows cyclical rule sets if the rules are universally quantified (the rules have variables in them). Such rule sets are not necessarily problematic, especially when there are *asymmetric dependencies* in the rule set that effectively render the rule set acyclic [Geiger and Heckerman, 1991; Fung and Shachter, 1990]. Unfortunately, Goo's rule application algorithm is not guaranteed to terminate for such rule sets. Currently, the burden is on the bayesl programmer to handle potential circularities correctly. An important future research topic for Goo concerns such circularities in rule sets, for both ground and nonground cases.

3.4 Answering Queries in Goo

To answer queries of the form (true? <fact>), Goo simply looks through its assertion database. To answer a probability query, Goo may need to assess probabilities by evaluating its current time net, possibly after first augmenting it with additional nodes. If the model has not changed since it was last evaluated, then Goo will not perform inference to answer a query.

Once the current time net has been evaluated, it is trivial to answer basic queries about a fact or event relative to a time point or interval. For a time point, Goo simply looks up the appropriate estimate from the density for the node representing the fact or event. For a time interval, Goo integrates the same density over the interval. Because Goo discretizes densities, its answer for a time point is really the probability for the "bucket" containing the point. For the same reason, integrating

[4] It is also possible to directly read or plot the densities associated with time net nodes.

[5] The probability of an event at a point is always 0. Goo returns the event density over a small ϵ interval corresponding to the discretization granularity in sampling computations.

[6] Assuming that the probability density functions are different – if they are the same, then C *doesn't* depend on B.

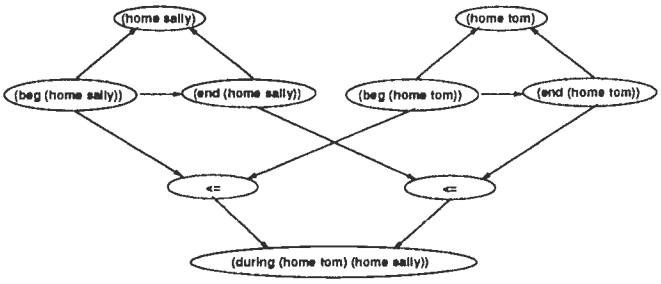


Figure 2: The portion of a time net for a compound query.

over an interval is a simple summation over the buckets contained in the interval.

A query may also involve the derivation of relations from known entities. For instance, a query may ask "What is the probability that `flight1437` will arrive before `flight989`?". We may know the densities for each separate flight, but not explicitly their relation. To compute it, Goo performs backward chaining to extend the model first.

The rules that Goo uses for this backward chaining are similar to `pdf` rules. They take the form `(answer <query> <antecs> <fun>)` where `<query>` corresponds to a relational query, `<antecs>` is the set of antecedents to the rule, and `<fun>` is the density of the consequent relation. For example, this rule applies in conjunctive queries:

```
(answer (and ?wff1 ?wff2)
        (?wff1 ?wff2)
        (lambda ()
          (indicator (and ?wff1 ?wff2))))
```

This states that to answer a (probability) query that is the conjunction of two wffs, the function given as a lambda expression can be used given the nodes for the wffs. `indicator` implements the indicator function I: it returns 1 if its argument is non-nil, and 0 otherwise.

If nodes corresponding to the antecedents do not exist, then Goo will recursively apply the backward chaining algorithm to try to add them to the current time net. For instance, consider the query "how likely is it that Tom will be home during the entire time that Sally is home?":

```
(P (and (<= (date (beg (home sally)))
            (date (beg (home tom))))
        (<= (date (end (home tom)))
            (date (end (home sally))))))
```

Goo will apply the rule for `and`, and then try to find nodes corresponding to each of the conjuncts. If it fails to find such nodes, then it will try to add them recursively by backward chaining. Goo has backward chaining rules for all of the basic logical and relational constructs allowed in queries such as `<=`. So provided that the facts `(home tom)` and `(home sally)` have been asserted in the database, Goo will eventually construct a node to answer the query (Figure 2).

```
;;;  Arrival at Times Square

(pdf (arrive times-square)
  ((leave home) (drive home times-square))
  (lambda ()
    (norm (+ (date (leave home)) #!"1:00")
             #!"0:30")))

;;;  Leaving depends on the traffic jam

(pdf (leave times-sq)
  ((arrive times-sq)
   (beg traffic-jam)
   (end traffic-jam))
  (lambda ()
    (if (or (< (date (arrive times-square))
               (date (beg traffic-jam)))
            (> (date (arrive times-square))
               (date (end traffic-jam))))
        (norm (+ (date (arrive times-square))
                 #!"0:15")
              #!"0:05")
        (norm (+ (date (arrive times-square))
                 #!"0:30")
              #!"0:15"))))

;;;  Being in Times Square

(pdf (beg (loc times-square))
  ((arrive times-square))
  (lambda ()
    (date (arrive times-square))))

(pdf (end (loc times-square))
  ((beg (loc times-square)) (leave times-square))
  (lambda ()
    (if (< (date (leave times-square))
           (date (beg (loc times-square))))
        (error "Left Times Square before arrival!")
        (date (leave times-square)))))
```

Figure 3: Goo code for the Times Square example.

4 An Example

Let us now examine an example of the use of Goo for dynamic reasoning about time and probability. This is the example of driving through Times Square on our way to the Statue of Liberty. Again, our interest is in predicting what time we are likely to arrive in Times Square, whether or not the traffic jam has already begun, and in predicting our eventual time of arrival at the Statue of Liberty.

Figure 3 shows the part of the `bayesl` causal theory used for reasoning about when we are likely to arrive in Times Square and how long we are likely to be there once we arrive. The first rule gives time of arrival at Times Square as a normal density with mean at an hour after time of departure from home and a variance of 30 minutes. The time of leaving Times Square depends on

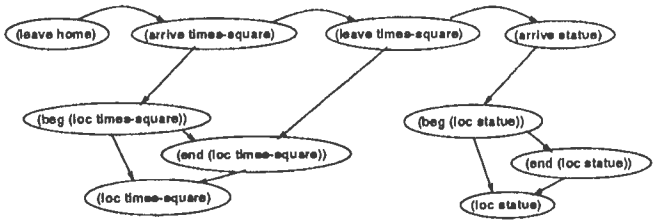


Figure 4: The time net after recording departure time.

whether or not the traffic jam is going on at the time of arrival. Two normal densities with different means and variances are given for the two cases. Note that this is an approximation, in that the rule only depends on whether or not the traffic jam was going on time of arrival at Times Square: even if the Traffic Jam begins one second after arrival, this rule gives time of departure from Times Square as if there is no traffic jam. A different model might give it in terms of a proportional hazard model [Cox, 1972]. The last two rules relate the dates of the enabling and clipping events of being in Times Square, with the arrival event for the former, and the leaving event for the latter. These are given deterministically with no uncertainty (e.g., the date of beginning to be in Times Square is exactly the same as the date of arriving in Times Square).

For the entire planning problem, there would also be similar rules for reasoning about arrival at the Statue of Liberty, and various other rules and assertions. Given such a `bayesl` causal theory, `Goo` constructs the appropriate model automatically on the basis of asserted facts and events and queries.

To initiate inference in `Goo`, a user makes assertions.[7] For example, we inform `Goo` of our plan to drive to Times Square by asserting a simple fact: `(record (true (drive home times-square)))`. Next, we record the time at which we are planning to leave home: `(record (occ #@"11am" (leave home)))`. `Goo` first checks to see if an assertion concerning the `(leave home)` event is already in the database. If no such assertion exists (as in this case), then `Goo` adds a node to the current time net representing the event. In addition, it begins forward chaining to determine what other facts or events need to be added as a consequence. As it happens, `Goo` adds quite a few events and facts (Figure 4). Note that at this point, we have not asserted any information about the traffic jam.

`Goo` never evaluates its time net unless requested by a user, either explicitly with the `(compute)` form, or implicitly by a probability query. In addition to answering queries, `Goo` can display a density or distribution by window graphics, or by dumping a plot files in popular formats. Here we request `Goo` to plot the likely time of arrival at Times Square to a file:[8] `(display #{(arrive times-square)} :output :gnuplot)`. This is an implicit query for a probability value, and therefore, `Goo` first evaluates the time net, and then dumps a plot file (Figure 5).

[7] This is an excerpt from an actual `Goo` session that is recorded in fuller detail in [Kanazawa, 1992].

[8] The `bayesl` `#{<wff>}` construct is shorthand for the node representing `<wff>`.

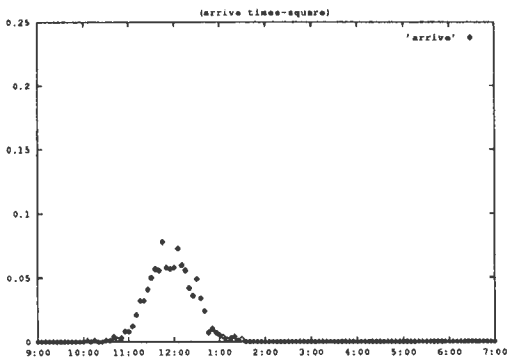


Figure 5: The plot for time of arrival.

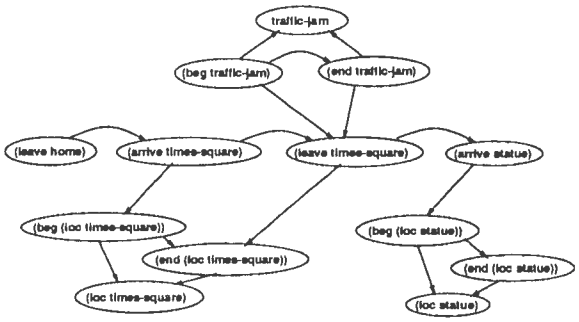


Figure 6: The time net after recording traffic jam.

Let us now assert information about the traffic jam: `(record (pdf (beg traffic-jam) () (lambda (norm #@"12:30" #!"0:45"))))`. This results in nodes being added about the traffic jam (Figure 6). In addition, the conditional density for the time of leaving Times Square is replaced by a new function that depends on the traffic jam. To estimate the start time of the traffic jam, `Goo` knows that it needs to recompute the probabilities in the time net. If we request its density, `Goo` goes ahead and makes the inferences. Figure 7 shows the plot of this density.

One of our main interests is the probability of ending up in a traffic jam if we drive through

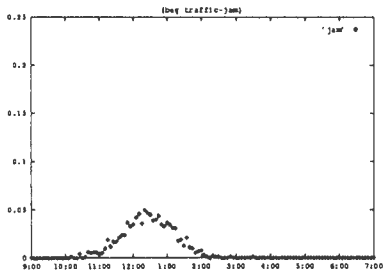


Figure 7: The density of the beginning of traffic jam.

Times Square: `(P (< (date (arrive times-square)) (date (beg traffic-jam))))`. Because this is an event relational query, and it has not been asked before, Goo must create a node representing the relational query. After creating the node, Goo would evaluate the time net again to estimate the probability of the query (which it estimates as 0.69).[9]

What if we now ask for the probability that we arrive in Times Square before noon? This time, we can estimate the probability directly from the density for the arrival event as outlined in Section 3.4. All we need is to integrate the density through noon. Goo will not create a node for this query, and will return an estimate right away (in this case, 0.523).

This concludes a brief example of how to program Goo, and how it operates in response to user assertions and queries.

5 Related Work

McDermott considered the issues behind a system for managing knowledge about time in [McDermott, 1982]. Dean [Dean, 1985] developed those ideas in his TMM (Time Map Manager). The TMM was a database for knowledge about time, facts, and events. Its representation of time was discrete, and it did not address issues of uncertainty directly.

The author earlier implemented a PTDB called ODDS [Dean and Kanazawa, 1989]. ODDS was one of the first AI systems for probabilistic temporal reasoning, along with efforts by Cooper and colleagues [Cooper *et al.*, 1988] and Hanks [Hanks, 1990]. ODDS was similar to Goo but considerably less sophisticated in its language and model of uncertainty.

LL combines elements of the logics of time by Shoham [Shoham, 1988] with elements of the logics of probability by Bacchus [Bacchus, 1988] and Halpern [Halpern, 1989]. As noted, the resulting logic is similar to work by Haddawy [Haddawy, 1991]. Martin and Allen have proposed a framework for reasoning about time and statistics [Martin and Allen, 1991]. The time net is an extension of the *network of dates* of Spiegelhalter and Berzuini [Berzuini, to appear].

6 Summary

We have presented Goo, a system for reasoning about time and probability in support of dynamic activity. Goo features a flexible and semantically well-founded language for expressing knowledge about facts, events, and their probability. The same language is also used for expressing a rich class of queries. Goo offers a simple and expressive framework for supporting reasoning about time and probability in a practical manner.

[9] Goo has an option whereby it can estimate this probability without re-evaluating the time net. This option can be expensive because it involves keeping around samples generated during evaluation. Therefore, it is usually not enabled.

Goo has other capabilities omitted from this paper. First of all, Goo implements a simple capability for performing hypothetical reasoning. Users are able to postulate facts or events and their times in order to determine their effects. Goo implements this by allowing creation of copies of the database. In addition to the point temporal relations mentioned in this paper, Goo implements all of the interval temporal relations defined by Allen [Allen, 1983]. Finally, Goo can be used to generate *discrete time nets*, Bayesian networks that model discrete Markov processes and semi-Markov processes [Howard, 1969; Kanazawa, 1992; Dean *et al.*, 1992; Provan and Clarke, 1993; Nicholson, 1992].

Goo has two major restrictions. First of all, the only type of update allowed in Goo is to assert new facts and events, and to enter a known value for a previously uncertain random variable. Goo currently does not allow retraction of previously asserted facts and events.[10] This hampers its ability to support a more sophisticated scheme for hypothetical inference.

A more fundamental limitation of Goo is the inability to translate arbitrary LL theories into time nets. It was necessary to ensure that propositional theories are acyclic to ensure termination of the model construction algorithm. Although Goo allows the specification of less restricted theories, it does not guarantee that models will be acyclic or that the model construction algorithm will terminate. Theories for Goo need to be built with a firm understanding of the implementation rather than being truly general purpose. The gap between the expressiveness of the logic and time nets is a ripe area for future research.

Of other restrictions, the single time net assumption is either very simple or very difficult to solve. It can be very difficult if different time nets can provide different estimates for the same quantity. Complete subsumption over competing rules is restrictive but well motivated. It can be relaxed by considering *prototypical interactions* between events such as a temporal analogue of the *noisy-or* [Pearl, 1988].

Acknowledgements

Most of this work was performed at Brown University and supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601, by the Air Force and the Advanced Research Projects Agency of the Department of Defense under Contract No. F30602-91-C-0041, and by the National Science Foundation in conjunction with the Advanced Research Projects Agency of the Department of Defense under Contract No. IRI-8905436. The author is currently supported by the State of California, PATH MOU-130.

References

[10] It is possible to hypothesize different values for existing facts and events.

[Allen, 1983] James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832 843, 1983.

[Bacchus, 1988] Fahiem Bacchus. *Representating and Reasoning with Probabilistic Knowledge.* PhD thesis, University of Alberta, 1988. Also issued as Waterloo University Technical Report CS-88-31.

[Berzuini, to appear] Carlo Berzuini. A probabilistic framework for temporal reasoning. *Artificial Intelligence*, to appear.

[Breese, 1987] John S. Breese. Knowledge representation and inference in intelligent decision systems. Technical Report 2, Rockwell International Science Center, 1987.

[Charniak, 1991] Eugene Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

[Cooper *et al.*, 1988] Gregory F. Cooper, Eric J. Horvitz, and David E. Heckerman. A method for temporal probabilistic reasoning. Memo KSL-88-30, Knowledge Systems Laboratory, Stanford University, 1988.

[Cox, 1972] D. R. Cox. Regression models and life tables (with discussion). *Journal of the Royal Statistical Society Series B*, 34:187–220, 1972.

[Dean and Kanazawa, 1989] Thomas Dean and Keiji Kanazawa. Persistence and probabilistic projection. *IEEE Transactions on Systems, Man and Cybernetics*, 19(3):574–585, May/June 1989.

[Dean *et al.*, 1992] Thomas Dean, Jak Kirman, and Keiji Kanazawa. Continuous-time stochastic processes for applications in planning and control. In *Proceedings of the First International Conference on AI Planning Systems*, 1992.

[Dean, 1985] Thomas Dean. Temporal imagery: An approach to reasoning about time for planning and problem solving. Technical Report 433, Yale University Department of Computer Science, 1985.

[Fung and Shachter, 1990] Robert M. Fung and Ross D. Shachter. Contingent influence diagrams. Submitted for publication, 1990.

[Geiger and Heckerman, 1991] Dan Geiger and David Heckerman. Advances in probabilistic reasoning. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 118–126, Anaheim, California, 1991.

[Goldman, 1990] Robert Goldman. *A Probabilistic Approach to Language Understanding.* PhD thesis, Department of Computer Science, Brown University, 1990. Available as Technical Report CS-TR-90-34.

[Haddawy, 1991] Peter Haddawy. *Representing Plans Under Uncertainty: A Logic of Time, Chance, and Action.* PhD thesis, Department of Computer Science, University of Illinois Urbana-Champaign, 1991.

[Halpern, 1989] Joseph Y. Halpern. An analysis of first-order logics of probability. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1375–1381, Detroit, Michigan, 1989. IJCAI.

[Hanks, 1990] Steven John Hanks. *Projecting Plans for Uncertain Worlds.* PhD thesis, Yale University Department of Computer Science, January 1990.

[Howard, 1969] Ron A. Howard. *Dynamic Probabilistic Systems*, volume I: Markov Models. Wiley, New York, 1969.

[Kanazawa, 1991] Keiji Kanazawa. A logic and time nets for probabilistic inference. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, California, 1991. AAAI.

[Kanazawa, 1992] Keiji Kanazawa. *Reasoning about Time and Probability.* PhD thesis, Department of Computer Science, Brown University, Providence, Rhode Island, 1992.

[Martin and Allen, 1991] Nathaniel Martin and James Allen. A language for planning with statistics. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 220–227, Anaheim, California, 1991.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4, 1969.

[McDermott, 1982] Drew V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Nicholson, 1992] Ann E. Nicholson. *Monitoring Discrete Environments using Dynamic Belief Networks.* PhD thesis, Department of Engineering Sciences, Oxford University, 1992.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, Los Altos, 1988.

[Provan and Clarke, 1993] Greogory M.A. Provan and John R. Clarke. Dynamic network construction and updating techniques for the diagnosis of acute abdominal pain. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15, 1993.

[Shachter and Peot, 1989] Ross D. Shachter and Mark A. Peot. Evidential reasoning using likelihood weighting. Technical report, Artificial Intelligence, Engineering-Economic Systems Department, Stanford University, 1989.

[Shoham, 1988] Yoav Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, 1988.

[Wellman, 1990] Michael P. Wellman. *Formulation of Tradeoffs in Planning Under Uncertainty.* Pitman, London, 1990.

The specification and implementation of a first order logic for uncertain temporal domains

Ehric Ho
Bell-Northern Research
P.O. Box 3511, Station C
Ottawa, Ontario
Canada, K1Y 4H7
ehric@bnr.ca

André Trudel
Jodrey School of Computer Science
Acadia University
Wolfville, Nova Scotia
Canada, B0P 1X0
trudel@AcadiaU.ca

Abstract

We formally define a first order logic that is suitable for representing and reasoning about uncertain temporal domains. The logic can represent both interval and point based qualitative and quantitative information. We provide a syntax, semantics, and axiomatization for the logic. We then describe the constraint logic programming implementation of the logic. The implementation, along with its formal specification, is suitable for tackling real world temporal problems.

1 Introduction

A popular approach in Artificial Intelligence for representing and reasoning about temporal domains is to use a first order (temporal) logic. Ideally, the first order logic should be formally defined and implemented. The formal definition, which consists of a syntax and semantics, removes all ambiguity from the structure and meaning of formulas. The implementation allows a user to solve real world temporal problems and evaluate the usefulness of the logic.

Very few temporal logics in AI are formally defined and implemented. Table 1 contains a list of some of the most influential and popular temporal logics. Note that out of the six logics, only the situation calculus is formally defined and implemented. This logic is primarily suited for simple single agent domains where actions occur in isolation. If the user has a complex domain which involves interval based information and/or simultaneous actions, one of the other logics must be chosen. When choosing another logic, the user must sacrifice either the formal definition or the implementation. We are not aware of a popular logic in AI suitable for complex real world domains that is both implemented and formally defined.

In the next section, we formally define a first order temporal logic by giving its syntax, semantics, and axioms. The proposed logic is based on the technique used in RGCH [Goodwin *et al.*, 1992] of defining interval based information in terms of what is true at the point level, and using the Riemann integral. We then describe the logic's implementation in the constraint logic programming language CLP($\mathbb{R}$) [Heintze *et al.*, 1992]. One feature of the implementation is the ability to specify temporal uncertainty with probabilities. The implementation, which is available from the authors, can be thought of as a temporal expert system shell that is ready to be used with real world problems.

2 Proposed logic

There are four types of temporal information that need to be represented: point and interval based qualitative and quantitative information. An example of point based qualitative information is:

John is working at time t_1. (1)

An example of point based quantitative information is:

John is walking at a speed of 5 km/hr at time t_1. (2)

An example of interval based qualitative information is:

John is not working from time t_1 to t_2. (3)

An example of interval based quantitative information is:

John is walking at a speed of 3 km/hr from time t_1 to t_2. (4)

Both qualitative and quantitative temporal information are represented by real valued functions of several variables. The quantitative examples (2) and (4) are written as:

$$velocity(t_1, john) = 5, \text{ and}$$
$$velocity(t_1, t_2, john) = 3. \quad (5)$$

A 0-1 valued function is used to represent qualitative information. Zero and one represent falsity and truth respectively. Examples (1) and (3) are written as:

$$work(t_1, john) = 1, \text{ and}$$
$$work(t_1, t_2, john) = 0. \quad (6)$$

Formula (6) is equivalent to:

$$\forall T.\ t_1 < T < t_2 \rightarrow work(T, john) = 0 \quad (7)$$

(i.e., John is not working at each point in the open interval). Similarly for formula (5).

The uniform representation of qualitative and quantitative information as real valued functions simplifies the

Logic	Implementation	Syntax and semantics
Situation calculus [McCarthy, 1985, Kowalski, 1979]	Yes	Yes
Allen [Allen, 1984]	Yes	No
McDermott [McDermott, 1982]	?	No
Shoham [Shoham, 1987]	No	Yes
Kowalski and Sergot [Kowalski and Sergot, 1986]	Yes	No
BTK [Bacchus *et al.*, 1989]	No	Yes

Table 1: Comparison of popular logics in AI

implementation. Externally, the user makes a clear distinction between the two types of information. However, both types of information are represented with identical functions internally.

We use the approach from [Goodwin *et al.*, 1992] where real valued functions can be integrated. When integrating a 0-1 valued qualitative function, the duration of truth is obtained. The result of integrating a real valued quantitative function depends on the function being integrated. For example, the integral of a velocity function is the total displacement:

$$displacement(t_1, t_2, john) = \int_{t_1}^{t_2} velocity(t, john)dt.$$

The syntax, semantics and axioms for the proposed logic are presented below.

2.1 Syntax

Given a set of non-temporal function symbols F_N, non-temporal constant symbols C_N, non-temporal variable symbols V_N, temporal function symbols F_T, temporal constant symbols C_T, and temporal variable symbols V_T (where $t \in V_T$), temporal terms (T_T) are defined as:

- $C_T, V_T \subseteq T_T$.
- If $g \in F_T$ is an n-ary ($n \geq 1$) temporal function symbol, $s_1, \ldots, s_n \in T_T$, then $g(s_1, \ldots, s_n) \in T_T$.

The temporal terms T_T give time a special status in the logic. Non-temporal terms (T_N) are defined as:

- $C_N, V_N \subseteq T_N$.
- If $f \in F_N$ is an n-ary ($n \geq 1$) non-temporal function symbol, $p_1, p_2 \in T_T$, $p_1 \leq p_2$, and $r_1, \ldots, r_{n-1} \in T_N$, then $f(p_1, p_2, r_1, \ldots, r_{n-2})$, $f(p_1, r_1, \ldots, r_{n-1})$, and $\int_{p_1}^{p_2} f(t, r_1, \ldots, r_{n-1})dt \in T_N$. The first argument to a non-temporal function is temporal, the second argument is temporal for interval based information and non-temporal for point based information, and the remaining arguments are non-temporal.
- If $r_1, r_2 \in T_T \cup T_N$, then $(r_1 + r_2), (r_1 - r_2), (r_1 \times r_2) \in T_N$.

Well-formed formulas (wffs) are defined as:

- If $\pi_1, \pi_2 \in T_N \cup T_T$, then $\pi_1 < \pi_2$, $\pi_1 \leq \pi_2$, $\pi_1 > \pi_2$, $\pi_1 \geq \pi_2$, and $\pi_1 = \pi_2$ are wffs.
- If ϕ_1, ϕ_2 are wffs, and $z \in V_T \cup V_N$ then $[\phi_1 \wedge \phi_2]$, $[\phi_1 \vee \phi_2]$, $[\phi_1 \rightarrow \phi_2]$, $[\phi_1 \leftrightarrow \phi_2]$, $[\neg\phi_1]$, $[\forall z.\ \phi_1]$, and $[\exists z.\ \phi_1]$ are wffs.

When there is no ambiguity, parentheses and square brackets are sometimes omitted.

Our proposed logic is based on another first order temporal logic called RGCH [Goodwin *et al.*, 1992]. As in RGCH, we have real valued functions that are integrated to produce interval based information. One major difference with RGCH, is our distinction between temporal and non-temporal terms. This distinction, which does not exist in RGCH, allows the user to customize the temporal terms which are used for representing time. In RGCH, the user is forced to use the real numbers. Another difference with RGCH, are interval based real valued functions (e.g., formulas (5) and (6)). Only point based functions are allowed in RGCH.

2.2 Semantics

The semantic domain or ontology is $\mathbb{R}$. An interpretation for the proposed logic is a tuple $I = \langle MC, SF, MF \rangle$ where:

- $MC : C_T \cup C_N \mapsto \mathbb{R}$.
- SF is a set of piece wise continuous functions. Each element of SF is a function from $\mathbb{R}^n$ to $\mathbb{R}$ for some n.
- $MF : F_T \cup F_N \mapsto SF$.

A variable assignment is a function $VA : V_T \cup V_N \mapsto \mathbb{R}$. The function TA assigns an element of $\mathbb{R}$ to each temporal or non-temporal term as follows:

- If $x \in C_T \cup C_N$ then $TA(x) = MC(x)$.
- If $x \in V_T \cup V_N$ then $TA(x) = VA(x)$.
- If $g \in F_T$ is an n-ary ($n \geq 1$) temporal function symbol, and $s_1, \ldots, s_n \in T_T$, then $TA(g(s_1, \ldots, s_n)) = MF(g)(TA(s_1), \ldots, TA(s_n))$.
- If $f \in F_N$ is an n-ary ($n \geq 1$) non-temporal function symbol, $p_1, p_2 \in T_T$, $p_1 \leq p_2$, and $r_1, \ldots, r_{n-1} \in T_N$, then:

$$TA(f(p_1, p_2, r_1, \ldots, r_{n-2})) = MF(f)(TA(p_1), TA(p_2), TA(r_1), \ldots, TA(r_{n-2})),$$
$$TA(f(p_1, r_1, \ldots, r_{n-1})) = MF(f)(TA(p_1), TA(r_1), \ldots, TA(r_{n-1})),$$
$$TA(\int_{p_1}^{p_2} f(t, r_1, \ldots, r_{n-1})dt) = \int_{TA(p_1)}^{TA(p_2)} MF(f)(t, TA(r_1), \ldots, TA(r_{n-1}))dt.$$

  Note that the above definite integral is always defined because the integrand is a piece wise continuous function.

- If $r_1, r_2 \in T_T \cup T_N$, then:

$$TA((r_1 + r_2)) = TA(r_1) + TA(r_2),$$
$$TA((r_1 - r_2)) = TA(r_1) - TA(r_2),$$
$$TA((r_1 \times r_2)) = TA(r_1) \times TA(r_2)$$

The interpretation $I = \langle\ MC, SF, MF\ \rangle$ and variable assignment VA satisfy a formula φ (written $\models_I \varphi$ [VA]) under the following conditions:

- $\models_I \pi_1 < \pi_2$[VA] iff TA(π_1) < TA(π_2).
- $\models_I \pi_1 \leq \pi_2$[VA] iff TA($\pi_1$) $\leq$ TA(π_2).
- $\models_I \pi_1 > \pi_2$[VA] iff TA(π_1) > TA(π_2).
- $\models_I \pi_1 \geq \pi_2$[VA] iff TA($\pi_1$) $\geq$ TA(π_2).
- $\models_I \pi_1 = \pi_2$[VA] iff TA(π_1) = TA(π_2).
- $\models_I [\varphi_1 \wedge \varphi_2]$[VA] iff $\models_I \varphi_1$[VA] and $\models_I \varphi_2$[VA].
- $\models_I [\varphi_1 \vee \varphi_2]$[VA] iff $\models_I \varphi_1$[VA] or $\models_I \varphi_2$[VA].
- $\models_I [\varphi_1 \rightarrow \varphi_2]$[VA] iff $\models_I [\neg\varphi_1 \vee \varphi_2]$[VA].
- $\models_I [\varphi_1 \leftrightarrow \varphi_2]$[VA] iff $\models_I [\varphi_1 \rightarrow \varphi_2]$[VA] and $\models_I [\varphi_2 \rightarrow \varphi_1]$[VA].
- $\models_I [\neg\varphi]$[VA] iff $\not\models_I \varphi$[VA].
- $\models_I [\forall z.\ \varphi]$[VA] iff $\models_I \varphi$[VA'] for all VA' that agree with VA everywhere except possibly on z.
- $\models_I [\exists z.\ \varphi]$[VA] iff $\models_I \varphi$[VA'] for some VA' that agrees with VA everywhere except possibly on z.

2.3 Axioms

Besides the standard integral axioms for polynomials, we have an axiom for converting between interval and point based information:

$$\begin{aligned}&[\ Predicate(Time_{begin},\ Time_{end},\ Arguments)\\ &\quad = RealValuedFunction\ ] \Leftrightarrow\\ &\quad [\ \forall T.\ Time_{begin} < T < Time_{end} \rightarrow\\ &\quad Predicate(T,\ Arguments) =\\ &\quad RealValuedFunction\ ].\end{aligned} \tag{8}$$

This axiom is used to convert formula (6) into formula (7).

Note that in axiom (8), $RealValuedFunction$ cannot contain any temporal terms. For example, the axiom is not applicable to $displacement(T_1, T_2, john) = T_2 - T_1$. This formula has no point based equivalent.

3 Implementation

Our proposed logic is implemented in the constraint logic programming language CLP($\mathbb{R}$) [Heintze *et al.*, 1992]. Figure 1 gives an overview of the implemented system which consists of a temporal format specification component (Time File) and three typical expert system shell components: a knowledge base, an inference engine, and a user interface.

Recall that in the syntax (section 2.1), temporal terms are not explicitly specified. The onus is on the user to supply temporal constants, variables, and functions that are appropriate for the particular problem domain. The code for the temporal representation and its operators is stored in the "Time File".

The knowledge base contains problem domain dependent facts and rules that are stored in files, which are loaded during user consultation.

The inference engine is the work horse of the system and is divided into three modules: the interpreter, the analyser, and the estimator. The interpreter retrieves pertinent point and interval information from the knowledge base. Using the retrieved information and axioms (e.g., axiom (8)), the analyser attempts to derive required conclusions. The estimator, an optional feature, tries to compute missing point information. The final result produced by the inference engine is passed to the user interface.

The user interface is responsible for formatting, validating and packaging the input and output.

The system also optionally supports uncertainty with probabilities.

An overview of the implemented system is given in the remainder of the paper. See [Ho, 1994] for a detailed description. A sample session with the implementation is given in the Appendix.

4 Temporal format specification

The user must specify a temporal representation in the time file. A sample time file, which is included with the implementation, contains a specification for a temporal representation called MDHM. MDHM uses a Month:Day@Hour:Minute format. For example, 3:00am on June 10 is written as 6:10@3:0, and 2:30pm on July 8 is written as 7:8@14:30. Intervals are written as pairs of points. All points are assumed to be in the same calendar year (not a leap year). In practice, the finite number of time points is not a limitation since there are over 500,000 point constants.

It is possible to represent repeated time points. For example, X:1@Y:2 means the second minute of every hour of the first day of every month and X@3:Y means every minute of the third hour of every day.

4.1 Syntax of MDHM

Given a set of temporal constants C_T where $C_T \in \{0, \ldots, 59\}$, a set of temporal functions $F_T = \{:, @\}$, disjoint sets of temporal variable symbol V_{Month}, V_{Day}, V_{Hour}, V_{Minute}, V_{LHS}, V_{RHS}, and V_W, temporal terms T_T are defined as follows:

- Month $\in V_{Month}$ or $\{1, \ldots, 12\}$,
- Day $\in V_{Day}$ or $\{1, \ldots, 31\}$,
- Hour $\in V_{Hour}$ or $\{0, \ldots, 23\}$,
- Minute $\in V_{Minute}$ or $\{0, \ldots, 59\}$,
- LHS $\in V_{LHS}$ or Month:Day,
- RHS $\in V_{RHS}$ or Hour:Minute,
- V_W, LHS@RHS $\subseteq T_T$.

The Month:Day pair must be a valid combination. For example, 2:30 is invalid because February 30 is not a valid date.

4.2 Operators for MDHM

System defined operators, such as < and $\leq$, can only be used to compare real numbers. Binary operators are introduced for comparing temporal terms: >@, >=@, <@, <=@ and =@. The comparison of temporal terms

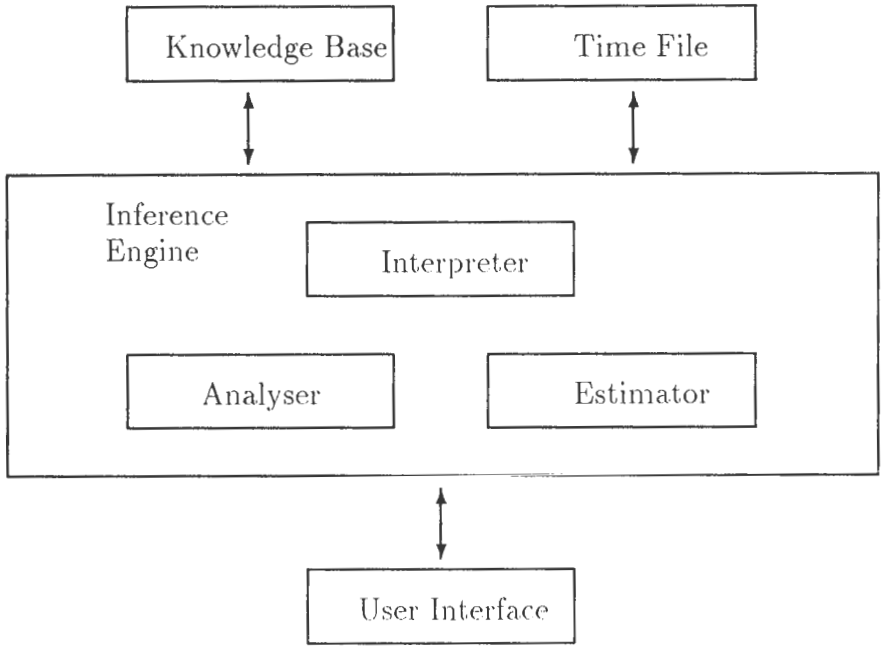


Figure 1: Implementation Overview

usually results in a system of constraints. For example, the solution to the inequality:

?- X@12:0 >@ Y@Z.

is:

X = Y, Z = HH:MM, HH < 12; or
X = M:D1, Y = M:D2, Z = HH:MM, D2 < D1; or
X = M1:D1, Y = M2:D2, Z = HH:MM, M2 < M1.

5 Knowledge base

The syntax used in the knowledge base differs from the syntax given in section 2.1 for the proposed logic. Information is stored in the knowledge base with the kb prefixed predicates: kb_point/4, kb_interval/5 and kb_integral/5. Since we make no distinction between qualitative and quantitative information, only two predicates, kb_point/4 and kb_interval/5, are needed to represent point and interval based information. A third predicate, kb_integral/5, is used for representing integrals.

To represent a wff from the proposed logic in the knowledge base, we extract the temporal arguments. For example, $velocity(t_1, john) = 5\ km/hr$ is represented by the following fact in the knowledge base:

$$kb\_point(t_1, velocity(john), 5).$$

Note that inequalities, such as $velocity(t_1, john) > 0$, cannot be stored in the knowledge base.

The reason for changing the syntax in the knowledge base is to simplify the implementation. The kb predicates allow us to write generic code for point, interval, and integral information. For example, the code for kb_point/4 works for any possible point based information that can appear as its second argument. In addition, the kb predicates allow the implementation to easily distinguish between point and interval information.

To deal with uncertainty, we use probabilities. The last argument of each relation in the knowledge base is optional and is used to store a probability value. If the argument is omitted, a probability of one is assumed.

In the remainder of this section, we discuss the relations used for representing point, interval, and integral information.

5.1 Point information

Qualitative and quantitative point based information is represented with the kb_point/4 predicate. To specify that *Fact* has a *Value* at *Time* with *Probability*, we write

$$kb\_point(Time, Fact, Value, Probability)$$

where *Time* is a time point, *Fact* is the symbolic representation of the information, *Value* is a real number or a polynomial, and *Probability* (optional argument) is a real number which reflects the probability of the fact. For example,

$$kb\_point(12{:}31@12{:}00, work(john), \#true, 0.7)$$

expresses the fact that there is a 70 percent chance of John working on New Year's Eve at noon. #true and #false are constants which are defined as one and zero respectively.

The probability (the 4th argument) in the kb_point/4 predicate may be omitted. For example, the following represents the formula p ∧ q — r without relying on probabilities:

$kb\_point(T, r, Z)$:- $kb\_point(T, p, Z)$,
$kb\_point(T, q, Z)$.

Probabilities at a point are related by the following axiom:

$$kb\_point(T, F, \#true, P) \Leftrightarrow kb\_point(T, F, \#false, 1 - P). \quad (9)$$

The implementation makes a distinction between explicit and derived information in the knowledge base. Derived point based information is specified with the point_value/4 predicate. For example, consider the case where $q \rightarrow p$ at time 1:1@5:0 and q is true over the interval (1:1@1:0, 1:1@10:0):

$kb\_point(1{:}1@5{:}0, p, Z)$:- $kb\_point(1{:}1@5{:}0, q, Z)$,
$kb\_interval(1{:}1@1{:}0, 1{:}1@10{:}0, q, \#true)$.

The query $kb\_point(1{:}1@5{:}0, p, \#true)$ fails because the knowledge base does not contain a fact that explicitly specifies that q is true at time 1:1@5:0. If instead we write:

$kb\_point(1{:}1@5{:}0, p, Z)$:-
$point\_value(1{:}1@5{:}0, q, Z)$,
$kb\_interval(1{:}1@1{:}0, 1{:}1@10{:}0, q, \#true)$,

we can prove $kb\_point(1{:}1@5{:}0, p, \#true)$. The subgoal $point\_value(1{:}1@5{:}0, q, \#true)$ succeeds because we derive it from the interval information stored in the kb_interval relation. Similarly, there are interval_value/5 and integral_value/5 predicates for derived interval and integral information respectively.

5.2 Interval information

Qualitative and quantitative interval based information is captured with the kb_interval/5 predicate. The general format of kb_interval/5 is:

$kb\_interval(Time_{begin}, Time_{end}, Fact, Value, Probability)$,

where $Time_{begin}$ is the beginning time of the interval, $Time_{end}$ is the ending time of the interval, $Fact$ is the symbolic representation of the information, $Value$ is a real number or a polynomial function, and $Probability$ (optional argument) is a real number which reflects the probability of the fact. Over the open interval from $Time_{begin}$ to $Time_{end}$, the value of $Fact$ is $Value$ with $Probability$ at each point. For example, assume John works from nine to five everyday and at each point in time between nine and five there is a 75% probability that he is actually working:

$kb\_interval(X@9{:}0, X@17{:}0, work(john), \#true, 0.75)$.

Intervals are open at both ends. Values at end points are specified with kb_point/4. This allows us to specify facts that are true only at one of the end points. Consider the above example, John starts work at nine in the morning and leaves at 5:00pm (he does not work at 5:00pm):

$kb\_point(X@9{:}0, work(john), \#true)$,
$kb\_interval(X@9{:}0, X@17{:}0, work(john), \#true)$.
$kb\_point(X@17{:}0, work(john), \#false)$.

The following example states that r is true when p is followed by q and they overlap:

$p(A, B)$ & $q(C, D)$ — $r(A, D)$
when $A < C < B < D$.

Assume the probability of r equals the probability of q. In our implemented system, we have:

$kb\_interval(A, D, r, \#true, Pq)$:-
$interval\_value(A, B, p, \#true, Pp)$,
$interval\_value(C, D, q, \#true, Pq)$,
$A <@ C$, $C <@ B$, $B <@ D$.

Note that interval_value/5 is used instead of kb_interval/5 since p or q may be computed from other information.

From axioms (8) and (9), the following axiom is derived:

$$kb\_interval(T_1, T_2, F, \#true, P) \Leftrightarrow kb\_interval(T_1, T_2, F, \#false, 1 - P). \quad (10)$$

5.3 Integral information

It is possible to integrate qualitative and quantitative point based information. For example, John walks 5 km to work each morning between 8:00am and 9:00am is expressed as:

$kb\_integral(X@8{:}0, X@9{:}0, velocity(john), 5)$.

The general format for the predicate is:

$kb\_integral(T_1, T_2, F, Z, P)$

which holds if $\int_{T_1}^{T_2} F(t)dt = Z$, with probability P (optional).

Subintervals must sometimes be considered when computing an integral. For example, assume John walks at a speed of five and three km/hr from 8:00 to 8:30 and 8:30 to 9:00 respectively. In order to compute his total displacement, two kb_interval/5 facts are used. The probability for the integral is defined as the weighted average of the P's of all subinterval values used in computing the integral.

6 Inference engine

The inference engine is made up of three components: an interpreter, an analyser and an estimator. The interpreter interacts with the knowledge base. The analyser deals with the conversion between point, interval, and integral information. Incomplete knowledge is handled by the estimator.

When a query is received, the interpreter accesses the knowledge base and returns pertinent information to the analyser and estimator. The interpreter ensures that returned information is within the time interval of interest which is set by the user. The interpreter also ensures that all temporal terms passed to the estimator are bound. The estimator only deals with completely specified time points. Another task of the interpreter is to compute the probability for rules when required.

The analyser deals with the transformation between point, interval and integral information. Point information may be computed from interval information. Integral information may be computed from one or more

pieces of subinterval information. The analyser also computes probabilities for the transformed information. When point information is derived from interval information, it takes the probability of the corresponding interval information. When integrating an interval, the probability is defined as the weighted average of the probabilities of all the subintervals.

As its name implies, the estimator estimates unknown point information. Exponential decay functions are used to approximate probabilities (a similar approach is used in [Eberbach and Trudel, 1992]). With quantitative information, linear interpolation is used to estimate a point value. For qualitative information, the estimated point value is based on the computed probabilities. User defined approximation functions can be associated with temporal information. See [Ho, 1994] for a detailed description of the estimator. We conclude with a qualitative example. John works between 9:00am and noon, and has lunch between 12:30 and 1:30pm:

$kb\_interval(X@9{:}0, X@12{:}0, work(john),$
$\#true, 1),$
$kb\_interval(X@12{:}30, X@13{:}30, work(john),$
$\#false, 1).$

The truth value of $work(john)$ is unknown between noon and 12:30. Figure 2 shows the estimated values of $work(john)$ over this interval. The probabilities are computed to be near 1 over the interval (12:00, 12:09), and near 0 over the interval (12:29, 12:30). We estimate $work(john)$ to be true and false over these intervals respectively. Between 12:09 and 12:29, the probabilities are estimated to be near 0.5 and we make no prediction about the truth value of $work(john)$. The coefficient of the decay function is determined by the length of the interval used in computing the estimated value (there is a three hour and one hour duration between 9:00-12:00 and 12:30-13:00 respectively). Hence, the duration of the two intervals estimated above are different.

7 User interface

The command line user interface supports a predicate, called ui, which takes a variable number of arguments. It validates input arguments and formats output data. All user interface routines are invoked from this ui predicate. The first argument of the predicate is a selector which describes the routine to be called. The following example specifies the time range that is of interest:

?- $ui(change\_time\_range, 6{:}6@0{:}0, 6{:}7@23{:}59).$

Any information that lies outside of the above time range will not be reported to the user. The following example queries the point information of any fact in the knowledge base within the time boundary:

?-ui(point_value, Time, Fact, Value, Probability).

When specifying temporal information, the first argument to *ui* makes the qualitative or quantitative distinction. For example, qualitative information is entered as:

?-ui(point_truth, X@9:15, work(john), #true).

Quantitative information is entered as:

?- $ui(point\_value, X@8{:}45, velocity(john), 3).$

8 Directions for future work

The following is a list of improvements to the implementation that we are contemplating:

- Re-implement the logic in a constraint logic programming language other than CLP($\mathbb{R}$) that supports graphical user interfaces. Instead of the current command line interface, all functions would be entered and displayed using graphs. Different colors could be used for point, interval, integral, and estimated values.
- As in a typical expert system, the implementation should prompt the user for missing information.
- Units of measurement should be added to the system (e.g., miles and kilometres).
- Using our implementation, construct a temporal expert system to solve a complex real world problem.
- Each consultation is independent. Routines should be provided to save the environment of a session, which includes the user's time of interest and the certainty threshold.
- Currently, the system supports integration over truth. It should be an option for a user to integrate over true or false.

9 Conclusion

We proposed a first order logic for temporal domains. The logic is formally specified via a syntax, semantics, and axiomatization. Point and interval based qualitative and quantitative information are uniformly dealt with in the logic. Also, uncertainty can be represented with probabilities. The logic is implemented using a constraint logic programming language. The implementation, along with its formal specification, is suitable for tackling real world temporal problems.

Acknowledgements

Research of the first author is supported by an Acadia University Graduate Fellowship. Research of the second author is supported by Natural Sciences and Engineering Research Council of Canada grant OGP0046773. We would like to thank Scott Goodwin and Eric Neufeld for helpful comments on the paper.

A Example

We present an example, its representation in the proposed logic, its knowledge base file, and a sample session with the implementation. The example deals with John who works in an office from nine to five everyday including Saturday and Sunday. He takes an hour lunch break from 12:30 to 13:30. It is unknown if John works between 12:00 and 12:30. Every morning, he walks to work at a velocity of five and three kilometres per hour from eight to eight thirty and eight thirty to nine respectively. It takes him an hour to walk to work. The example is represented in the logic as follows:

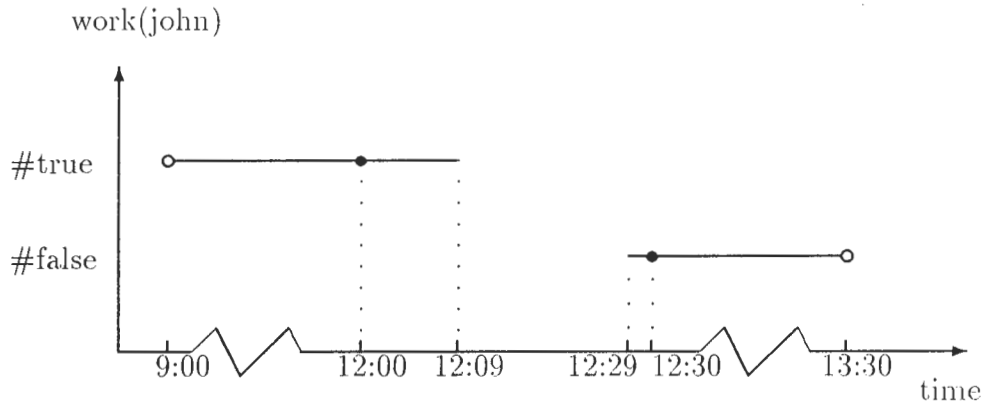


Figure 2: A plot of work(john) against time

work(X@0:00, john) = #false.
work(X@0:00, X@9:00, john) = #false.
work(X@9:00, john) = #true.
work(X@9:00, X@12:00, john) = #true.
work(X@12:00, john) = #true.
work(X@12:30, X@13:30, john) = #false.
work(X@13:30, john) = #true.
work(X@13:30, X@17:00, john) = #true.
work(X@17:00, john) = #false.
work(X@17:00, X@23:59, john) = #false.
work(X@23:59, john) = #false.
$\forall T$. work(T, john) $\rightarrow$ in_office(T, john).
velocity(8:00, 8:30, john) = 5.
velocity(8:30, 9:00, john) = 3.
$\forall X, Y$. displacement(X, Y, john) =
$\int_X^Y$ velocity(t, john) dt.

Note that probabilities are omitted. Also note that it is unknown if John is working between 12:00 and 12:30. See figure 3 for a plot of the work function. The code in the knowledge base for the example is:

```
kb_point( X@9:0, work(john), #true ).
kb_interval( X@9:0, X@12:0, work(john), #true ).
kb_point( X@12:0, work(john), #true ).
kb_interval(X@12:30,X@13:30,work(john),#false).
kb_point( X@13:30, work(john), #true).
kb_interval(X@13:30,X@17:0,work(john),#true).
kb_point( X@17:0, work(john), #false ).
kb_interval(X@17:0,Y@9:0,work(john),#false):-
        time_next_day( X, Y ).
kb_point( X, in_office(john), A, P ) :-
        point_value(X,work(john),A,P).
kb_interval( X, Y, in_office(john), A, P ) :-
        interval_value(X,Y,work(john),A,P).
kb_interval( X@8:0, X@8:30, velocity(john), 5 ).
kb_interval( X@8:30, X@9:0, velocity(john), 3 ).
kb_interval( X, Y, displacement(john), D, P ) :-
        integral_value(X,Y,velocity(john),D,P).
```

The following is an annotated (lines beginning with a "%") session with the implementation:

```
2 ?- ui( load_new, john ).
% load a knowledge base file
Time Lower Bound     = 1 : 1 @ 0 : 0
Time Upper Bound     = 1 : 4 @ 23 : 59
Probability Threshold = 0.5
The Point Estimator is ON
The knowledge base has the following list of
Truth functions:-
                        work(john)
                        in(john, office)
and the following list of Real Valued functions:-
                        velocity(john)
                        displacement(john)
** Yes

3 ?- ui( interval_truth, 1:1@9:15, 1:1@11:0,
work(john), Z ).
% query if john is working between
% the time interval
Z = true

4 ?- ui( interval_value, 1:1@8:0, 1:1@9:0,
displacement(john), Z ).
% query the displacement of john between
% the time interval
% it is computed by integrating the velocity of john
Z = 240

5 ?- ui( turn_estimator, off ).
% disable estimator
** Yes

6 ?- ui( point_truth, 1:1@12:2, work(john), Z ).
% truth value is unknown at this point
** No

7 ?- ui(turn_estimator, on), ui(point_truth,
1:1@12:2, work(john), Z, P).
% enable estimator, and query point based
% qualitative information
Estimating point information...
Z = true
P = 0.795956
```

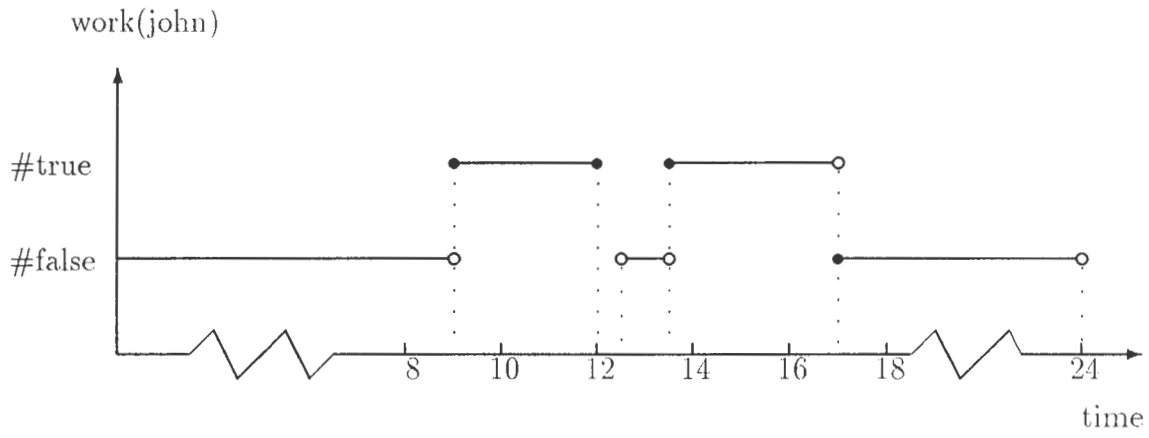


Figure 3: A plot of work(john) against time

```
8 ?- ui( change_prob_threshold, 0.9 ),
ui( point_truth, 1:1@12:2, work(john), Z, P ).
% change the probability threshold
% to a higher value
% and query the same point information again
Estimating point information...
** No

9 ?- ui( integral_truth, 1:2@8:0, 1:2@10:0,
work(john), Z, P ).
% how long does John work between the interval?
Z = 0 : 0 @ 1 : 0
P = 1
```

References

[Allen, 1984] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[Bacchus *et al.*, 1989] F. Bacchus, J. Tenenberg, and J. A. Koomen. A non-reified temporal logic. In *First International Conference on Principles of Knowledge Representation and Reasoning*, pages 2–10, Toronto, Canada, May 1989.

[Eberbach and Trudel, 1992] Eugene Eberbach and André Trudel. Representing spatial and temporal uncertainty. In *the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 529–532, Mallorca, Spain, July 1992.

[Goodwin *et al.*, 1992] Scott D. Goodwin, Eric Neufeld, and André Trudel. Temporal reasoning with real valued functions. In *Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence*, Seoul, Korea, September 1992.

[Heintze *et al.*, 1992] Nevin C. Heintze, Joxan Jaffar, Spiro Michaylov, Peter J. Stuckey, and Roland H. C. Yap. *The CLP(R) Programmer's Manual*, 1992.

[Ho, 1994] Ehric K. H. Ho. The specification and implementation of a first order logc for uncertain temporal domains. Master's thesis, Acadia University, 1994.

[Kowalski and Sergot, 1986] R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[Kowalski, 1979] R. A. Kowalski. *Logic for Problem Solving*. Elsevier North Holland, New York, 1979.

[McCarthy, 1985] J. McCarthy. Progams with common sense. In R.J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 299–307, Los Altos, USA, 1985. Morgan Kaufmann.

[McDermott, 1982] D. V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Shoham, 1987] Y. Shoham. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33:89–104, 1987.

Circumscription in a Paraconsistent Logic

Zuoquan Lin*
Computer Science Department
Shantou University
Shantou 515063, China
zqlin%stumis@hkucnt.hku.hk

Abstract

In this paper we describe paraconsistent circumscription by the application of circumscription in a paraconsistent logic. It turns out that paraconsistent circumscription can be well characterized by the minimal semantics which is nonmonotonic and paraconsistent. It brings us advantages in two respects: it makes that nonmonotonic logic would be nontrivial when there was a contradiction, and that paraconsistent logic would be equivalent to classical logic when there was not effect of a contradiction.

Keywords. paraconsistent logic, nonmonotonic logic, circumscription, minimal models

1 Introduction

A theory is *nonmonotonic* if it might not entail A from an enlarging set of premises S' while A was entailed by S which is contained in S'. *Nonmonotonic logics* are those theories which are nonmonotonic. Reasoning with incomplete knowledge is nonmonotonic as well be motivated in recent Artificial Intelligence (AI) literature [Ginsberg 1987]. A theory is *inconsistent* if it contains both A and $\neg A$ for some proposition A. A theory is *trivial* if it contains every proposition. *Paraconsistent logics* are those theories in which an inconsistent theory can be nontrivial. Reasoning with inconsistent knowledge is paraconsistent as early motivated in philosophy literature [Priest *et al.*, 1989].

In the presence of incompleteness, the reasoner usually makes some assumptions which may be restracted when facing with new knowledge. The key idea of nonmonotonic reasoning is that it transforms partial knowledge into more complete one. The state of relatively complete knowledge is arrived on by the mechanism of managing the consistency of knowledge by ruling out a contradiction. In the other words, nonmonotonic reasoning can be viewed as a kind of handling inconsistency. In the presence of inconsistency, the reasoner usually localizes the effect of a contradiction and destroies troublesome problem of triviality. The key idea of paraconsistent reasoning is that it takes some state of affairs of consistent knowledge under inconsistent one. The state of affairs of knowledge constrains the effect of no controlling contradictions in order to maintain relative consistency of knowledge. In the other words, paraconsistent reasoning can be viewed as a kind of handling incompleteness.

Although they share some common feature of dealing with contradictions, both logics differ in a significant sense. From the viewpoint of contradiction, the former dynamically maintains the consistency of knowledge by ruling out a contradiction when facing with new knowledge, while the later statically tolerates contradictions when facing with inconsistent knowledge . Genenally speaking, nonmonotonic logic is not paraconsistent since everything would follow from a single contradiction, and paraconsistent logic is not nonmonotonic since adding new knowledge might not invalidate the previous conclusions.

As best known, classical logics are not adequate for formalizing commonsense reasoning because they are *consistent, trivial* and *monotonic*. In human commonsense reasoning, the complete and consistent knowledge is usually expected but not attained. The knowledge-based systems would be apparent intelligent only if they had capacity of reasoning with commonsense knowledge. It is well recognized that commonsense reasoning with incomplete and inconsistent knowledge is nonmonotonic *and* paraconsistent [Lin 1992b]. There is increasing interest in search for the formalisms of reasoning in the presence of incompleteness and inconsistency. As argued in [Lin 1992b], sometimes nonmonotonic logic needs paraconsistency, and paraconsistent logic needs nonmonotonicity. Nonmonotonic and paraconsistent logic is more general basis for formalizing commonsense reasoning. Interestingly, it is possible and useful to combine with both logics in the same logical framework.

The *logic of paradox LP* was proposed by Priest [1979], which is one of well-known paraconsistent logics [Priest *et al.*, 1989]. As a paraconsistent logic, LP can localize contradictions and obtain nontriviality. It has, however, one important drawback: some inferences that are classical valid are not valid in LP so that it would be too weak to permit any interesting conclusion. In [Priest,

*The author is supported in part by Natural Science Foundation (NSF number 69375011), in part by National Hi-Tech R&D '863' Project under contract 863306513A and in part by Key Project of Fundamental Research Climbing Program of China.

1989], the logic of *minimal* paradox LP_m provided as a nonmonotonic extension of LP can overcome the drawback. Therefore, nonmonotonicity can yield a solution to the weakness of paraconsistent logic.

LP_m is nonmonotonic in a sense that the inconsistency is minimal. However, it is not well suitable for dealing with reasoning with incomplete knowledge in the sense of the motivation of nonmonotonic formalism. In fact, LP_m extended LP based on the idea of circumscription. *Circumscripton CIRC* was proposed by McCarthy [1980], which is well-known one of major nonmonotonic logics [Ginsberg 1987]. $CIRC$ is a nonmonotonic extention of classical logic based on minimal models. As a nonmonotonic logic, $CIRC$ can transform incomplete knowledge into more complete one. It has, however, one basic objection: the nonmonotonic theory would be collapsed into triviality if the theory contained a single contradiction. As pointed out in [Hanks and McDermott, 1987], nonmonotonic logic in general and circumscription in special lead paradoxical problems. This is because conflicting or contradictory conslusions are split into different models. It is not guarantee that circumscription has unique model property. If nonmonotonic logic is not trivial then the problems of nonmonotonic paradox would not be arised. Therefore, paraconsistency can yield a solution to the problem of nonmonotonic logic.

In this paper we would like to revise LP_m by the application of circumscription in LP, the *logic of circumscriptive paradox* LP_c , or so-called *paraconsistent circumscrtption*, will be *truly* nonmonotonic and paraconsistent. It truns out that paraconsistent circumscription can be well characterized by minimal semantics. Simply speaking, the logic LP_c will have the nice properties of LP_m and have the ability of circumscription for nonmonotonic reasoning. It brings us advantages in two respects: it makes that nonmonotonic logic would be nontrivial when there was a contradiction, and that paraconsistent logic would be equivalent to classical logic when there was not effect of a contradiction.

The remainder of this paper is organized as follows. In Section 2, we define the minimal semantics of circumscription from the viewpoint of minimal entailment. In Sections 3 and 4, we review the semantics of logics of paradox LP and LP_m and point out some problems, respectively. In Section 5, we provide paraconsistent circumscription LP_c as a paraconsistent version of circumscription. Finally, we discuss some related works in the concluding section.

2 Minimal Entailment

Throughout this paper, we suppose L as a propositional language, and the (well-formed) formulas are defined as usual. An evaluation v assigns to each atomic proposition p in L one of the two values: 0 (*false*) or 1 (*true*). In the rest of the paper, we say that an evaluation is a model of a proposition (a set of propositions) if the proposition (every member of the set) is *true* under the evaluation. Let S be a set of formulas and A a formula. In classical logic, A is a semantic consequence of S, written as $S \models A$, defined as: *A is true in all models of S*. The *minimal entialment* will be determined by restricting entailment to the subclass of *minimal models* with respect to a partial order (preferential relation) $\prec$ over evaluations.[1]

Definition 1 *Let v, v' be two evaluations. We say that v' is* smaller than *v, written as $v' \prec v$, iff for every atomic proposition p,*

1. *if $v'(p) = 1$ then $v(p) = 1$, and*
2. *there is an atomic proposition q such that $v(q) = 1$ but $v'(q) \neq 1$.*

Definition 2 *Let S be a set of formulas. An evaluation v is a* minimal model *of S, iff*

1. *v is a model of S, and*
2. *there is no other model v' of S such that $v' \prec v$.*

Definition 3 *Let S be a set of formulas and A a formula. We say that S* minimally entails *A, written as $S \models_m A$, if A is true in all minimal models of S.*

The property of minimal entailment can be easily seen by the following simple example.

Example 1. Let p and q be two different atomic propositions. It can be ckecked that $p \vee q \models_m \neg p \vee \neg q$, for under the evaluations v_1 and v_2 ,there are only two minimal models of $p \vee q$ such that $v_1(p) = 1, v_1(q) = 0$ and $v_2(q) = 1, v_2(p) = 0$.

One of the motivations behind nonmonotonic logic is that we should transform the incomplete knowledge into less incomplete one. For example, if we had an incomplete set of propositions $S = \{p, r\}$ and a query about q, we would answer the query $\neg q$ by $S \models_m \neg q$. This is so-called *closed-world assumption* in [Reiter, 1978] since we can add the negative knowledge into the incomplete knowledge base when the positive knowledge lacked.

Minimal entailment is nonmonotonic because the minimal models of an enlarging set of premises would be changed so that some previous conclusion could be withdrawn. In fact, the minimal entailment stems from the idea of circumscription. The semantic counterpart of circumscription is just minimal entailment based on a dynamic concept of minimality for a set of atomic propositions.[2] Semantically, circumscription is defined as the minimal entailment with respect to the partial order $\prec^P$.

Definition 4 *Let v and v' be two evaluations, $v' \prec^P v$, iff*

1. *for every atom $p \in P$, if $v'(p) = 1$ then $v(p) = 1$, and*
2. *there is an atom $q \in P$ such that $v(q) = 1$ but $v'(q) \neq 1$.*

[1] Refer [Shoham, 1987] and [Lin, 1991] for some intuitions of the preferential semantics for nonmonotonic reasoning.

[2] For further explanation of the variants of circumscription see [Lifschitz, 1988] and [Lin, 1992a], among others. Theoretically, We only need to consider the simplest version of circumscription.

Definition 5 *Let S be a formula and P a tuple of atomic propositions that appears in S. Circumscription is defined as the minimal entailment, denoted by $\models_P$, with respect to $\prec^P$.*

The property of circumscription can also be seen by the following simple example.

Example 2. Let S be the conjunction of the following formulas:

$$bird_{Tweety} \wedge \neg abnormal_{Tweety} \rightarrow fly_{Tweety}$$

$$bird_{Tweety}$$

$$penguin_{Tweety} \rightarrow \neg fly_{Tweety}$$

We have that neither $S \models fly_{Tweety}$ nor $S \models_m fly_{Tweety}$. If we set $P = \{abnormal_{Tweety}, penguin_{Tweety}, bird_{Tweety}\}$ we find that $S \models_P fly_{Tweety}$.

Thus circumscription, as a major nonmonotonic logic, is indeed stronger than plain form of minimal entailment. There is, however, one basic objection for nonmonotonic logics, namely circumscription. Since circumscription is based on classical logic, everything should follow from circumscribing theory S if S contains a single contradiction. For instance, if we have a contradictory knowledge of wether the color of $Tweety$ is *yellow* or not, by incorperating $yellow_{Tweety} \wedge \neg yellow_{Tweety}$ into S, then we infer anything. Intuitively, we hope that this contradictory knowledge does not effect the conclusion of the ability of $Tweety$ to fly.

3 Logic of Paradox

The semantics of logic of paradox LP is defined as follows. An evaluation π assigns to each atomic proposition p in L one of the following three values: 0 (*false and false only*), 1 (*true and true only*) and 01 (*both true and false*).

Definition 6 *We say that a proposition p is* true *under π if $\pi(p) = 1$ or $\pi(p) = 01$; and p is* false *under π if $\pi(p) = 0$ or $\pi(p) = 01$.*

Thus LP is one kind of three-valued semantics in which under an evaluation, some proposition may be both true and false.

Definition 7 *Under an evaluation, truth values can be extended conventionally to non-atomic propositions as follows:* [3]

1. *$\neg A$ is true iff A is false;*
   $\neg A$ is false iff A is true.
2. *$A \vee B$ is true iff either A is true or B is true;*
   $A \vee B$ is false iff both A and B is false.
3. *$A \wedge B$ is true iff both A and B are true;*
   $A \wedge B$ is false iff either A or B is false.

[3] It is easy to illustrate the truth tables of LP by the definition which are exactly the same as Kleene's strong three-valued logic.

Obviously, other connectives can be easily defined as usual, *e.g.*, $A \rightarrow B$ defined as $\neg A \vee B$.

Definition 8 *Let S be a set of formulas and A a formula. A is a semantic consequence of S, written as $S \models_{LP} A$, iff A is true in all models of S, that is , for any evaluation π, if every member of S is true under π, then A is also true under π.*

The properties of LP can be easily seen by the following simple example.

Example 3. Let p and q be two different atomic propositions. It is easy to see that $p \wedge \neg p \models_{LP} p$; but $p \wedge \neg p \not\models_{LP} q$, for under the evaluation π such that $\pi(p) = 01$ and $\pi(q) = 0$, $p \wedge \neg p$ is true (actually it is both true and false) but q is not true.

One of the motivations behind paraconsistent logic is that we should not allow everything to follow from a single contradiction. Thus LP get rid of the trivial problem of classical logic, and as a paraconsistent logic, it indeed in a sense localizes contradictions. It has, however, paid a price: if S is a classically consistent proposition and A follows from S in classical logic, then A may not follow from S in LP. What fails is the disjunctive syllogism: $A, \neg A \vee B / B$, for taking an evaluation π that makes A both true and false and B false only. In fact, the disjunctive syllogism is the only classically valid inference to fail for LP in the sense that if this is added into LP then LP collapses into classical logic [Priest, 1989]. The logic of minimal paradox LP_m, among other things, overcomes this drawback.

4 Minimal Inconsistentcy

Notice that to obtain a LP counter example to the disjunctive syllogism we must render the situation inconsistent by making some formula both true and false. Thus it is natural to take consistency as a default assumption. LP_m extends LP based on the intuition that normally contradictions are rare, and we assign a truth value that is both true and false (01) to a proposition only when we are forced to do so, that is, only when the proposition is a contradiction. In fact, LP_m is based on the idea of circumscription as a kind of minimally inconsistent entailment.

Definition 9 *Let S be a set of formulas. A model π of S is* minimally inconsistent (mi) *iff there is no other model π' of S such that $\pi' \prec_m \pi$, where the partial order $\prec_m$ is defined in the following: let π and π' be two evaluations, $\pi' \prec_m \pi$, if for any atomic proposition p,*

1. *if $\pi'(p) = 01$ then $\pi(p) = 01$, and*
2. *there is an atomic proposition q such that $\pi(q) = 01$ but $\pi'(q) \neq 01$.*

Intuitively, $\pi' \prec_m \pi$ iff π contains more contradictions than π' does, and the *mi*-models are those in which the contradictions would be minimal.

Semantically, we define the entaiment of LP_m , written as $\models_{LP_m}$, as follows.

Definition 10 *Let S be a set of formulas and A a formula, $S \models_{LP_m} A$, iff every minimally inconsistent model (mi-model) of S is also a model of A.*

Similarly, the properties of LP_m can be also seen by the following simple example.

Example 4. Let p and q are two different atomic propositions. It is no hard to see that $p, \neg p \vee q \models_{LP_m} q$, for under the evaluation π that makes $\pi(p) = 01$ would not be a *mi*-model (actually there is unique *mi*-model π that makes both $\pi(q) = 1$ and $\pi(p) = 1$) ; but $\neg p \vee q, p, p \wedge \neg p \not\models_{LP_m} q$, for under the evaluation π such that $\pi(p) = 01$ and $\pi(q) = 0$, $p \wedge \neg p$ is true and q is not true.

LP_m has some nice properties. It can gives all classical consequences if the premises are consistent. This can be proved by noting that the *mi*-models of consistent premises are exactly classical models, that is, there are no assignment 01 to any proposition of consistent premises. Moreover, LP_m still validates the disjunctive syllogism even in inconsistent situation. For example, $p, \neg p \vee q, r \wedge \neg r \models_{LP_m} q$, for the evaluation π that makes $\pi(p) = 1, \pi(q) = 1$ and $\pi(r) = 01$ is the unique *mi*-model. In the other words, LP_m validates all classical inferences except where inconsistency would make them naturally doubtful anyway. As we mentioned above, the disjunctive syllogism is the only classically valid to fail for LP. Therefore LP_m is equivalent to classical logic when there was not effect of a contradiction.

A further fact of interest about LP_m is that there is no more danger of collapse into triviality with $\models_{LP_m}$ than $\models_{LP}$ (as will be seen in the next section).

LP_m is nonmonotonic only since the inconsistency is minimal similar to the plain form of minimal entailment. [4] Obviously, LP_m is not enough to capture the reasoning of transforming partial knowledge into more complete one such as circumscription in Example 2.

5 Paraconsistent Circumscription

As mentioned above, we would like to consider circumscription in the paraconsistent logic LP. We consider that there are several advantages to do so. Firstly, we define circumscription based on a paraconsistent logic so that circumscription should not allow everything to follow from a single contradiction. Secondly, we extend the paraconsistent logic by circumscription that should have the ability of circumscription to infer conclusion from incomplete knowledge and permit all classical inferences reasonably.

The logic of circumscriptive paradox LP_c will be defined as a paraconsistent version of circumscription. Technically, we only need to define the minimal semantics of LP_c by the policy of combining the criterion of the minimality of extension of propositions with the minimality of inconsistency based on LP in circumscription.

[4] Note that LP_m is weaker than the plain logic of minimal entailment to capture nonmonotonic reasoning.

Definition 11 *Let S be a set of formulas in L in which the tuple P of atomic propositions appears, and π, π' be two evaluations of LP. We define $\pi' \prec_m^P \pi$, iff*

1. *for every atom $p \in P$, if $\pi'(p) = 1$ then $\pi(p) = 1$; if $\pi'(p) = 01$ then $\pi(p) = 01$.*
2. *there is an atom $p \in P$ such that $\pi(p) = 1$ or 01 but $\pi'(p) \neq 1$ or 01.*

A model π of S is said $\prec_m^P$- minimal iff there is no other model π' of S such that $\pi' \prec_m^P \pi$.

The semantic entailment of paraconsistent circumscription LP_c, written as $\models_{LP_c}$, can be easily defined as follows.

Definition 12 *Let S be a set of formulas and A a formula. $S \models_{LP_c} A$, iff A is true in all $\prec_m^P$-minimal models of S.*

We have the following simple and important fact.

Theorem 1 *Let π be a model of a set S of formulas. There is a $\prec_m^P$-minimal model π' such that $\pi' \prec_m^P \pi$.*

Proof. Let Π be the following set of models:

$\Pi = \{\pi_i |$ *π_i is a model of S, $\pi_i \prec_m^P \pi$ and if p is true in π_i then p occurs in S* $\}$

Since Π is finite, it has a minimal element with respect to $\prec_m^P$.

The property of LP_c can be easily seen by the following simple example.

Example 5. Let S be the conjunction of the following formulas:

$bird - Tweety \wedge \neg abnormal - Tweety \rightarrow fly_{Tweety}$

$bird_{Tweety}$

$yellow_{Tweety} \wedge \neg yellow_{Tweety}$

$penguin_{Tweety} \rightarrow \neg fly_{Tweety}$

and $P = \{abnormal_{Tweety}, penguin_{Tweety}, bird_{Tweety}\}$. We have that $S \models_{LP_c} fly_{Tweety}$ as expected. That is, the contradictory knowledge of whether the color of *Tweety* is *yellow* or not does not effect the conclusion of the ability of *Tweety* to fly. Notice that we do not have that $S \models_{LP_m} fly_{Tweety}$.

Thus the logic LP_c is truly nonmonotonic and paraconsistent. In fact, we can prove the following reassuring facts which represent the relations among LP, LP_m and LP_c.

Theorem 2 *Let S be a set of formulas, if there exists a $\prec_m^P$-model of S, then $S \models_{LP_c} A$ is true for every A iff $S \models_{LP} A$ is true for every A. That is, there is no more greater of collapsing into triviality with $\models_{LP_c}$ than with $\models_{LP}$.*

Proof. It is easy to see that we only need to prove that a $\prec_m^P$-model of LP_c contains all formulas iff the model of LP does. It is straightforward since there exists a $\prec_m^P$-model π of S which is finite, and $S \models_{LP_c} A$ is true for every A, it is also the LP-model of S, and $S \models_{LP} A$ is true for every A.

Theorem 3 *Let S be a set of formulas which does not contain contradictions and A a formula, then $S \models_{LP_c} A$ iff $S \models_P A$. That is, there is no more greater of collapsing into monotonicity with $\models_{LP_c}$ than with $\models_P$.*

Proof. It is easy to check that for consistent S, a $\prec_m^P$-model of LP_c is the same as the minimal model of circumscription under taking the evaluation that no atomic proposition is both true and false. Thus if there exists a $\prec_m^P$-model π of S which is finite, and $S \models_{LP_c} A$, it is the same as the $\prec^P$-model of S, and $S \models_P A$.

Theorem 4 *Let S be a set of formulas, if there exists a $\prec_m^P$-model of S, then $S \models_{LP_c} A$ is true for every A if $S \models_{LP_m} A$ is true for every A. That is, there is no more greater of collapsing into triviality and monotonicity with $\models_{LP_c}$ than with $\models_{LP_m}$.*

Proof. It is no hard to see that we only need to prove that a $\prec_m^P$-model of LP_c contains all formulas if the model of LP_m does. It is straightforward since there exists a $\prec_m^P$-model π of S which is finite, and $S \models_{LP_c} A$ is true for every A, it is also the LP_m-model of S, and $S \models_{LP} A$ is true for every A.

Hence the logic LP_m can be viewed as a special case of the logic LP_c. The reversion of the theorem is not true just as the case of Example 5. As a corollary, we conclude this section by the following reassurance theorem of LP_m announced in [Priest, 1989].

Theorem 5 *Let S be a set of formulas, if there exists a mi-model of S, then $S \models_{LP_m} A$ is true for every A iff $S \models_{LP} A$ is true for every A. That is, there is no more greater of collapsing into triviality with $\models_{LP_m}$ than with $\models_{LP}$.*

6 Concluding Remarks

To summarize, LP, as a paraconsistent logic, can destroy the triviality to formalize reasoning in the presence of inconsistency, but it invalidates some classical inferences that seems too weak to permit reasonable conclusions. Circumscription, as a nonmonotonic logic, makes minimal inference to formalize reasoning in the presence of incompleteness, but it is trivial that everything should follow from a single contradiction. We define paraconsistent circumscription LP_c by the application of circumscription in LP. It turned out that LP_c is truly nonmonotonic and paraconsistent. We found that LP_c has advantages in two respects: it makes that nonmonotonic logic would be nontrivial when there was a contradiction, and that paraconsistent logic would be equivalent to classical logic when there was not effect of a contradiction. In other words, nonmonotonicity yields a solution to the weakness of paraconsistent logic, and paraconsistency yields to a solution to the triviality of nonmonotonic logic. The nonmonotonic and paraconsistent logic can solve the difficulties existing in nonmonotonic logic and paraconsistent logic and obtain the advantages each other. As pointed out in [Lin, 1994b], the nonmonotonic and paraconsistent logic is the formalization of reasoning with incomplete and inconsistent knowledge which provides a more general basis for formalizing commonsense reasoning in AI. We note that the logic LP_c can be viewed as a special case of so-called *fault-tolerant logic* in [Lin, 1994b], and the technique of LP_c is general enough in a sense that it could be viewed as a general approach to define nonmonotonic and paraconsistent logic.

The main results of this paper can be straightforwardly extended into first-order case. The logic LP_c can be defined based on various variants of circumscription to increase the capacity of nonmonotonic reasoning, and defined based on other paraconsistent and relevant logics for different considerations where they found applications. We, however, wish to provide first-order LP_c by considering to provide a satisfactory proof theory. As pointed out by [Priest, 1989], there was not a satisfactory proof theory for LP_m , though some proof theories for LP were introduced. Although circumscription was originally provided as a simple schema of proof theory based on classical logic, it is not available for LP since paraconsistent logic lacks necessarily classical inferences to formulate circumscriptive shema. Fortunately, [Lin, 1993] has proposed a minimal tableaux for the logic LP_m as a satisfactory proof theory, and hence it is not difficult to present a proof theory for LP_c [Lin, 1994a]. Futhermore, it has remained to see if first-order LP_c has the same propeties as in propositional level. We conjecture that first-order LP_c would have better properties, though circumscription lacks completeness result in general. Hopefully, first-order paraconsistent circumscription would shed new sight to solve the paradoxical problems of nonmonotonic reasoning.

Acknowledges:

The author would like to thank Wei Li, Fangzhen Lin, Vladimir Lifschitz, Yoav Shoham, Grogori Schwarz, Jiahuai You and Jun Gu. Thanks also the referees for useful comments on improving this paper.

References

[Ginsberg 1987] Ginsberg. M (Ed.), *Readings in Nonmonotonic Reasoning*, Morgan Kaufmann, 1987

[Hanks and McDermott, 1987] Hanks. S and McDermott. D, *Nonmonotonic Reasoning and Temporal Projection*, Artificial Intelligence 35 (1987)

[Lifschitz, 1988] Lifschitz. V, *Circumscriptive Theories: A Logic-Based Framework for Knowledge Representation*, J. of Philosophical Logic 17 (1988), 391-441

[Lin, 1991] Lin. Z, *Parametric Systems: A Uniform Basis for Monotonic and Nonmonotonic Logics*, Pattern Recognition and Artificial Intelligence 4 (1991), 20-27

[Lin, 1992a] Lin. Z, *A Generalization of Circumscription*, J. of Computer Science and Technology 7 (1992), 97-104

[Lin 1992b] Lin. Z, *Reasoning with Incomplete and Inconsistent Information*, Proceedings of ICIIPS'92, IAP Press, 1992, 382-388

[Lin, 1993] Lin. Z, *A Note on Proof Theories of Logics of Paradox*, STU-AI-TR-41, Computer Science Department, Shantou University, 1993; also in: Proceedings of Chinese Conference on Intelligent Computer, 1994

[Lin, 1994a] Lin. Z, *Paraconsistent Circumscription*, Working Papers on the 3th International Symposium on Artificial Intelligence and Mathematics, January 2-5, Florida, 1994

[Lin, 1994b] Lin. Z, *Fault-Tolerant Reasoning*, Proceedings of the 2nd World Congress on Expert Systems, January 10-14, Lisbon, 1994

[McCarthy, 1980] McCarthy. J, *Circumscription – A Form of Non-Monotonic Reasoning*, Artificial Intelligence 13 (1980), 27-39

[Priest, 1979] Priest. G, *Logic of Paradox*, J. of Philosophical Logic 8 (1979), 219-241

[Priest, 1989] Priest. G, *Reasoning about Truth*, Artificial Intelligence, 39 (1989), 231-244

[Priest *et al.*, 1989] Priest. G *et al.* (Eds.), *Paraconsistent Logic: Essays in the Inconsistency*, Philosophia Verlag, 1989

[Reiter, 1978] Reiter. R, *On Closed World Data Base*, in: *Logic and Data Base*, Gallaire. H and Minker. J (Eds.), Plenm Press, 1978

[Shoham, 1987] Shoham. Y, *Nonmonotonic Reasoning: Meaning and utility*, Proceedings of IJCAI-87, Morgan Kaufmann, 1987, 388-393

Two Cumulativity Results On J- and PJ-Default Logics

Jia-Huai You*
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
you@cs.ualberta.ca

Liwu Li
School of Computer Science
University of Windsor
Windsor, Ontario, Canada N9B 3P4
liwu@cs.uwindsor.ca

Abstract

This paper reports two cumulativity results for J- and PJ-default logics formalized by Delgrande and Jackson [Delgrande and Jackson, 1991] in the context of skeptical reasoning: (1) PJ-default logic is cumulative, and (2) J-default logic is not cumulative in general but there is a nontrivial class of theories for which the cumulativity property can still hold. We also investigate semantical implications of the Cumulative Default Logic [Brewka, 1991] and compare it with J-default logic with regard to cumulativity for skeptical reasoning.

1 Introduction

In nonmonotonic reasoning, the traditional measure of complexity seems not indicative of computational difficulties, since it is normal that the inference process endorsed by a formalism is totally undecidable in general and intractable in the propositional case.

Some authors have addressed *behavioral regularity* of nonmonotonic inferences. Makinson [Makinson, 1988] has investigated the regularity properties for some inference formalisms. One property is named *cumulativity*. Intuitively, the property guarantees that adding a theorem derived from a set of premises to the set will not eliminate any previous theorem. The property can be formally described by the condition:

When we have $\mathcal{W} \mid\sim x$, $\mathcal{W} \mid\sim y$ iff $\mathcal{W} \cup \{x\} \mid\sim y$,

where $\mid\sim$ denotes an arbitrary inference relation. Makinson's result shows that Reiter's default logic is not cumulative [Makinson, 1988; Reiter, 1980].

The cumulativity property is closely related to the problem of whether default inferences can be carried out incrementally. In light of the difficulties evidenced in nonmonotonic reasoning, there exist systems which attempt to compile some of the important queries and store them for future use (cf. [Sattar and Goebel, 1991]). The property of cumulativity is essential for such an approach.

Recently, some consistency-based default logics have been proposed to address some of the problems associated with Reiter's default logic [Brewka, 1991; Delgrande and Jackson, 1991]. J- and PJ-default logics of Delgrande and Jackson [Delgrande and Jackson, 1991] are intended to solve the problem of unintuitive extensions and missing extensions in Reiter's logic. (The similar idea was explored earlier in [Lukaszewicz, 1987].) Reasoning in J-default logic is a natural nondeterministic iterative process of augmenting a given theory, where each augmentation step is required to yield a consistent new theory. A sequence of partial extensions built this way is thus nondecreasing. When no more beliefs can be added without causing inconsistency, a J-extension is reached. PJ-default logic converts all defaults in a given default theory to prerequisite-free defaults, whose J-extensions are then defined as the semantics of PJ-default logic. PJ-default logic has been used in belief revision [Ghose *et al.*, 1993].

This paper studies the cumulativity property of J- and PJ-default logics. We also compare them with the Cumulative Default Logic (CDL) [Brewka, 1991], and provide insights into semantical implications of CDL. Two results are reported. First, we show that the class of prerequisite-free J-default theories possess the cumulativity property. This leads to the conclusion that PJ-default logic is cumulative. This result is presented in Section 3. Incidentally, the key reason that the revision method given in [Ghose *et al.*, 1993] works smoothly seems to mainly due to the fact that PJ-default logic is cumulative.

Secondly, we demonstrate that the precise reason for PJ-default logic to be cumulative is because it has weakened the skeptical reasoning ability of J-default logic, resulting in possible loss of intuitive skeptical beliefs. Cumulativity seems inherently difficult for skeptical reasoning. Recently proved cumulativity results are all for the well-founded semantics and its extensions for

*Currently on leave with Department of Computer Science, Rice University, Houston, Texas.

logic programs (see, for example, [Dix, 1991; Dix, 1992; Dix, 1994; Li and You, 1991]). However, we show that there is a nontrivial class of J-default theories for which the cumulativity property still holds. The details can be found in Section 5.

CDL is cumulative both for skeptical reasoning and choice reasoning. However, the notion of belief in CDL is quite different from the one in Reiter's logic and J- and PJ-logics. This may cause strange behavior for some default theories. This is discussed in details in Section 4.

The next section introduces J- and PJ-default logics, followed by sections on the cumulativity results as well as on CDL.

2 J- and PJ-Default Logics

We assume a first order language that consists of usual well-formed formulas (wffs) over an alphabet $\mathcal{A}$. Recall that a default theory is a pair (D, W), where D is a set of defaults and W is a set of first-order formulas. A default is an inference rule of the form $A : B_1, ..., B_k/C$ with $k \geq 0$, where A, B_i's and C are formulas, and A is called the *prerequisite*, B_i's the *justifications* and C the *consequent*. We will be dealing with default theories with *closed* defaults. An *open* default is a shorthand denoting the set of all instantiated defaults.

J-default logic is defined in terms of J-extension. The following definition is a reformulation of J-default logic for arbitrary default theories. This reformulation is also crucial in simplifying proofs of our cumulativity results.

Definition 2.1 $\langle E, H\rangle$ is a J-extension of a default theory (D, W) iff

$$\langle E, H\rangle = \langle \bigcup_{i=0}^{\infty} E_i, \bigcup_{i=0}^{\infty} H_i\rangle,$$

where

$$E_0 = W \text{ and } H_0 = \emptyset,$$

and for $i \geq 0$, if $A : B/C \in D$, $E_i \models A$, and $E_i \cup H_i \cup \{B, C\}$ is consistent, then

$$E_{i+1} = E_i \cup \{C\}$$
$$H_{i+1} = H_i \cup \{B\}.$$

We sometimes also refer to E as a J-extension when we are primarily concerned with the derived beliefs, with the understanding that there exists a set H of formulas which are used to derive E. □

Remarks:

1. In the above definition, we define a J-extension E as a set of derived formulas instead of $Cn(E)$, where Cn denotes the familiar Tarskian consequence operator. This is only a tactic choice since for any formula ϕ, $E_i \models \phi$ iff $\phi \in Cn(E_i)$. We define E this way only because it is more convenient technically in proving the cumulativity results in Section 3.
2. Our primary interest in a J-extension $\langle E, H\rangle$ is the set E of derived beliefs. When we refer to E as a J-extension, however, it should be mentioned that the corresponding assumption set H may not be unique. For example, with the default theory $(D, \emptyset)$ with $D = \{: a/b, : \neg a/b\}$, there are two J-extensions with identical E.
3. For J-default logic, a default $A : B_1, ..., B_k/C$ is equivalent to $A : B_1 \wedge \cdots \wedge B_k/C$. Therefore, we only consider defaults with a single justification.

In P-default logic, a normal default $(A : B/B)$ is replaced with a default without prerequisite $(:A \supset B/A \supset B)$ and a semi-normal default $(A{:}B \wedge C/C)$ is replaced by $(:A \supset (B \wedge C)/A \supset C)$. This allows a default to fire without requiring that the prerequisite be proved. Let us call such a resulting default a P-default and resulting theory a P-default theory. The J-extensions of a P-default theory are then defined as the semantics of PJ-default logic. It is known PJ-default logic is closely related to Poole's framework for default reasoning [Poole, 1988].

According to Reiter, there are two reasoning modes in using default logic: each arbitrarily chosen extension can be seen as an acceptable set of beliefs, or the truth of a formula is determined by whether it is contained in all extensions. The former is called *choice reasoning* and latter *skeptical reasoning*. This paper is mainly concerned with skeptical reasoning.

The rest of this section provides additional notations.

Notations:

1. Given a default theory (D, W), we define the following sets:

$$Preq(D) = \{A \mid A : B/C \in D\}$$
$$Just(D) = \{B \mid A : B/C \in D\}$$
$$Cons(D) = \{C \mid A : B/C \in D\}.$$

2. If E is a J-extension of (D, W), then by Definition 2.1, there exists a sequence E_0, E_1, ..., such that $E = \bigcup_{i=0}^{\infty} E_i$. We denote such a sequence by $\{E_i\}$ and say that the sequence *leads to* E.

Definition 2.2 Let E and H be sets of formulas such that $E \cup H$ is consistent. The set of *generating defaults* of $\langle E, H\rangle$, with respect to a default theory $\Delta = (D, W)$, denoted as $G_\Delta(E, H)$, is defined as:

$$G_\Delta(E, H)$$
$$= \{A : B/C \mid A : B/C \in D, \{A, C\} \subseteq E \text{ and } B \in H\}.$$

We will simplify $G_\Delta(E, H)$ to $G(E, H)$ when the default theory is understood. □

The following lemma is useful in proving the cumulativity results in the next section.

Lemma 2.1 *Assume $Preq(D) = \emptyset$ for default theory (D, W). Let also $\langle E, H\rangle$ be a J-extension of (D, W). Then,* (i) *$E = Cons(G(E, H)) \cup W$ and $H = Just(G(E, H))$; and* (ii) *for any $(: B/C) \in D/G(E, H)$, $E \cup H \cup \{B, C\}$ is inconsistent.*

Proof:

For any J-extension $\langle E, H\rangle$, one can construct a sequence $\{E_i, H_i\}$ by using each default from $G(E, H)$ at a step, in any order. This shows (i).

For (ii), suppose $E \cup H \cup \{B, C\}$ is consistent for some default $(: B/C) \in D/G(E, H)$. Then the default $(: B/C)$ can be used to construct $\langle E, H\rangle$ so that $C \in E$ and $B \in H$. This contradicts the assumption that $(: B/C) \in D/G(E, H)$. □

3 Cumulativity Results

In this section we show that J-logic for default theories with only prerequisite-free defaults is cumulative. This leads to the result that PJ-default logic is cumulative.

In the following, by $(D,W) \mid\!\sim x$, we mean for each J-extension E of (D,W), $E \models x$.

Theorem 3.1 *Let (D,W) be a default theory with $Preq(D) = \emptyset$ and assume $(D,W) \mid\!\sim x$. Then, for every J-extension E of (D,W) there exists a J-extension E' of $(D, W \cup \{x\})$ such that $Cn(E) = Cn(E')$, and vice versa.* □

We postpone the proof to Appendix.

This result can be extended to default theories (D,W) where each prerequisite therein can be derived directly from W.

Corollary 3.1 *Let (D,W) be a default theory and assume $(D,W) \mid\!\sim x$. If $W \models A$ for each default $(A : B/C) \in D$, then, for every J-extension E of (D,W) there exists a J-extension E' of $(D, W \cup \{x\})$ such that $Cn(E) = Cn(E')$, and vice versa.*

Proof:

Let $D' = \{: B/C \mid A : B/C \in D\}$. Clearly, $\langle E, H\rangle$ is a J-extension of (D,W) if and only if it is a J-extension of (D', W). The claim then holds by Theorem 3.1. □

We point out that the prerequisite-free condition does not make Reiter's default logic cumulative. Consider the following default theory:

$$D = \{: \neg a/b,\ : \neg b/a,\ : \neg a/c,\ : \neg c/c\}.$$

The defaults are all prerequisite-free, but $(D, \emptyset)$ is not cumulative.

J-default logic is not guaranteed with the cumulativity property when prerequisites are present in defaults.

Example 3.1 Let $\Delta = (D, \emptyset)$ where

$$D = \{: \neg b/a,\ : \neg b/c,\ c : \neg a/b\}.$$

There is exactly one J-extension that contains a and c. Adding c to W makes the last default applicable and results in an additional J-extension. □

The question arises as to what makes PJ-default logic cumulative while J-default logic is not. The following example shows that the price for attaining the cumulativity property for PJ-default logic is its weakening of the skeptical reasoning ability, resulting in possible loss of skeptical beliefs.

Example 3.2 Consider again the default theory in Example 3.1.

The first two defaults therein are already P-defaults. The third default can be transformed to a P-default resulting in a P-default theory as:

$$D_P = \{: \neg b/a,\ : \neg b/c,\ : c \supset (\neg a \wedge b)/c \supset b\}.$$

Now there are two J-extensions for theory $(D_P, \emptyset)$. Besides the J-extension $E_1 = \{a, c\}$, we have an additional one $E_2 = \{a, c \supset b\}$. As the result, the skeptical belief c of $(D, \emptyset)$ is no longer a skeptical consequence of $(D_P, \emptyset)$. From this example we see that the reason that PJ-default logic possesses the cumulativity property is precisely because it is a weaker logic than J-logic with regard to skeptical reasoning. □

4 Cumulative Default Logic: Abstract Beliefs versus Compositive Beliefs

In this section we give a reformulated definition of CDL and show it is essentially the same as J-logic. The *only* difference is that the former semantically relies on *compositive beliefs*, the beliefs semantically *inseparable* from the formulas used to derive them, while the latter accommodates *abstract beliefs* which are normal first order formulas. It is only this difference that makes the former cumulative and the latter not cumulative in general.

4.1 Cumulative Default Logic

In Brewka's Cumulative Default Logic (CDL) [Brewka, 1991], objects being manipulated are called *assertions*. An assertion is of the form

$$\langle A : \{j_1, \ldots, j_m\}\rangle,$$

where A is a formula, called the (asserted) *formula* of the assertion, and j_i's are formulas, called the *supports* of A.

Default theories (D,W) with assertion set W are called *assertion default theories*. A default $(A : B/C)$ can be applied if an assertion $(A : J)$ holds for some support J and $\{B, C\}$ is consistent with the formulas of the assertions, either derived or in W, and their supports. The derived formula C is then attached with the support of A, along with B and C to form new assertion $\langle C : J \cup \{B, C\}\rangle$.

Given a set S of assertions, we denote by $\Gamma(S)$ the set of all formulas that occur in any assertion in S, either as an asserted formula or as a support. That is,

$$\Gamma(S) = \{C \mid \langle A : J\rangle \in S \text{ such that } C = A \text{ or } C \in J\}.$$

The propagation of supports is important. We say an assertion $\langle A : J\rangle$ is derived from a set S of assertions, denoted as $S \vdash_{supp} \langle A : J\rangle$, if there exist assertions $\langle B_1 : J_1\rangle, \ldots, \langle B_k : J_k\rangle$ in S such that $\{B_1, \ldots, B_k\} \models A$ and $J = \{J_1, \ldots, J_k\}$.

We are now ready to define CDL-extension based on an iterative construction. Once again, our definition does not use the closure of first order consequences.[1]

Definition 4.1 E is a CDL-extension of an assertion default theory (D,W) iff

$$E = \bigcup_{i=0}^{\infty} E_i,$$

where

$$E_0 = W$$

and for $i \geq 0$, if $A : B/C \in D$, $E_i \vdash_{supp} \langle A : J\rangle$, and $\Gamma(E_i) \cup \{B, C\}$ is consistent, then

$$E_{i+1} = E_i \cup \{\langle C : J \cup \{B\}\rangle\}.$$ □

[1] There are also some nonessential differences between our reformulation and Brewka's definition. For example, we only record justifications of defaults as *supports* while in Brewka's definition, consequents are also included as part of the supports in the derivation.

From our reformulations of J-default logic and CDL, it is easy to see a one-to-one correspondence between the extensions of the two logics.

Given a set S of assertions, let us denote by $Form(S)$ the set of asserted formulas of S, that is:

$$Form(S) = \{A \mid \langle A : J \rangle \in S\}.$$

Theorem 4.1 *Let (D, W) be a default theory and (D, W_a) be the corresponding assertion default theory with $W_a = \{\langle A : \emptyset \rangle \mid A \in W\}$.*

(i) *If E is a CDL-extension of (D, W_a), then there exists a set H of formulas such that $\langle Form(E), H \rangle$ is a J-extension of (D, W).*

(ii) *For any J-extension $\langle E, H \rangle$ of (D, W), there exists a CDL-extension E' of (D, W_a) such that $Form(E') = E$.*

Proof: By using an induction on the sequence of partial extensions in both cases. □

This result says that computing a CDL-extension is essentially the same process of computing the corresponding J-extension, and vice versa. This implies that when computational efficiency is concerned, neither offers more advantage over the other.

4.2 Abstract Beliefs versus Compositive Beliefs

We assume a knowledge base that consists of the knowledge:

> *Tweety has wings and there is no evidence suggesting its wings being degenerated, so we believe Tweety flies.*

In CDL we do not have the abstract belief *Tweety flies*; we are only able to derive the compositive belief:

> *Tweety flies because there is no evidence suggesting its wings are degenerated.*

Arguably, the user of a knowledge base sometimes does want to know what assumptions have been made in deriving an abstract belief. Indeed, a default logic is a logic of plausible belief inference. Assumptions and derived beliefs are nevertheless two distinct concepts. Because of this, a derived belief and its assumptions should be separable to the user of a knowledge base. Compositive beliefs however do not separate a derived belief with its associated assumptions. This is an undesirable feature, in our opinion, particularly when the size of the knowledge base is large and the derivation of a belief involves a long sequence; in such a case, a compositive belief may contain a large number of assumptions. In many situations, *a compositive belief is virtually a reflection of the entire derivation sequence.*

Compositive beliefs have another drawback. Suppose we are able to derive some totally irrelevant but consistent compositive beliefs, we can then have many confusing and unintuitive compositive beliefs with the same asserted formula. For example, suppose we also have a compositive belief:

> *Fred is healthy because there is no evidence suggesting he is sick,*

we would then have a seemingly strange compositive belief:

> *Tweety flies because there is no evidence suggesting its wings are degenerated and there is no evidence suggesting Fred is sick.*

If we ask the knowledge base if the above statement is true, strangely, the answer is *yes*. To see this, let us represent the above problem by

$$W = \{wings\}$$
$$D = \{wings : \neg deg/fly, \ : \neg sick/healthy\}$$

Then, the assertion $\langle fly : \{\neg deg, \neg sick\} \rangle$ is indeed in the unique CDL-extension.

This second problem may be resolved by requiring support be minimal. However, it is the mechanism of recording all assumptions that limits our ability to think of a compositive belief in terms of its asserted belief. From Theorem 4.1, we know that the *only* reason that CDL is cumulative while J-default logic is not is CDL's employment of compositive beliefs. Therefore, it is impossible to reduce a compositive belief to its abstract counterpart and at the same time retain the cumulativity property.

The failure to accommodate abstract beliefs can sometimes cause strange behaviors to arise in skeptical reasoning.

Example 4.1 Assume an assertion default theory (D, W) where

$$W = \emptyset \text{ and } D = \{: a/b, \ : \neg a/b\}.$$

There are two CDL-extensions, one containing $\langle b : \{a\} \rangle$ and the other having $\langle b : \{\neg a\} \rangle$. No skeptical belief about b can be drawn in CDL, even though b is intuitively a skeptical belief. □

From Theorem 3.1, and also by Theorem 4.1 on the one-to-one correspondence between J-extensions and CDL-extensions, we know that when defaults in a default theory are prerequisite-free, compositive beliefs can be reduced to their abstract counterparts without losing the cumulativity property. This means that cumulativity is guaranteed without resorting to the attachment of assumptions. For the preceding example for instance, we can have b as a skeptical belief.

5 Practical Considerations of Cumulativity

Practically, a nonmonotonic reasoning formalism can be affected by the cumulativity property in two aspects: computational efficiency and computational strategy.

Consistency-based default logics, such as J-logic and CDL, have indeed gained some computational efficiency over Reiter's default logic. For example, it is easy to see that when defaults in a propositional default theory (D, W) only involve literals and W is a set of literals, the problem of computing *an* extension is a tractable problem.[2] On the other hand, the one-to-one correspondence between J-extensions and CDL-extensions implies

[2] In fact, each default can be applied at most once, the problem of consistency check for the class of theories considered here has a polynomial algorithm, and an extension can be incrementally constructed.

that neither offers more advantage over the other in terms of computational efficiency. Thus, cumulativity may only affect computational strategies, such as pre-compiling queries and incremental computation. Now, if abstract beliefs and a full skeptical reasoning capability are desired, J-default logic seems to be a good candidate.

In this section we consider how to make pre-compiling queries and incremental computation a safe computational strategy for some important classes of J-default theories. First, we follow the same line of argument by Poole [Poole, 1991]: if a default cannot be used to derive any belief, there is no reason for it to be in the theory. Due to Theorem 3.1, we need only be concerned with defaults with non-empty prerequisites. We then show that under this approach, there is an important class of J-default theories that possess the cumulativity property.

Definition 5.1 Let $\Delta = (D, W)$ be a default theory. A default $d \in D$ of the form $A : B/C$ with a non-empty prerequisite A is said to be *irrelevant* if for every J-extension $\langle E, H\rangle$ of (D, W), $d \notin G(E, H)$; otherwise, it is *relevant*. We denote by $\mathcal{F}(\Delta)$ the set of all relevant defaults in D. □

Theorem 5.1 *$\langle E, H\rangle$ is a J-extension of a default theory $\Delta = (D, W)$ iff it is a J-extension of $(\mathcal{F}(\Delta), W)$.*
Proof: It follows directly from Definition 5.1 □

We argue that the semantics of a default theory should be defined with respect to relevant defaults in $\mathcal{F}(\Delta)$ as far as reasoning with the (fixed) theory is concerned. In such a setting, incremental computation and pre-compilation can be carried out with respect to the defaults in $\mathcal{F}(\Delta)$. Non-applicable defaults are not abandoned; they will be used to accommodate possible changes of the theory. This is the case when W needs to be revised according to newly acquired knowledge, say x, irrelevant defaults can then be re-calculated with respect to $(D, W \cup \{x\})$ to reflect the change of the theory.[3]

The definition of irrelevant defaults is based on all J-extensions. We now show that one need not compute all extensions in order to determine whether a default is irrelevant. There is a simpler way to identify irrelevant defaults.

To illustrate the idea, we consider a simple example.

Example 5.1 Consider the default theory given earlier in Example 3.1:

$$W = \emptyset$$
$$D = \{: \neg b/a,\ \ : \neg b/c,\ \ c : \neg a/b\}$$

The reason that the third default above initially was not applicable is that there is no way c can be confirmed by consistent justification and consequent. To see this, let's reduce the third default using the second. We then get $(: \neg b \wedge \neg a/b)$. Because its justification is inconsistent with its consequent, the default cannot be fired. If this non-applicable default is "ignored", then the default theory is guaranteed with the cumulativity property. □

Let's exam a more involved example.

[3]This will affect pre-compiled queries. However, due to the nonmonotonic nature of default reasoning, they need to be re-complied anyway.

Example 5.2 [Brewka, 1991] We assume a default theory (D, W) with

$$W = \{dog \vee bird \supset pet,\ \ dog \supset \neg bird,\ \ sings\}$$
$$D = \{pet : dog/dog,\ \ sings : bird/bird\}$$

The only J-extension (which coincides with the unique extension in Reiter's default logic) contains *bird* and *pet*. Adding *pet* to W makes the first default applicable and results in an additional J-extension.

First, we see that PJ-default logic has weakened skeptical reasoning. The corresponding P-defaults are:

$$: pet \supset dog/pet \supset dog,$$
$$: sings \supset bird/sings \supset bird$$

Besides the previous J-extension that contains *bird*, there is one more J-extension that contains $pet \supset dog$ and derives $\neg bird$. Therefore, we no longer have the intuitive belief *bird* as a skeptical belief. Clearly, this is caused by allowing every default to have a chance to be applied without its prerequisite being proved, and by doing so, a different extension is introduced.

The reason that the default $(sings : bird/bird)$ is applicable is clear: the prerequisite *sings* can be reduced to *true* and thus the default can be reduced to an equivalent one without any prerequisite. This is not the case for the default $(pet : dog/dog)$, as shown by the following reasoning.

Since $W \cup \{dog\} \vdash pet$ and $W \cup \{bird\} \vdash pet$, the applicability of this default depends on that of $(dog : dog/dog)$ and $(bird : dog/dog)$. The first is circular and can be thrown away. The second can be reduced, using $(sings : bird/bird)$, to

$$sings : bird \wedge dog/dog,$$

and further to

$$: bird \wedge dog/dog,$$

whose justification is inconsistent with W. □

The preceding two examples suggest that a default with a non-empty prerequisite is not applicable if it cannot be reduced to a prerequisite-free default whose justification and consequent are consistent with W.

We now define this process more formally. To simplify the notation, let us consider defaults $A : B/C$ where A and C are literals and B is a conjunction of literals.

Definition 5.2 Let $\Delta = (D, W)$ be a default theory.

A default $(A : B/C) \in D$ may be transformed repeatedly by the following rules:

(a) if $W \models A$, then get $(: B/C)$;

(b) if $(P : Q/Z) \in D$, $\{P, Q, Z\} \cup W$ is consistent, and $W \cup \{Z\} \vdash A$, then get $P : B \wedge Q/C$.

A default $(A : B/C) \in D$ is said to be *applicable* if and only if there is a default $(: B \wedge R/C)$, where $R = Q_1 \wedge ... \wedge Q_n$, for some $n \geq 0$, which is derived from $(A : B/C)$ using the transformation rules given above, such that $\{B, R, C\} \cup W$ is consistent. Otherwise, it is said to be *non-applicable* □

If a default $(A:B/C)$ is applicable, then there must exist a J-extension $\langle E, H\rangle$ and a sequence $\{E_n\}$ leading to E so that $(A:B/C)$ is in $G(E,H)$. On the converse, if $(A:B/C)$ is a generating default of J-extension $\langle E, H\rangle$, then A can be consistently derived using W and other defaults, and hence can be reduced to a prerequisite-free default. This means that the notion of *applicability* (resp. *non-applicability*) coincides with that of *relevance* (resp. *irrelevance*) given earlier.

Under this approach, we are able to identify a non-trivial class of default theories for which the cumulativity property is guaranteed. The following theorem is an extension of Theorem 3.1.

Theorem 5.2 *Let $\Delta = (D,W)$ be a default theory with $\mathcal{F}(\Delta) = D$. Suppose $(D,W) \vdash x$. Further assume for each default $A:B/C \in D$ where $W \not\models A$, $A:B/C \in G(E,H)$ for any J-extension $\langle E,H\rangle$ of (D,W). Then, for every J-extension E of (D,W) there exists a J-extension E' of $\Delta' = (D, W\cup\{x\})$ such that $Cn(E) = Cn(E')$, and vice versa.*

The condition in the theorem essentially says that any applicable default must be a generating default of every J-extension. The condition seems very easy to be satisfied in many practical situations; for example, it is trivially satisfied for default theories with exactly one J-extension. A proof of this theorem is included in Appendix.

We note that the approach described here is not applicable to Reiter's logic, since default theories with prerequisite-free defaults are not guaranteed with cumulativity, and hence every default in a default theory is potentially irrelevant.

6 Summary

Recently proposed consistency-based default logics, like J- and PJ-logics of Delgrande and Jackson and CDL of Brewka, have shown very good promises in default reasoning by improving Reiter's logic for different purposes.

This paper has provided a unified reformulation of these consistency-based logics, based on an iterative construction of extensions. We have proved that prerequisite-free J-default theories possess the cumulativity property. Two important implications immediately follow: (i) PJ-default logic is cumulative, and (ii) for this class of default theories, it is not necessary to make the process of a reasoned belief explicit as in CDL. We have shown that PJ-default logic attains the cumulativity property by weakening skeptical reasoning, resulting in sometimes losing intuitive skeptical beliefs. Since J-default logic can provide abstract beliefs and has a reasonable capability of skeptical reasoning, we have emphasized on an approach to gain the practical benefits of cumulativity by identifying non-applicable defaults. This approach enables a safe application of computational strategies such as precompilation and incremental computation for some important classes of default theories.

Acknowledgement: Both authors are supported by grants from the Natural Sciences and Engineering Council of Canada. The final revision was carried out while Jia-Huai You was on sabbatical with Rice University.

7 Appendix

Proof of Theorem 3.1:

We show this result in two steps, one about the if part of the cumulativity and other about the only-if part. It turns out that only the only-if part requires the condition $Preq(D) = \emptyset$. The if part is given in Theorem 7.1 and the only-if part is proved in Theorem 7.2.

Theorem 7.1 *Let E be a J-extension of default theory (D,W). Assume $(D,W) \vdash x$. Then, there exists a J-extension E' of $(D, W\cup\{x\})$, such that for any sequence $\{E_i\}$ leading to E there exists sequence $\{E'_i\}$ leading to E' with $E_i \cup \{x\} = E'_i$ for each $i \geq 0$, and $Cn(E) = Cn(E')$.*

Proof:

The claim trivially holds when W is inconsistent.

Suppose W is consistent. From $(D,W) \vdash x$, and the fact that E is a J-extension of (D,W), we have $E \models x$, and $E \cup H \cup \{x\}$ is consistent.

Now, from sequence $\{E_i\}$, we show by induction that $\{E'_i\}$ can be constructed which leads to a J-extension of $(D, W\cup\{x\})$.

Basis: For $E_0 = W$ and $H_0 = \emptyset$, let $E'_0 = W \cup \{x\}$ and $H_0 = H'_0$. Clearly, E'_0 is consistent.

Inductive step: Assume $E_i \cup \{x\} = E'_i$ and $H_i = H'_i$.

Let E_{i+1} be deduced from E_i by the default $A_i : B_i/C_i \in D$. We have $E_i \models A_i$, $E_i \cup H_i \cup \{B_i, C_i\}$ is consistent, and $E_{i+1} = E_i \cup \{C_i\}$.

Since $E \models E_i$, $H \models H_i$, and $E \cup H \cup \{x\}$ is consistent, we get

$$E_i \cup \{x\} \cup H_i \cup \{B_i, C_i\}$$

is consistent.

From $E_i \models A_i$ and $E_i \cup \{x\} = E'_i$, we have $E'_i \models A_i$. It follows that

$$E'_i \cup H'_i \cup \{B_i, C_i\}$$

is consistent. Thus, we can extend E'_i to $E'_{i+1} = E'_i \cup \{C_i\}$. Hence we have

$$E_{i+1} \cup \{x\} = E'_{i+1} \text{ and } H_{i+1} = H'_{i+1} = H_i \cup \{B_i\}.$$

Therefore, $E \cup \{x\} = E'$. By the fact that $E \models x$, we have $Cn(E) = Cn(E')$.

To show that E' so constructed is a J-extension of $(D, W\cup\{x\})$, we only need to show that no default in D can be used to properly extend E'.

That E is a J-extension of (D,W) implies there is no default $A:B/C \in D$ such that $E \models A$, $E \cup H \cup \{B, C\}$ is consistent, and $E \cup \{C\}$ is a proper extension of E, i.e., $E \not\models C$. The same clearly holds for E' because $E \cup \{x\} = E'$ and $E \models x$. Thus, E' is a J-extension of $(D, W\cup\{x\})$. □

Theorem 7.2 *Let (D,W) be a default theory with $Preq(D) = \emptyset$ and assume $(D,W) \vdash x$. Then, for any J-extension E of $(D, W\cup\{x\})$, there exists a J-extension E' of (D,W), such that for any sequence $\{E_i\}$ leading to E for default theory $(D, W\cup\{x\})$ there exists sequence $\{E'_i\}$ leading to E' for default theory (D,W) with*

$E_i = E'_i \cup \{x\}$ *for each* $i \geq 0$ *and* $Cn(E) = Cn(E')$.
Proof:

Assume W is consistent, otherwise it is trivial.

We first show that there exists such a sequence $\{E'_i\}$.

Basis: For $E_0 = W \cup \{x\}$ and $H_0 = \emptyset$, let $E'_0 = W$ and $H'_0 = \emptyset$.

Inductive step: Assume $E_i = E'_i \cup \{x\}$, and $H_i = H'_i$.

Consider a default $(:B_i/C_i) \in D$ such that $E_i \cup H_i \cup \{B_i, C_i\}$ is consistent. Assume the default $(:B_i/C_i)$ is used to extend E_i, and we have $E_{i+1} = E_i \cup \{C_i\}$ and $H_{i+1} = H_i \cup \{B_i\}$.

Since $E_i = E'_i \cup \{x\}$ and $H_i = H'_i$, $E'_i \cup H'_i \cup \{B_i, C_i\}$ is consistent; and hence the default $(:B_i/C_i)$ can be used to extend E'_i. Therefore, we have

$$E_{i+1} = E'_{i+1} \cup \{x\} \text{ and } H_{i+1} = H'_{i+1} = H_i \cup \{B_i\}.$$

Therefore, $E = E' \cup \{x\}$. If we can show that E' is a J-extension of (D, W), then by the hypothesis $(D, W) \mid\sim x$, we will have $E' \models x$ and it follows $Cn(E) = Cn(E')$.

Since E is a J-extension of $(D, W \cup \{x\})$ and $Preq(D) = \emptyset$, by Lemma 2.1, for any $(:B/C) \in D/G(E, H)$, $E \cup H \cup \{B, C\}$ must be inconsistent. We show that $E' \cup H' \cup \{B, C\}$ must be inconsistent, too.

Assume not. Then by the definition of J-extension, there exists a J-extension E'' of (D, W) such that $E' \subseteq E''$, $G(E, H) = G(E', H') \subset G(E'', H'')$, and

$$E'' \cup H'' \cup Cons(G(E'', H'')) \cup Just(G(E'', H''))$$

is consistent.

Since E'' is a J-extension of (D, W), we have $E'' \models x$. It follows $E'' \models E' \cup \{x\}$. By the fact that $E = E' \cup \{x\}$, we have $E'' \models E$. Since $H'' \models H$, and $G(E'', H'') - G(E, H) \neq \emptyset$, there exists some default $(:B'/C') \in G(E'', H'')/G(E, H)$ such that $E \cup H \cup \{B', C'\}$ is consistent. This contradicts the fact that E is a J-extension of $(D, W \cup \{x\})$ and no default outside its generating default set may be applicable.

Therefore, E' is a J-extension of (D, W). □

Proof of Theorem 5.2:

Let $\Delta = (D, W)$ be a default theory with $\mathcal{F}(\Delta) = D$. Suppose $(D, W) \mid\sim x$. Further assume for each default $A : B/C \in D$ where $W \not\models A$, $A : B/C \in G(E, H)$ for any J-extension $\langle E, H \rangle$ of (D, W). Then, for every J-extension E of (D, W) there exists a J-extension E' of $\Delta' = (D, W \cup \{x\})$ such that $Cn(E) = Cn(E')$, and vice versa.

We first prove a utility lemma.

Lemma 7.1 *Let* $\Delta = (D, W)$ *and* $\Delta' = (D, W \cup \{x\})$ *be two default theories. Assume* $\mathcal{F}(\Delta) = D$. *Then* $\mathcal{F}(\Delta') = D$ *only if* $(D, W) \mid\sim x$.

Proof: Assume $(D, W) \mid\sim x$. That $\mathcal{F}(\Delta') \subseteq D$ is immediate. We show $D \subseteq \mathcal{F}(\Delta')$, i.e., assume for any $d \in D$, and show $d \in \mathcal{F}(\Delta')$. From the assumption $\mathcal{F}(\Delta) = D$, we know d is a generating default of some J-extension E of Δ. By Theorem 3.1, there exists a J-extension E' such that $Cn(E) = Cn(E')$. Clearly, d is a generating default of E' and hence is in $\mathcal{F}(\Delta')$. □

Proof of Theorem 5.2:

The if part follows directly from Theorem 7.1.

For the only-if part, we need to show that for any J-extension E' of $(D, W \cup \{x\})$, there exists a J-extension E of (D, W) with $Cn(E) = Cn(E')$.

If $W \models x$, then the claim holds trivially. Consider $W \not\models x$.

That E' is a J-extension of $(D, W \cup \{x\})$ implies, by the definition of J-extension, there exist one or more sequences $\{E'_n\}$ that lead to E'.

First, since E' is a J-extension of $(D, W \cup \{x\})$, elements in each sequence $\{E'_n\}$ that leads to E' can be partitioned into two parts:

$E'_0 = E_0 \cup \{x\}$ where $E_0 = W$;
$E'_{i+1} = E_{i+1} \cup \{x\}$
where $E_{i+1} = E_i \cup \{C_i\}$,
for $(A_i : B_i/C_i) \in D$ used in the ith step.

Let us also denote the corresponding sets of assumptions as:

$$H'_0 = H_0 = \emptyset$$
$$H'_{i+1} = H_{i+1} = H_i \cup \{B_i\}.$$

If we can show that there exists one such sequence $\{E'_n\}$ that leads to E', such that at each step k using default $A_k : B_k/C_k \in D$,

$$E'_k \models A_k \text{ if and only if } E_k \models A_k,$$

we then can conclude, $E = E' - \{x\}$, where E is the J-extension of (D, W) extended from E_i. The conclusion $Cn(E) = Cn(E')$ then directly follows from the assumption that $(D, W) \mid\sim x$. Thus, all we need to show is the above claim.

We now show this claim by contradiction.

Assume the claim does not hold. That is, for any sequence $\{E'_n\}$ that leads to E', the above property is not satisfied. This implies, for each such $\{E'_n\}$, there exists E'_i for some $i \geq 0$, such that either

$$E'_i \not\models A_i \text{ and } E_i \models A_i,$$

or

$$E'_i \models A_i \text{ and } E_i \not\models A_i.$$

The first case is obviously not possible since $E'_i \models E_i$ for each $i \geq 0$. We therefore only need to show a contradiction for the second case.

The fact that $E'_i \models A_i$ and $E_i \not\models A_i$ for *any* sequence $\{E'_n\}$ that leads to E' implies there exists at least one such sequence $\{E'_n\}$ such that (i) $E_i \not\models x$, and (ii) there is no step j, $j > i$, such that $A_j : B_j/C_j \in D$ is applicable at the ith step with $E_i \models A_j$. (i.e. there must exist one such sequence $\{E'_n\}$ in which the application of default $A_i : B_i/C_i$ cannot be delayed in $\{E'_n\}$). Note that all prerequisite-free defaults are applied before ith step.

Let us define a subset D_α of defaults as:

$$D_\alpha = \{A : B/C \mid A : B/C \in D \text{ and } W \not\models A\}.$$

From Lemma 7.1, we know that $\mathcal{F}(\Delta) = \mathcal{F}(\Delta') = D$. That is, the defaults that are used to construct $\{E'_n\}$ are also available for the construction of $\{E_n\}$. Thus, there exists a J-extension $\langle E, H \rangle$ of (D, W) which is extended from E_i and, by the assumption that the defaults in D_α

are all generating defaults of any J-extension, we have $D_\alpha \subseteq G(E,H)$. Since $E_i \not\models A_i$, there exists a subset D' of defaults in D that can be used to extend E_i to yield $E_i \cup N$ such that $E_i \cup N \models A_i$. Since $D_\alpha \subseteq G(E,H)$, the defaults in D' must be all prerequisite-free. Then, the set $E_i \cup N \cup Cons(D_\alpha)$ is a J-extension of (D,W). This implies the defaults in D', which are not part of generating defaults of E', can be used to properly extend E'. Hence, E' is not a J-extension of $(D, W \cup \{x\})$. Contradiction. □

References

[Brewka, 1991] G. Brewka. Cumulative default logic: in defense of nonmonotonic inference rules. *Artificial Intelligence*, 50:183–205, 1991.

[Delgrande and Jackson, 1991] J. Delgrande and W. Jackson. Default logic revisited. In *Proc. 2nd international Conference on Principles of Knowledge Representation and Reasoning*, pages 118–127, 1991.

[Dix, 1991] J. Dix. Classifying semantics of logic programs. In *Proc. the Workshop on Nonmonotonic Reasoning and Logic Programming*, pages 167–180, 1991.

[Dix, 1992] J. Dix. A framework for representing and characterizing semantics of logic programs. In *Proc. KR '92*, pages 591–602, 1992.

[Dix, 1994] J. Dix. A classification theory of semantics of normal logic programs. *Fundamenta Informaticae (to appear)*, 1994.

[Ghose *et al.*, 1993] A. Ghose, P. Hadjinian, A. Sattar, J. You, and R. Goebel. Iterated belief change: A preliminary report. In *Proc. Australian Joint Conference on Artificial Intelligence*, 1993.

[Li and You, 1991] L. Li and J. You. Making default inferences from logic programs. *Computational Intelligence*, 7(3):142–153, 1991.

[Lukaszewicz, 1987] W. Lukaszewicz. Two results on default logic. *Computers and Artificial Intelligence*, 6:329–343, 1987.

[Makinson, 1988] D. Makinson. General theory of cumulative inference. In M. Reinfrank, J. de Kleer, M.L. Ginsberg, and E. Sandewall, editors, *Non-Monotonic Reasoning, Lecture Notes in Artificial Intelligence 346*, pages 1–18. Springer-Verlag, 1988.

[Poole, 1988] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[Poole, 1991] D. Poole. The effect of knowledge on belief: conditioning, specificity and the lottery paradox in default reasoning. *Artificial Intelligence*, pages 281–308, 1991.

[Reiter, 1980] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Sattar and Goebel, 1991] A. Sattar and R. Goebel. Using crucial literals to select better theories. *Computational Intelligence*, 7(1):11–22, 1991.

A Clausal Form Translation for Propositional Modal Logic *

Christophe MATHIEU
Laboratoire d'Informatique de Marseille URA CNRS 1787
Université de Provence Case A
3 place Victor Hugo
13331 Marseille Cedex 3
✆: (33) 91 10 61 28 Fax: (33) 91 10 61 02
E-Mail: mathieu@gyptis.univ-mrs.fr

Abstract.
In this paper we present a clausal form translation for modal logic. In order to define a theorem prover based on the resolution principle, we need modal formula to be in clausal form. But an important problem with modal logic is that there is no such simple clausal form as in classical logic. So we propose an original clausal transformation for modal formulas. This new translation preserves the modal structure of formulas and also avoids the exponential increase in size which may occur with another translation. Moreover, initial modal formulas and their translation are equivalent on the language of the initial formula. This technique should increase the attractiveness of modal resolution theorem provers for automated reasoning.

1 Introduction

Modal Logic is now very popular in Computer Science. It's used in the Logic of Programs such as temporal logic or dynamic logic, in Artificial Intelligence i.e. Knowledge Representation, Natural Language, ... The importance of modal logic motivates the development of automated proof methods.

Thus, several methods such as Tableau methods [Fitting, 1988], Matrix methods [Wallen, 1987], Resolution methods [Abadi and Manna, 1990], ... have been developed. Many automated theorem provers are based on resolution (a standard in theorem proving) [Robinson, 1965], they deal with a given knowledge represented by a set of clauses that confers them a great simplicity and allows optimizations. But there is a reason which makes the definition of resolution methods for modal logic difficult: there is a lack of normal clausal form in modal logic. This lack is due to this fact: in normal modal logic $\Box(l_1 \vee l_2 \vee \ldots \vee l_n)$ and $\Box l_1 \vee \Box l_2 \vee \ldots \vee \Box l_n$, $\Diamond (l_1 \wedge l_2 \wedge \ldots \wedge l_n)$ and $\Diamond l_1 \wedge \Diamond l_2 \wedge \ldots \wedge \Diamond l_n$ are not equivalent. So it's impossible to exhibit the same clausal form translation as classical logic one's. Then, among several automated deduction methods for modal logic given in the literature, some are based on translation into other formalisms such as classical logic or deterministic logic that possess normal form theorem [Fariñas and Herzig, 1991]. Even the best adaptation of propositional normal clausal form translation defined in [Fariñas and Herzig, 1991] can't solve the fact that the and connective can't cross over the possible modality.

To solve these problems, we present a clausal form translation for modal logic that converts any propositional modal formula f in a set C of what we called pseudo-clauses, featuring a linear ratio of lengths. The obtained pseudo-literals don't contain any conjunction. A resolution method based on these pseudo-clauses should be easily defined.

* This work has been partially supported by the ESPRIT BRA (Basic Research Action) project DRUMS II (Defeasible Reasoning and Uncertainty Management Systems) Action 6156 of the European community and the french project PRC-GDR CNRS Gestion de l'évolutif et de l'incertain dans les bases de connaissances.

The basic idea of the translation, due to [Tseitin, 1983] and [Plaisted and Greenbaum, 1986] (see also [Siegel, 1987]), is to introduce new proposition P to refer to various sub-formulas A of the original formula. Then the assertion $P \rightarrow A$ may be added to the set of formulas (this technique is used in propositional calculus to cut a clause in two clauses of inferior length). This idea can be extended to modal logic: if the formula is $\Box f$ (or $\Diamond f$), we add $\Box P \wedge \Box(P \rightarrow f)$ (or $\Diamond P \wedge \Box(P \rightarrow f)$) to the set of formulas where P is a new proposition, and $\Box$ ($\Diamond$) is the necessity (possible) operator of some modal logic.

The translation enables us to transform modal propositional formulas into pseudo-clauses without exponential increase of the size of formulas. Its interest is, first, that the translation should happen linearly (the ratio between the initial and final formula's length is linear), as the classical algorithm under conjunctive normal form is theoretically exponential. The typical propositional example is $(p_{11} \vee p_{12} \vee \ldots \vee p_{1n}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee \ldots \vee p_{mn})$ where the associated clausal form is a formula with n^m clauses of m literals. With our method, we obtain m+1 clauses of n+1 literals. Secondly, as the translation uses new propositions, the initial formula and the set of modal pseudo-clauses C are equivalent on the language which contains only the propositions occurring in the initial formula. Particularly, the translation preserves the satisfiability or the unsatisfiability of the initial formula. And lastly the translation preserves the modal structure of the formulas: the obtained pseudo-clauses are modal formulas.

In the first part, we introduce modal logic. In the second we present the modal pseudo-clausal translation.

2 Modal Logic

2.1 Definitions

A logic of any kind needs a language. Although we consider different logics here, the syntax for all of them is essentially the same. Given a set of propositions P labelled p, p', q, q',... and the modal operators $\Box$ and $\Diamond$, the ***language*** $\mathfrak{L}$ is a set of formulas defined recursively by:

- every proposition of P is a formula
- if f and g are formulas, $f \vee g$, $f \wedge g$ and $\neg f$ are formulas
- if f is a formula, $\Box f$ and $\Diamond f$ are formulas

We use the standard abbreviations $f \rightarrow g$ for $\neg f \vee g$, $\Diamond f$ for $\neg \Box \neg f$.

The ***size of a formula f*** in $\mathfrak{L}$ is its length over the alphabet $P \cup \{\Box, \Diamond\}$.

The ***modal degree of a formula f*** (deg(f)) is the number of nesting modal operators in f. More formally, deg(p)=0 for a proposition $p \in P$, $deg(\neg f)=deg(f)$, $deg(f \rightarrow g)= \max(deg(f),deg(g))$ and $deg(\{\}f)=deg(f)+1$ where $\{\}$ is $\Box$ or $\Diamond$. For example, the modal degree of the formula $\Box(p_1 \rightarrow \Diamond(p_1 \rightarrow p_2) \rightarrow \Box\Box(p_3 \rightarrow p_4))$ is 3.

We also define a ***sub-formula of a formula f*** by induction on the structure of f: g is a sub-formula of f if either f=g, or f is of the form $\neg f'$ (respectively, $f' \rightarrow f''$, $\Box f'$) and g is a sub-formula of f' (resp. g is either a sub-formula of f' or of f'', g is a sub-formula of f'). For example, $\Diamond(p_1 \rightarrow p_2)$ is a sub-formula of $\Box(p_1 \rightarrow \Diamond(p_1 \rightarrow p_2) \rightarrow \Box\Box(p_3 \rightarrow p_4))$. In order to simplify the notation, we write F=uGv to mean that G is a sub-formula of F.

Let $\mathbb{1}$ be the empty modal operator defined by $\mathbb{1}(f)=f$. The **power of n of the necessity modal operator** $\Box$ *(written $\Box^n$) is defined for all $n \geq 0$ by:*

- $\Box^0 = \mathbb{1}$
- $\Box^{n+1} = \Box\, \Box^n \quad \forall\, n \geq 0$.

2.2 Axiom systems

The ***normal system K*** consists of axioms and inference rules:

Axioms

- all instances of tautologies of propositional calculus.
- distribution axiom $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$.

Inference rules

- $\dfrac{\vdash A}{\vdash \Box A}$ (generalization)
- $\dfrac{A\ ,\ A \rightarrow B}{B}$ (Modus Ponens)

If we add axioms to the minimal normal system K, we can obtain different modal systems such as T adding $\Box f \rightarrow f$, S4 adding $\Box f \rightarrow f$ and $\Box f \rightarrow \Box\Box f$, and S5 adding $\Diamond p \rightarrow \Box \Diamond p$ to S4. We now only consider normal systems.

2.3 Possible worlds semantics

A ***Kripke model*** is a tuple (W,R,V), where W is a set of state or possible-worlds, R a binary relation on the states of W (w is in relation with w' is written wRw') and V is a truth assignment to the propositions of P (i.e. V : WxP$\mapsto$ {**true**,**false**}). (W,R) is a ***Kripke structure***.

We define an inductive "satisfaction" relation $\models$ between a model and a formula.

$(W,R,V) \models_w g$ means that the model (W,R,V) satisfies the formula g in the world w.

- $(W,R,V) \models_w p$ iff $V(w,p)$=**true**, $p \in P$
- $(W,R,V) \models_w \neg g$ iff $(W,R,V) \not\models_w g$
- $(W,R,V) \models_w (g \rightarrow h)$ iff: if $(W,R,V) \models_w g$ then $(W,R,V) \models_w h$
- $(W,R,V) \models_w \Box g$ iff $(W,R,V) \models_{w'} g$ for all w' satisfying wRw'
- $(W,R,V) \models_w \Diamond g$ iff if there is some w' satisfying wRw' and $(W,R,V) \models_{w'} g$

We say that a formula g is ***valid in a model*** (W,R,V) or ***satisfiable*** and write $(W,R,V) \models g$ iff $(W,R,V) \models_w g$ for all $w \in W$; we say that a formula g is ***valid in a structure*** (W,R) and write $(W,R) \models g$ iff g is valid in all models (W,R,V); we say that a formula g is ***valid*** and write $\models g$ iff g is valid in all the structures (W,R).

3 Pseudo-Clausal Form Translation

To simplify the translation, we first put formulas into negative normal form; this can be done linearly and retaining the logical equivalence.

3.1 Negative Normal Form

A formula is in ***negative normal form*** if the $\neg$ connective occurs only in the literals and the $\rightarrow$ connective does not appear.

Theorem: *Any modal formula is equivalent to a formula in negative normal form with the same size.*

Proof : The translation into negative normal form is classical. We just mention the modal cases: $\neg \Box f$ is translated into $\Diamond \neg f$ and $\neg \Diamond f$ is translated into $\Box \neg f$. The entire proof can be found in [Mathieu, 1993a]. ■

The formulas are now in negative normal form.

3.2 Clausal Form Translation in propositional calculus

The idea of the translation is to introduce new proposition P to replace subformula g of the original formula, and to add the new clause $P \rightarrow g$. We can see it on an example:

Let $(l_{11} \wedge l_{12} \wedge \ldots \wedge l_{1n}) \vee (l_{21} \wedge l_{22} \wedge \ldots \wedge l_{2n}) \vee \ldots \vee (l_{m1} \wedge l_{m2} \wedge \ldots \wedge l_{mn})$ be the initial formula.

We obtain $P_1 \vee (l_{21} \wedge l_{22} \wedge \ldots \wedge l_{2n}) \vee \ldots \vee (l_{m1} \wedge l_{m2} \wedge \ldots \wedge l_{mn}) \wedge (P_1 \rightarrow l_{11}) \wedge \ldots \wedge (P_1 \rightarrow l_{1n})$ by replacing the subformula $(l_{11} \wedge l_{12} \wedge \ldots \wedge l_{1n})$ by P_1 and adding $P_1 \rightarrow (l_{11} \wedge l_{12} \wedge \ldots \wedge l_{1n})$.

We obtain finally n+m+1 clauses :

$P_1 \vee P_2 \vee \ldots \vee P_m$

$\neg P_1 \vee l_{11} \quad \neg P_1 \vee l_{12} \quad \ldots \quad \neg P_1 \vee l_{1n}$

$\ldots$

$\neg P_m \vee l_{m1} \quad \neg P_m \vee l_{m2} \quad \ldots \quad \neg P_m \vee l_{mn}$

3.3 Clausal Form Translation in modal logic

3.3.1 Two examples

The translation is extended to the modal case by retaining the "modal context" of a formula. This can be seen on two examples that show a necessary and a possible translation example:

First example:

Let $\Box[(l_{11} \wedge l_{12} \wedge \ldots \wedge l_{1n}) \vee \ldots \vee (l_{m1} \wedge l_{m2} \wedge \ldots \wedge l_{mn})]$ be the initial formula.

We obtain $\Box[P_1 \vee (l_{21} \wedge l_{22} \wedge \ldots \wedge l_{2n}) \vee \ldots \vee (l_{m1} \wedge l_{m2} \wedge \ldots \wedge l_{mn})] \wedge \Box(P_1 \rightarrow l_{11}) \wedge \ldots \wedge \Box(P_1 \rightarrow l_{1n})$ by replacing the subformula $(l_{11} \wedge l_{12} \wedge \ldots \wedge l_{1n})$ by P_1 and adding $\Box[P_1 \rightarrow (l_{11} \wedge l_{12} \wedge \ldots \wedge l_{1n})]$ distributed in $\Box(P_1 \rightarrow l_{11}) \wedge \ldots \wedge \Box(P_1 \rightarrow l_{1n})$.

We obtain finally n+m+1 "clauses":

$\Box[P_1 \vee P_2 \vee \ldots \vee P_m]$

$\Box(\neg P_1 \vee l_{11}) \quad \Box(\neg P_1 \vee l_{12}) \quad \ldots \quad \Box(\neg P_1 \vee l_{1n})$

$\ldots$

$\Box(\neg P_m \vee l_{m1}) \quad \Box(\neg P_m \vee l_{m2}) \quad \ldots \quad \Box(\neg P_m \vee l_{mn})$

Second example:

Let $\Diamond [(A \wedge B) \vee C]$ be the initial formula.

We obtain $\Diamond[P \vee C] \wedge \Box(P \rightarrow A) \wedge \Box(P \rightarrow B)$ by replacing the subformula $A \wedge B$ by P and adding $\Box[P \rightarrow (A \wedge B)]$ distributed in $\Box(P \rightarrow A) \wedge \Box(P \rightarrow B)$.

We obtain finally four "clauses":

$\Diamond P \quad \Diamond C \quad \Box(\neg P \vee A) \quad \Box(\neg P \vee B)$

As we can see on the two examples, we introduce two main translations:

$\Box f$ is translated into $\Box P \wedge \Box (P \rightarrow f)$ and $\Diamond f$ is translated into $\Diamond P \wedge \Box (P \rightarrow f)$.

3.3.2 Definitions

As the results of the translation in propositional calculus are clauses, the modal

logic results are what we call pseudo-clauses. These pseudo-clauses are composed of pseudo-literals defined by:

A ***pseudo-literal*** is:

- a classical literal[1]
- $\Diamond(l_i)$ where l_i is a classical literal
- $\Box(l_1 \vee l_2 \vee \ldots \vee l_m)$ where each l_i is a **pseudo-literal.**

Example: $\Box(B \vee C \vee F)$ or $\Diamond B$ are pseudo-literals, but not $\Box(A \wedge X)$ and $\Diamond(B \vee F)$.

A ***pseudo-clause*** is a disjunct $pl_1 \vee \ldots \vee pl_m$, where each pl_i is a pseudo-literal.

Example: $\Box(\Box(A \vee \Diamond X) \vee C \vee \Diamond F)$ is pseudo-clause.

A formula is in **pseudo-clausal form** if it's a conjunct of pseudo-clauses. As usual we identify a conjunct of pseudo-clauses with the sequence $c_1 c_2 \ldots c_n$.

Let $\mathcal{L}$ be a propositional modal language and $\mathcal{P}$ a set of propositions (non modal) that don't belong to $\mathcal{L}$. $\mathcal{L}_{\mathcal{P}}$ is the language obtained by adding $\mathcal{P}$ to $\mathcal{L}$.

A formula f of $\mathcal{L}$ and a formula h of $\mathcal{L}_{\mathcal{P}}$ are ***$\mathcal{L}$-equivalents*** ($\Leftrightarrow_{\mathcal{L}}$) iff for each formula k of $\mathcal{L}$: $\models f \rightarrow k$ if and only if $\models h \rightarrow k$ ($\models f$ means f is valid in all Kripke structures).

3.4 Translation Theorem

Let F = u G v be a modal formula in negative normal form of a modal language $\mathcal{L}$, u and v are sequences of symbols of $\mathcal{L}$ (by definition G is a subformula of F). Let P be a proposition of $\mathcal{P}$ not belonging to $\mathcal{L}$.

"one step" Theorem

Let F = u G v and set ${}^tF = u\ P\ v \wedge \Box^n(P \rightarrow G)$ where n is the number of modal operators governing G then F and ${}^tF$ are $\mathcal{L}$-equivalents.

Proof Schema: The complete proof can be found in [Mathieu, 1993a]. The first step is to say that if a formula is valid, then there exists a model such that the formula is satisfiable. We show by induction of the structure of the formula that the following $\mathcal{L}$-equivalencies are true:

$f \wedge g$	$\Leftrightarrow_{\mathcal{L}}$	$(f \wedge p) \wedge (p \rightarrow g)$
$f \vee g$	$\Leftrightarrow_{\mathcal{L}}$	$(f \vee p) \wedge (p \rightarrow g)$
$\Box(f \Delta g)$	$\Leftrightarrow_{\mathcal{L}}$	$\Box(p \Delta g) \wedge \Box(p \rightarrow f)$ where Δ is $\vee$ or $\wedge$
$\Diamond(f \Delta g)$	$\Leftrightarrow_{\mathcal{L}}$	$\Diamond(p \Delta g) \wedge \Box(p \rightarrow f)$ where Δ is $\vee$ or $\wedge$
$\Box(f \wedge g)$	$\Leftrightarrow_{\mathcal{L}}$	$\Box(f) \wedge \Box(g)$
$\Diamond(f \vee g)$	$\Leftrightarrow_{\mathcal{L}}$	$\Diamond(f) \vee \Diamond(g)$

(The two last equivalencies are classical modal equivalencies. They are added to simplify modal formulas and to guarantee modal formùulas to be in a minimal form)

The reasoning is first to say that if a formula F is valid in all structure then F is valid in one model for a world w_0.

If $\models F \rightarrow k$ then $\models {}^tF \rightarrow k$: It's easy to check that if we have a model of ${}^tF$ then it's a model of F, and then of k (for example if this model satisfies $(f \vee p) \wedge (p \rightarrow g)$ then it satisfies f (then it satisfies $f \vee g$) or it satisfies p then it satisfies g(then it satisfies $f \vee g$).

If $\models {}^tF \rightarrow k$ then $\models F \rightarrow k$: If we have a model of F then it's possible to extend this model to the new language $\mathcal{L}_{\mathcal{P}}$ (the language of F adding the new propositions) such that this extended model is a model of ${}^tF$ and then of k. This extension depends on the structure of the formula F: if F is $a \wedge b$ or $a \vee b$, the extended model is chosen to satisfy the proposition p in the world w_0 and false elsewhere. if F is $\Box g$, the extended model is chosen to satisfy the proposition p in all the world in relation with w_0 and false elsewhere. if F is $\Diamond g$, the extended model is chosen to satisfy the proposition p in all the world in relation with w_0 where g is satisfies and false elsewhere. ■

Translation Theorem

Any modal formula F can be translated into a set of pseudo-clauses ${}^tF$ such that F and ${}^tF$ are $\mathcal{L}$-equivalents.

Proof Schema: By induction on the "one step" theorem. ■

4 Detailed Pseudo-Clausal Translation

Let $p_1, p_2, \ldots, p_m$ be pseudo-literals and let P be a new proposition that doesn't belong to $\mathcal{L}$. We are going to detail the pseudo-clausal form translation.

If the formula F is a propositional one, the translation is:

$u \vee (p_1 \wedge p_2 \wedge \ldots \wedge p_m) \vee v \quad \Leftrightarrow_{\mathcal{L}}$

$u \vee P \vee v \wedge (\neg P \vee p_1) \wedge \ldots \wedge (\neg P \vee p_m)$

$u \wedge (p_1 \vee p_2 \vee \ldots \vee p_m) \wedge v \quad \Leftrightarrow_{\mathcal{L}}$

$u \wedge P \wedge v \wedge (\neg P \vee p_1 \vee p_2 \vee \ldots \vee p_m)$

If F is a modal formula, let u and v be sequences of symbols, u containing n-1 modal operators, and let u' and v' be sequences of symbols that don't contain any modal operator. The translation is:

(1) $u \Box(u' \wedge p_1 \vee p_2 \vee \ldots \vee p_m \wedge v') v \Leftrightarrow_{\mathcal{L}}$

$u \Box(u' \wedge P \wedge v') v \wedge \Box^n(\neg P \vee p_1 \vee p_2 \vee \ldots \vee p_m)$

[1] A classical literal is a non modal literal, i.e. a literal of propositional calculus.

(2) $u\,\Box(u' \vee p_1 \wedge p_2 \wedge \ldots \wedge p_m \vee v')\,v \Leftrightarrow$

$u\Box(u' \vee P \vee v')v \wedge \Box^n(\neg P \vee p_1) \wedge \ldots \wedge \Box^n(\neg P \vee p_m)$

(3) $u\,\Diamond(u' \wedge p_1 \vee p_2 \vee \ldots \vee p_m \wedge v')\,v \Leftrightarrow$

$u\,\Diamond(u' \wedge P \wedge v')\,v \wedge \Box^n(\neg P \vee p_1 \vee \ldots \vee p_m)$

(4) $u\,\Diamond(u' \vee p_1 \wedge p_2 \wedge \ldots \wedge p_m \vee v')\,v \Leftrightarrow$

$u\Diamond(u' \vee P \vee v')v \wedge \Box^n(\neg P \vee p_1) \wedge \ldots \wedge \Box^n(\neg P \vee p_m)$

(5) $\Box(p_1 \wedge p_2 \wedge \ldots \wedge p_m) \Leftrightarrow$

$\Box(p_1) \wedge \Box(p_2) \wedge \ldots \wedge \Box(p_m)$

(6) $\Diamond(p_1 \vee p_2 \vee \ldots \vee p \Leftrightarrow$

$\Diamond(p_1) \vee \Diamond(p_2) \vee \ldots \vee \Diamond(p_m)$

Example: The formula $\Box[(A \vee \Diamond(B \wedge C)) \wedge \Box D]$ is translated into the following set of pseudo-clauses:

$\Box P_2 \quad \Box\Box D \quad \Box\Box (P_1 \rightarrow B)$

$\Box\Box (P_1 \rightarrow C) \quad \Box(P_2 \rightarrow A \vee \Diamond P_1)$

with the following translations:

- $\Box[(A \vee \Diamond(\mathbf{B} \wedge \mathbf{C})) \wedge \Box D]$

  Using (4), replacing $B \wedge C$ by P_1 and adding $\Box\Box (P_1 \rightarrow B) \wedge \Box\Box (P_1 \rightarrow C)$ as $B \wedge C$ is governing by $\Box\Diamond$.
- $\Box[(\mathbf{A} \vee \Diamond \mathbf{P_1}) \wedge \Box D] \wedge \Box\Box (P_1 \rightarrow B) \wedge \Box\Box (P_1 \rightarrow C)$

  Using (1), replacing $A \vee \Diamond P_1$ by P_2 and adding $\Box (P_2 \rightarrow A \vee \Diamond P_1)$ as $A \vee \Diamond P_1$ is governing by $\Box$.
- $\Box[P_2 \wedge \Box D] \wedge \Box\Box (P_1 \rightarrow B) \wedge \Box\Box (P_1 \rightarrow C) \wedge \Box (P_2 \rightarrow A \vee \Diamond P_1)$

  Using (5), we obtain $\Box P_2 \wedge \Box\Box D$ from $\Box[P_2 \wedge \Box D]$.

5 A Linear Pseudo-Clausal Translation

A recursive and anarchic use of the theorem will not allow the achievement of a set of modal pseudo-clause without exponential increase of the ratio of the lengths. To make this ratio linear, we must replace only the elementary sub-formulas (the non splittable formulas) of the initial formula.

A ***non splittable formula*** is a conjunction or a disjunct of pseudo-literals that are not sub-formulas of a pseudo-literal.

Example: In the formula $\Box((A \wedge B) \vee (C \wedge D) \vee (E \wedge F))$ there are three non splittable formulas: $(A \wedge B)$, $(C \wedge D)$ and $(E \wedge F)$.

Property: *Any propositional modal formula f can be translated into a set of modal pseudo-clauses C such that the ratio of the lengths of f and C is at most 2n+3 where n is the modal degree of f.*

Proof: If we decrease the size of the formula by one unit (this difference corresponds to the translation of the formula $\Box(a_1 \wedge a_2)$ into $\Box P \wedge \Box(P \rightarrow a_1) \wedge \Box(P \rightarrow a_2)$), we must add at most two clauses of a total size 2(n+2). After the translation, the remaining formula has a size at most equal to L-1, if L is the size of the initial formula. We obtain a ratio of size at most equal to 2n+3. ■

6 Particular case

The pseudo-clausal form translation is independent of the modal system considered. Therefore the translation can be optimized if we consider a particular modal system.

In the modal system S5, every modalities (list of modal operator) can be rewritten in one of the following: $\mathbb{1}$, $\neg$, $\Box$, $\Diamond$, $\neg\Box$, $\neg\Diamond$ (see [Audureau et al., 1989] for example). In this case, the modal degree of the formula is equal to one and then $\Box^n$ is equivalent to $\Box$.

A ***pseudo-literal*** is:

- a classical literal
- $\Diamond(l_i)$ where l_i is a classical literal
- $\Box(l_1 \vee l_2 \vee \ldots \vee l_m)$ where each l_i is a **classical literal.**

The detailed rules are the following

$u\,\Box(u' \wedge p_1 \vee p_2 \vee \ldots \vee p_n \wedge v')\,v \Leftrightarrow$

$u\,\Box(u' \wedge P \wedge v')\,v \wedge \Box(\neg P \vee p_1 \vee p_2 \vee \ldots \vee p_n)$

$u\,\Box(u' \vee p_1 \wedge p_2 \wedge \ldots \wedge p_n \vee v')\,v \Leftrightarrow$

$u\Box(u' \vee P \vee v')v \wedge \Box(\neg P \vee p_1) \wedge \ldots \wedge \Box(\neg P \vee p_n)$

$u\,\Diamond(u' \wedge p_1 \vee p_2 \vee \ldots \vee p_n \wedge v')\,v \Leftrightarrow$

$u\,\Diamond(u' \wedge P \wedge v')\,v \wedge \Box(\neg P \vee p_1 \vee \ldots \vee p_n)$

$u\,\Diamond(u' \vee p_1 \wedge p_2 \wedge \ldots \wedge p_n \vee v')\,v \Leftrightarrow$

$u\Diamond(u' \vee P \vee v')v \wedge \Box(\neg P \vee p_1) \wedge \ldots \wedge \Box(\neg P \vee p_n)$

and u doesn't contain any modal operator

7 Conclusion

We present a clausal form translation for modal logic. This method is based on the introduction of new propositions. This translation allows to obtain from a propositional modal formula f, a set of modal clauses whose length depends only linearly on the length of f. The set of clauses is

$\mathfrak{L}$-equivalent with the initial formula i.e. equivalent on the language $\mathfrak{L}$. This result is more general in the sense that it can be used not only to check the satisfiability of a set of formulas but also to produce formula with a consequence finding algorithm. Moreover, the structure of the modal formulas is retained. It appears that this translation should significantly increase the use of resolution based theorem provers for modal logic.

The clausal form translation can be extended to multimodal logic. The complete translation can be found in [Mathieu, 1993a]. It's also possible to define a resolution theorem prover using the pseudo-clausal form. This new modal clausal form simplifies this kind of proof procedure because there is no and connective inside modal formulas. Resolution rules are defined between clauses. An example can be found in [Mathieu, 1993b].

References

[Abadi and Manna, 1990] M. Abadi et Z. Manna. Nonclausal Deduction in First Order Temporal Logic, *Journal of ACM*, Vol 37, N° 2, p. 279-317

[Audureau et al., 1989] E. Audureau, P. Enjalbert et L. Farinas del Cerro, *Logique temporelle sémantique et validation de programmes parallèles*, ERI MASSON

[Fariñas and Herzig, 1991] L. Fariñas Del Cerro, A. Herzig: Modal Deduction with Applications in Epistemic and Temporal Logics. 5th draft, to appear in: *Handbook of Logic in Artificial Intelligence* edited by Dov Gabbay and Chris Hogger, Oxford University Press

[Fitting, 1988] M. Fitting, First Order Modal Tableaux, *Journal of automated Reasoning*, 4, p. 191-213

[Mathieu, 1993a] C. Mathieu: *Une procédure de preuve pour une logique modale non monotone*, Thèse de doctorat informatique de l'université de Provence, Marseille

[Mathieu, 1993b] C. Mathieu, A Resolution Method for a Non Monotonic Multimodal Logic, *Proc. of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, ECSQARU'93. LNCS 747

[Plaisted and Greenbaum, 1986] D. A. Plaisted and S. Greenbaum: A Structure-Preserving Clause Form Translation, *Journal of Symbolic Computation*, 2, pp 293-304

[Robinson, 1965] J. A. Robinson: A Machine Oriented Logic Based on the Resolution Principle, JACM, Vol 12 n°1, pp 23-41

[Siegel, 1987] P. Siegel: *Représentation et utilisation de la connaissance en calcul propositionnel*, Thèse de Doctorat d'état en Informatique, Université d'Aix-Marseille II

[Tseitin, 1983] G.S. Tseitin: On the Complexity of Derivations in *Propositional Automation Of Reasoning 2: Classical Papers* On Computational Logic pp 466-483 Siekmann Wrightson eds (1983)

[Wallen, 1987] L.A. Wallen: Matrix Proof Methods for Modal Logic, *Proc. of the International Joint Conference on Artificial Intelligence*, Milano

A Non-Horn ATMS Which Allows Flexible Specification of Required Completeness*

Bruce Spencer
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick
Canada E3B 5A3
bspencer@unb.ca

Robin Cohen
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1
rcohen@watdragon.uwaterloo.ca

Abstract

In this paper, we present an ATMS that can reason with non-Horn clauses. The ATMS is significantly different from de Kleer's extended ATMS. It incorporates a variant of Loveland's MESON proof procedure and a mechanism for reducing redundant proofs, to improve efficiency. The ATMS is designed to allow users to specify whether one, some or all proofs are required. This, in turn, allows further reduction in computation. The design of the ATMS is described in detail, illustrating how reductions in proofs can be achieved. The usefulness of the ATMS is illustrated with examples from plan recognition and circuit diagnosis and through comparisons to de Kleer's extended ATMS and Stickel's PTTP.

1 Introduction

Current assumption-based truth maintenance systems (ATMS) [de Kleer 86a] used for automated reasoning are not sufficiently flexible. Users may want to reason with problems that can only be represented with disjunctions and so require non-Horn clauses as input to the ATMS. Further, with the introduction of disjunctions, the number of proofs to compute may be prohibitively expensive. But allowing users to specify whether all environments need to be computed or not allows a necessary reduction in work for long problems; it permits a complete result for problems that are not too time consuming.

In this paper, we present a system that allows users to employ an ATMS for a wider range of problems and to have flexibility in deciding how the ATMS is used. Our solution involves integrating the MESON proof procedure [Loveland 78] into an ATMS, but improving on that proof procedure to reduce the work. We have implemented the system and compared its performance to that of Stickel's Prolog technology theorem prover [Stickel 88] and de Kleer's extended ATMS [de Kleer 86b]. We demonstrate the usefulness of the method by discussing examples that benefit from the use of the system and comparisons to other systems. Our contribution, therefore, is to provide a tool for truth maintenance that is more efficient (in certain cases), more widely applicable (non-Horn) and flexible to allow users to sacrifice completeness for efficiency.

2 Motivation

The ATMS we designed was motivated by some problems in the area of plan recognition. Kautz's plan recognition system [Kautz 87] was significant in that it produced as output a list of all possible plans covered by the observations, rather than employing heuristics that may over-commit to a single interpretation. But the system operated with a predefined plan library; it was not designed to accommodate changes to the plan library in the face of new observations. In order to provide a system to do plan recognition with updates, we needed to incorporate a truth maintenance system, but now required one that could reason with the kinds of non-Horn clauses typically used to represent plan recognition. (Non-Horn clauses may arise to describe some uncertainty about the observation, as in "The agent is holding a key case or a wallet." A disjunction may also arise when drawing conclusions, "The agent is taking car keys either to lock the car door or to start the car.") Moreover, we needed to modify Kautz's plan recognition algorithms to accommodate truth maintenance, in order to manage changes to the library and the resulting candidate plans. To reduce redundancy in the possible proofs

* This work was supported by grants from NSERC.

(generated to display candidate plans), we developed a variant of the MESON proof procedure [Loveland 78] and incorporated it into the ATMS.

The solution for the plan recognition case is fairly detailed, including a specific description of Kautz's plan recognition system and the procedures required to manage both changes to the plan library and modifications to the proof procedures for generating candidate plans. In this paper, therefore, we will focus on a simpler example for illustration, as discussed below. (See [Spencer 90] and [Cohen and Spencer 93] for details on the plan recognition case.)

The application area of circuit diagnosis is one where it is useful to employ a truth maintenance system to reason from assumptions about whether gates are faulty or working to conclusions about the values of the circuits. If a set of conditions leads to a conclusion that is contrary to observations, then the assumptions underlying that conclusion are contradicted. This may allow us to conclude that other assumptions are correct. For these kinds of problems, it is often necessary to reason with non-Horn clauses – for example, it may simply be known that either a gate is tied high, tied low or working, leading to a disjunctive clause representation. The circuit diagnosis area is therefore one where a non-Horn ATMS is useful. In section 5 we comment further on these application areas, and the usefulness of our particular ATMS.

3 Background

In this section we discuss the assumption-based truth maintenance system (ATMS) of de Kleer's [de Kleer 86a], the MESON proof procedure of Loveland's [Loveland 78] and the ordered clause refinement of it [Spencer 91, Spencer 93]. Where propositional logic appears, we use $\vee$ for OR, ',' for AND, $\neg$ for NOT and $\leftarrow$ for IF.

3.1 The ATMS

This section illustrates the ATMS algorithms from a high level perspective, but for a more detailed discussion, we refer the reader to [de Kleer 86a].

The ATMS is a restricted default reasoning system [Reiter 80, Reiter 87, Kean and Tsiknis 93]. A default reasoning system computes explanations [Poole 87]. Given a formula J, where some literals of J are designated as assumptions (or normal defaults), an explanation for a literal L is a set E of assumptions such that $E \cup J \models L$, and $E \cup J$ is consistent. The ATMS restricts J to a set of propositional Horn clauses, and it computes and stores all minimal explanations for all positive literals in J. Each positive literal is stored in the ATMS with its label, a set of environments. An explanation is called an environment in ATMS terms.

The ATMS operates by building an and/or graph from each of the clauses in J. Each *or* node contains one positive literal and its label. There is exactly one *and* node for each clause in J. For the clause $a \leftarrow b_1,..., b_n$, directed arcs are created from the *or* node for bi to this *and* node, and one arc connects this *and* node to the *or* node containing a.

If the literal a in an *or* node is an assumption then it label is initially set to {{a}}.

For example, if the clauses are

Xis1 $\leftarrow$ A

Yis2 $\leftarrow$ B

Zis3 $\leftarrow$ Xis1, Yis2

where A and B are ATMS assumptions, then the ATMS generates the and/or graph in Figure 1. *Or* nodes are shown as literals; *and* nodes are shown as filled circles.

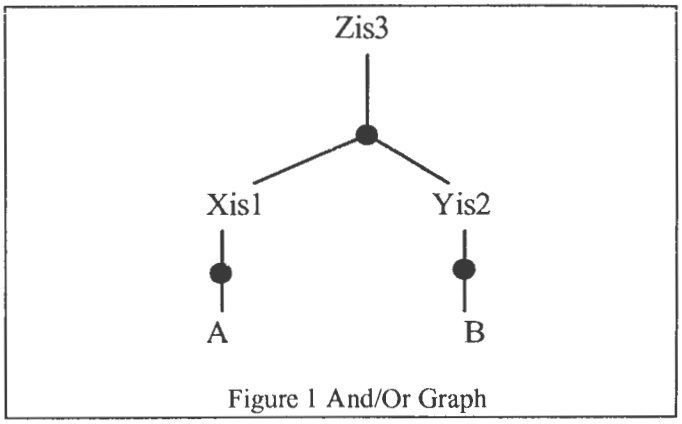


Figure 1 And/Or Graph

The ATMS algorithms propagate labels along the arcs in the graph. For instance the label for A is {{A}}, which is propagated to Xis1, so that its label becomes {{A}} also. Similarly Yis2 gets the label {{B}}. Theses labels are further propagated up the graph, but they are conjoined at the *and* node, creating {{A, B}} This new label is propagated to Zis3.

Propagation through the *and* nodes requires a conversion of a conjunction of labels, each of which is a disjunctive normal formula, to a single disjunctive normal formula. Non-minimal conjuncts are removed. For instance, if the label for Xis1 was {{A, C}, {D, E}} and the label for Yis2 was {{A, B}, {B, C, D}}, then the new label for Zis3 would be {{A, B, C}, {A, B, D, E}, {B, C, D, E}}. The label {A, B, C, D} is removed because {A, B, C} subsumes it.

There is always a distinguished node $\perp$ that represents false. Horn clauses that contain no positive literals have the form $\perp \leftarrow b_1,..., b_n$. Any environment propagated to $\perp$ is inconsistent. In ATMS terminology, it is called a *nogood*. Any superset of a nogood environment is removed from every label in the graph. So if {A, B, E} is a nogood, then the label for Zis3 will become {{A, B, C}, {B, C, D, E}}.

There are two operations that can be performed with an ATMS: making a literal into an assumption, and adding a new clause to J. To make a literal L into an assumption the ATMS creates an *or* node for L, if there is not already one,

and adds the environment {L} to L's label. Then propagation is invoked so that all consequences of L receive the effect of the new environment. To perform the operation of adding a $\leftarrow b_1,\ldots, b_n$, the ATMS creates a new *or* node for each atom if necessary, then creates a new *and* node for the clause and finally initiates a propagation phase to transfer the labels from the b_i to a. Further propagation carries the effect to the direct and indirect consequences of a.

Two features of the ATMS algorithms streamline propagation. First, calculating new explanations does not require searching. All of the information needed to calculate the new explanations is propagated to where it is needed. In the example above, there was no need to search beyond Xis1 and Yis2 to generate the label for Zis3.

Second, redundant and inconsistent environments are not propagated. Since the label contains all of the explanations, any duplicate or subsumed explanation can easily be detected. Inconsistent explanations are supersets of a nogood, so they can be detected. In the example above Xis1 has two environments and Yis2 also has two environments. Thus, there could have been four environments propagated to Zis3, but instead only two were actually propagated; the subsumed and inconsistent environments were not.

3.2 The MESON proof procedure

The MESON proof procedure [Loveland 78] is a variant of linear resolution that is convenient to automate. It operates on clauses, and so is not restricted to the Horn subset. We consider only the propositional form of MESON in this paper.

We need two preliminary definitions. Let the $\neg$ function mean "complement of" when applied to a literal so that if a is a negative literal then $\neg a$ is positive. A *contrapositive rule* from a clause

$a_1 \vee \ldots \vee a_n$

is for any i

$a_i \leftarrow \neg a_1,\ldots,\neg a_{i-1}, \neg a_{i+1},\ldots \neg a_n$

To build a MESON proof of a literal L from a set P of clauses, build a goal tree for L using contrapositives of the clauses. In that tree, a goal is satisfied in one of two ways. One way is to find a contrapositive rule with that goal to the left of the arrow such that all the literals to the right of it can be satisfied. These literals become children of L in the goal tree. The other way is to look for $\neg$L among the ancestors of L in the goal tree.

Consider the clauses

$p \vee \neg a$

$p \vee \neg b$

$a \vee b$

The contrapositive rules are

$p \leftarrow a$

$\neg a \leftarrow \neg p$

$p \leftarrow b$

$\neg b \leftarrow \neg p$

$a \leftarrow \neg b$

$b \leftarrow \neg a$

When we build a tree to satisfy p, we use $p \leftarrow a$, and then try to satisfy a. We may choose the rule $a \leftarrow \neg b$ and then $\neg b \leftarrow \neg p$. Since $\neg p$ has p for an ancestor, it is satisfied. See Figure 2.

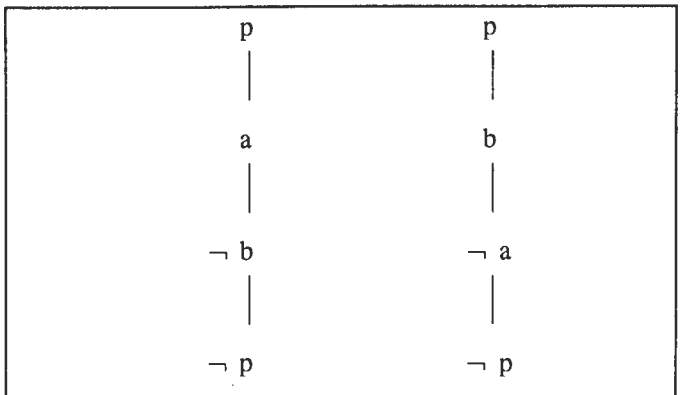


Figure 2 MESON proof trees

3.3 The Ordered Clause refinement of MESON

Figure 2 shows two MESON proofs of p. Both use essentially the same argument, but they use different contrapositive rules from the same clauses. This redundancy is common with MESON proofs, and it grows worse for more complex clause sets.

We can remove this redundancy with a simple extra condition. Assign a total order to the clauses. Build the goal tree as before. If a goal g was introduced to the tree by a clause with ordinal value M, and its ancestor is g was satisfied by a clause with ordinal value N then accept the complement ancestor proof if $M \geq N$.

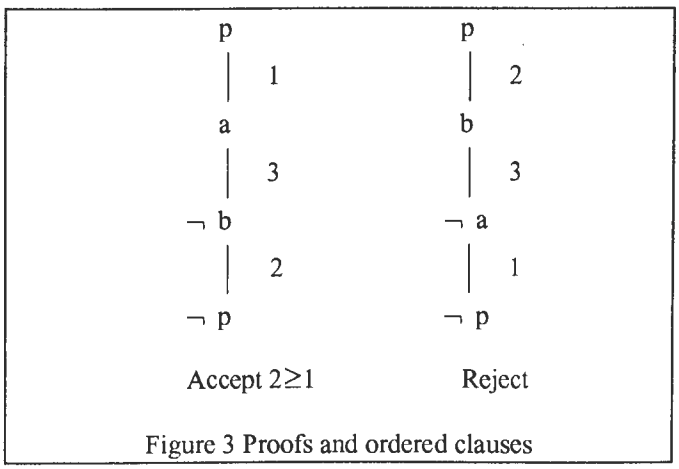


Figure 3 Proofs and ordered clauses

Figure 3 shows the effect of this condition on the proofs above when the clauses are ordered as

1: $p \vee \neg a$
2: $p \vee \neg b$
3. $a \vee b$

This restriction of MESON proofs preserves completeness and sometimes greatly improves efficiency because redundant proofs can be detected and avoided before they are completely built. In some cases the speed up is exponential.

4 Union of the MESON proof procedure and an ATMS structure

We feel it is natural to ask if the ATMS can be extended to employ the MESON proof procedure. Apparently two additions are necessary. For each clause of n literals

$L_1 \vee ... \vee L_n$.

there must be n justifications added to the truth maintenance system

$Li \leftarrow \neg L_1, ..., \neg L_{i-1}, \neg L_{i+1}, ..., \neg L_n$

Also some manner for reasoning by cases in the ATMS must be found to take the place of MESON's complement ancestor proof.

4.1 Ancestor Path Graph

We propose adding to the ATMS a separate justification structure, the ancestor path graph (APG), which contains the clauses that lead from a goal to its complementary ancestor in a proof tree. A justification is put in the APG only when the ATMS is required to prove a goal g and this justification is needed on a path from a literal to its complementary ancestor in a MESON proof of the goal. See Figure 4.

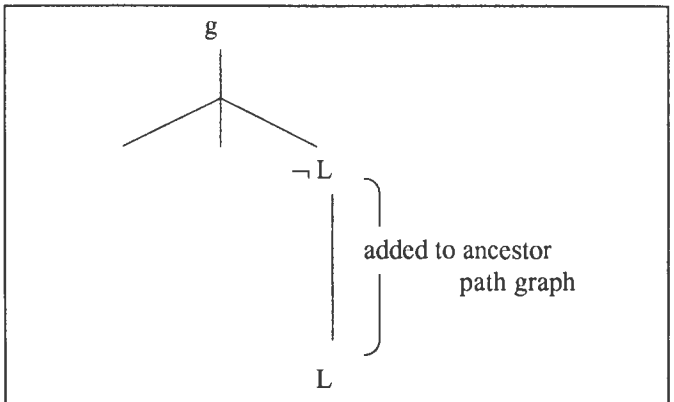


Figure 4 Ancestor Path Justifications

Thus our truth maintenance system has two components: one stores the usual ATMS graph plus the ancestor path graph, and the other searches for paths to add to the APG. Note that the searcher does not need to explicitly store the MESON proof, but it does need to traverse the implicit MESON proof.

Note, too, that it is not necessary to traverse every MESON proof of the goal. We discuss this more in the upcoming sections.

4.2 Propagation in the APG

Whenever a path of clauses is added to the ancestor path graph, the literal on the descendent end of that path, say L, is treated the same way an assumption is treated in the normal ATMS. That is, an environment containing only L is added to L's own label in the APG. This environment is propagated to its ancestors of L. Whenever an environment E containing L is added to the label for L then we have determined that $E - \{L\} \cup \{L\} \models \neg L$. But this means that $E - \{L\} \models \neg L$ so we can propagate $E - \{L\}$ to the *or* node for $\neg L$ in the normal ATMS graph. This simple addition to the ATMS algorithms ensures that we can prove every true conclusion from our set of propositional clauses. That is, we have completeness for propositional logic in the ATMS. This addition requires a minor extension to the existing ATMS algorithms.

4.3 Including the Ordered Clause Restriction

The ordered clause restriction of MESON is only a restriction on what ancestor paths ought to be accepted. We can incorporate this restriction into our ATMS simply by using a given ordering of the clauses and by not adding any paths to the ancestor path graph if the first contrapositive on the path is ordered after the last one.

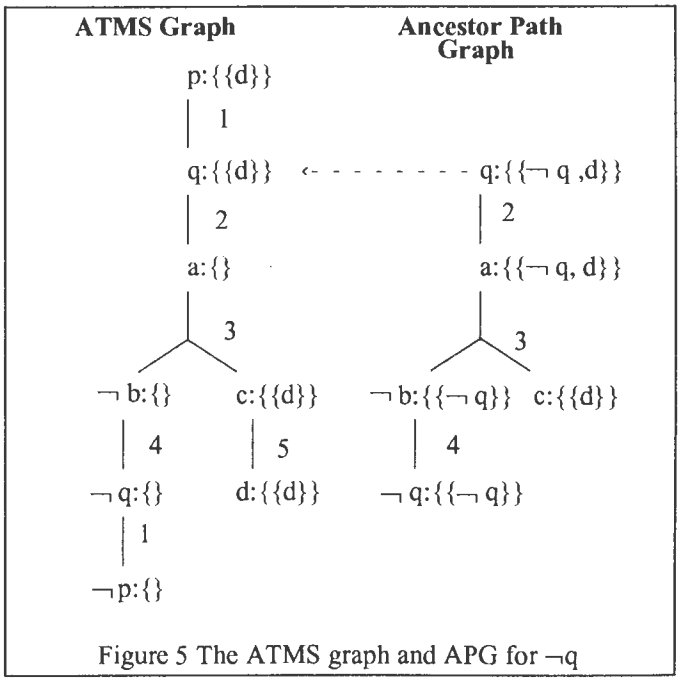


Figure 5 The ATMS graph and APG for $\neg q$

Suppose we are given the ordered clauses below, and we are told to assume d.

1. $p \vee \neg q$
2. $q \vee \neg a$

3. $a \vee b \vee \neg c$
4. $q \vee \neg b$
5. $c \vee \neg d$

Figure 5 shows a portion of the main ATMS and/or graph, including some of the justifications. If only this graph were used, the label of p would be empty. By using the MESON-like searcher, some contrapositives are identified to be added to the ancestor path graph for $\neg q$. Notice that ($q \leftarrow b$), ($b \leftarrow \neg a, c$) and ($\neg b \leftarrow \neg q$) are not added because they form a path rejected by the ordered clause restriction. Figure 5 shows the complete ancestor path graph for $\neg q$. The key idea for propagation is that the label {{d}} of q is propagated from the ancestor path graph on the right to the node q in the main ATMS graph since {{$\neg q$, d}} is the label for q in the ancestor path graph for $\neg q$. Thus the complete label for p is {{d}}.

4.4 The Payoff

The integration of our variant of MESON into an ATMS structure now provides us with an assumption-based truth maintenance system that can handle non-Horn clauses as input and that includes a mechanism for removing redundant proofs, making this a tool that is reasonable to use for various automated reasoning tasks. In the following section, we elaborate on possible uses for this particular ATMS and compare its performance to alternate tools.

5 Flexible completeness and examples of use

The ATMS described in section 4 is designed to allow flexible use — namely, to reduce computation, when users specifically request any proof that works rather than requiring a list of all possible proofs. For any problem where reasoning through cases is required, we may set the searcher to find one, some or all possible proofs. This feature is illustrated in the circuit example below.

5.1 Circuit diagnosis example

Consider the circuit diagnosis example in Figure 6. G1 is an OR gate and G2 is an AND gate. Suppose a gate can malfunction in one of two ways: tied high, which means it always produces a 1 output, or tied low, 0 output. Then each gate is in one of three states, working, high or low. We use propositional symbols, such as G1Works to represent statements about the world.

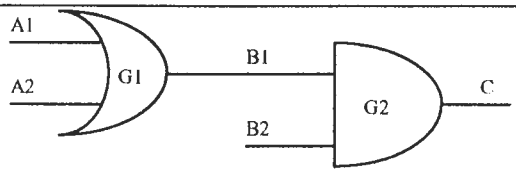


Figure 6: A simple circuit

Thus

G1Works $\vee$ G1High $\vee$ G1Low $\leftarrow$

says that either G1 is working or it is low or it is high.

These four clauses express the OR truth table about G1, assuming that G1 works. B1is0 says that B1's value is 0.

B1is0 $\leftarrow$ G1Works, A1is0, A2is0
B1is1 $\leftarrow$ G1Works, A1is0, A2is1
B1is1 $\leftarrow$ G1Works, A1is1, A2is0
B1is1 $\leftarrow$ G1Works, A1is1, A2is1

Similar clauses are provided for G2.

The next clauses describe the effect of faults on the output of a gate.

B1is0 $\leftarrow$ G1Low
B1is1 $\leftarrow$ G1High

Similar clauses are provided for G2.

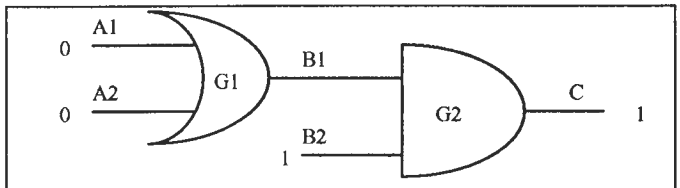


Figure 7 Initial Values (Time 1)

Suppose the input values are set as in Figure 7 and we observe that C's value is 1. One diagnosis of this error is that G1 is tied high while G2 works; another is that G2 is tied high. These are the only possibilities. Using a truth maintenance system, if a set of conditions leads to a conclusion that is contrary to observations, then the assumptions underlying that conclusion are contradicted (declared nogood). This may allow us to conclude that other assumptions are correct. After the first observation, we conclude that G2 is either high or working. We have reduced the possible states from nine (three possible states for each of two gates) to four: {{G1High, G2Works}, {G1Works, G2High}, {G1High, G2High}, {G1Low, G2High}}.

A user who required merely one possible proof could simply terminate interaction with the system at this point. In general with circuit diagnosis, a user would test a further observation and narrow down the range of possibilities. (This step-at-a-time request for proofs is typical of the plan recognition setting as well, where each new observation constrains the interpretation of possible plans.)

Continuing with the example, to further reduce the possibilities, we need more observations. Suppose we set B2 to 0 and then observe C is 0, as in Figure 8.

Then we can eliminate the conclusion that G2 is tied high, leaving just one possible diagnosis, that G1 is tied high and G2 is working.

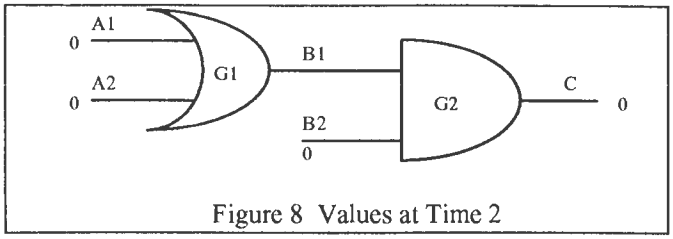


Figure 8 Values at Time 2

The second observation leads us to reject the assumption that G2 is low, nogood(G2Low), so we can be sure it is working. Returning to the first situation, since G2 works, we can be sure B1 is 1 which implies that G1 must be high.

We described the circuit diagnosis example to our truth maintenance system in clausal notation. The clauses described the action of each gate under working conditions and under each of the faulty conditions. It was told to assume G1Works, G1High, G1Low, G2Works, G2High and G2Low. Each of the observed inputs were entered as a single-literal clause with a time stamp, such as A1is0(1). Finally clauses describing the observed outputs were added:

obs ← Cis1(1), Cis0(2).

The label generated for obs after all of these additions was

{{G1High, G2Works}}

which concurs with our analysis. In fact, for this example, when all possible proofs were requested, following two separate observations, no extra searching was needed, and no paths were added to the ancestor path graph. Occasionally reasoning by cases is not needed to reach desired conclusions from non-Horn clauses.

6 Comparisons to other systems

6.1 Comparison with de Kleer's extended ATMS

The extended ATMS [de Kleer 86b] and our proposed truth maintenance system compute sound, complete, consistent and minimal explanations of literals. De Kleer's extended ATMS allows the problem solver to express disjunctions only in a restricted form

choose$\{A_1,...,A_n\}$

where each A_i is a positive ATMS assumption. It is possible to encode any clause as a set of Horn clauses and a set of disjunctions of positive assumptions. Four encoding methods are given by de Kleer; each method involves introducing new assumptions that are not relevant to the problem at hand. These so-called encoding assumptions ought not to be revealed to the problem solver, so any explanation that involves an encoding assumption should be ignored.

In order to reason with disjunctions, two new procedures are needed, positive hyperresolution and negative hyperresolution. The negative hyperresolution procedure finds nogoods that arise as a consequence of disjunctions. It implements the following rule:

choose$\{A_1, ..., A_n\}$
nogood α_i where $A_i \in \alpha_i$ and $A_{j\neq i} \notin \alpha_i$ for all i

---

nogood $\cup_i(\alpha_i - \{A_i\})$

In order to perform this step we must find a choose statement such that for each assumption there is a nogood with a singleton intersection with the choose. Then from the remainders of these nogoods we build a new nogood.

To discover all nogoods with the negative hyperresolution rule, this procedure must be applied whenever a new choose or a new nogood is given or discovered. Whenever the procedure succeeds, a new nogood is discovered, and the search must begin anew.

The positive hyperresolution procedure finds the new environments for propositions. See [de Kleer 86b] for more details.

Various special cases of each of these rules can be efficiently implemented. For example when a choose is a singleton, it can be removed from each nogood. The general cases, however, are still necessary.

The hyperresolution procedures in the extended ATMS are considered expensive [de Kleer 86b, de Kleer 88]. There are four reasons for this:

First, encoding adds work. Besides the cost of automatically translating non-Horn clauses into ATMS inputs, each assumption introduced for encoding adds some work to the ATMS. In the worst case, each assumption could double the amount of work, since the amount of work is proportional to the number of environments, which is exponential in the number of assumptions.

Second, hyperresolution must be applied frequently, to guarantee consistency and completeness. Both positive hyperresolution and negative hyperresolution will be invoked each time a new choose is declared and each time a new nogood is discovered. Positive hyperresolution will also be invoked as a new environment is added to a label, which is the most common operation.

Third, new information is not propagated to where it is needed. Hyperresolution must search for relevant information. For instance, a new choose requires positive hyperresolution to search the entire database, and parts of it many times. This is contrary to the design philosophy of the basic ATMS, where discovered information is propagated to all of the places it is needed.

Last, hyperresolution might produce uninteresting information. Positive hyperresolution may add environments containing encoding assumptions, that will be ignored on output. Negative hyperresolution may discover nogoods with encoding assumptions, so explanations with them would not have been reported anyway. In these cases the results are not relevant to the problem solver.

Our solution, with ancestor path graphs, does not exhibit these causes of inefficiency. Since the input form of the clauses is not restricted, there are no encoding

assumptions. This covers the first and last causes. The second does not apply. As for the third, our system does propagate discovered information to where it is needed, and integrates well with the basic ATMS.

Three criticisms can be raised against our solution. Our and/or graph contains all of the contrapositives of the clauses and additional and/or graphs, while de Kleer's extended ATMS just contains one form of each clause. This will increase our storage requirements and also the amount of propagation that is needed. It is difficult to compare how this affects the overall runtime however, since the propagation in our system is doing the same work that is done by searching in de Kleer's system. Next, our system requires an extra search phase for ancestor paths. We pointed out how this search can be controlled, by use of ordered clauses, and that it is not always necessary to do it. Our circuit example shows that useful conclusions can still be drawn. Finally our solution finds explanations for all positive and negative literals, whereas de Kleer's only explains positive literals. This additional work may be an advantage of our system, when there is interest in explanations for negative literals. What our system is designed to do, therefore, is to be generally applicable to a wide range of problems, in tune with the design of the basic ATMS.

The cases where additional computation is required are those cases where the computation is necessary, for completeness and the system is built to take advantage of reduction in proofs, wherever possible.

6.2 Comparison with PTTP

There are advantages of our ordered clause restriction which other theorem provers do not incorporate. Basically, a conclusion may be reached much faster, since redundant parts of the search space can be detected and avoided.

For example Stickel's Prolog Technology Theorem Prover, PTTP [Stickel 88] was asked to find one proof of s from the clauses below:

$s \leftarrow p, q, r, w$
$p \leftarrow q, r$
$q \leftarrow p, r$
$r \leftarrow p, q$
$p \vee q \leftarrow r$
$p \vee r \leftarrow q$
$q \vee r \leftarrow p$
$p \vee q \vee r \leftarrow$
s

The time was long (1.7 seconds) because PTTP first chose the first clause to prove s. That clause was doomed to fail because there is no clause mentioning w. On backtracking the final clause is tried, the unit clause s, and the proof succeeds

Our system was given the same set of clauses and the searcher was asked to find one proof of s. It, too, attempted the first clause first, but because of the ordered clause restriction, it concluded that there was no proof with that clause in 0.67 seconds. It then went on to find the proof. (All programs are written in Quintus Prolog 3.1.1 run on a Sun 670MP.)

This example illustrates that in cases where one proof is required there may be additional expense, and that this expense can be reduced by the ordered clause restriction.

7 Conclusions

We have presented an ATMS which can reason with non-Horn clauses and integrates well with de Kleer's original ATMS. In developing this ATMS we have shown how to integrate a MESON type proof procedure, but to do so by incorporating a restriction of the amount of search that has to be performed. This restriction makes our system faster in some cases than PTTP, a high performance reasoning system. Moreover, the ATMS is designed to permit a "pay as you go" policy for computing a complete set of environments. Users have the flexibility to further reduce computation by requiring one or some of a set of proofs, foregoing completeness. We feel that the resulting ATMS is useful, therefore, for reasoning with a variety of problems in an efficient and flexible manner.

References

[Cohen and Spencer 93] Robin Cohen and Bruce Spencer, Specifying and Updating Plan Libraries for Plan Recognition Tasks, In Proceedings of the IEEE Conference on Artificial Intelligence Applications, Orlando, Florida, 27-33, March 1993.

[de Kleer 86a]. Johan de Kleer, An assumption-based Truth Maintenance System. Artificial Intelligence, 28:163-196, 1986.

[de Kleer 86b] Johan de Kleer, Extending the ATMS. Artificial Intelligence, 28:128-162, 1986

[de Kleer 88] Johan de Kleer. A General Labeling Algorithm for Assumption-based Truth Maintenance. In Proceedings AAAI-88 Seventh National Conference on Artificial Intelligence, 188-192, 1988.

[Kautz 87] Henry Kautz, A Formal Theory of Plan Recognition. Ph.D. Thesis, University of Rochester, 1987. Available as Technical Report 215.

[Kean and Tsiknis 93] Alex Kean and George Tsiknis, Clause Management Systems (CMS), Computational Intelligence, Blackwell Publishers, Cambridge MA and Oxford UK, 9: 11-40, 1993.

[Loveland 78] Donald Loveland. Automated Theorem Proving: A logical Basis. North Holland, Amsterdam, 1978.

[Poole 87] David Poole, Randy Goebel and Romas Aleliunas. Theorist: a logical reasoning system for defaults and diagnosis. In Nick Cercone and Gord McCalla, editors, The Knowledge Frontier: Essays in

the Representation of Knowledge, 331-352. Springer Verlag, New York, 1987.

[Reiter 80] Raymond Reiter. Logic for Default Reasoning. Artificial Intelligence, 13:81-132, 1980.

[Reiter and de Kleer 87] Raymond Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: Preliminary Report. in Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence, 183-188, 1987.

[Spencer 90] Bruce Spencer, Assimilation in Plan Recognition via Truth Maintenance with Reduced Redundancy, University of Waterloo Ph.D. Thesis, available as Technical Report TR90-060, Faculty of Computer Science, University of New Brunswick, Fredericton, New Brunswick, Canada.

[Spencer 91] Bruce Spencer, Linear Resolution with Ordered Clauses. In Proceedings of the Workshop on Disjunctive Logic Programming, held in conjunction with the International Logic Programming Symposium, 1991.

[Spencer 93] Bruce Spencer, The Ordered Clause Restriction of Model Elimination and SLI. In Proceedings of the International Logic Programming Symposium, 678, 1993.

[Stickel 88] Mark Stickel, A Prolog Technology Theorem Prover. Journal of Automated Reasoning, 4:353-380, 1989.

An Event-Based Abductive Model of Update

Craig Boutilier
Department of Computer Science
University of British Columbia
Vancouver, British Columbia
CANADA, V6T 1Z4
email: cebly@cs.ubc.ca

Abstract

The Katsuno and Mendelzon theory of belief update has been proposed as a reasonable model for revising beliefs about a changing world. However, the semantics of update relies on information which is not readily available. We describe an alternative semantical view of update in which observations are incorporated into a belief set by: a) explaining the observation in terms of a set of plausible events that might have caused that observation; and b) predicting further consequences of those explanations. We also allow the possibility of *conditional explanations*. We show that this picture naturally induces an update operator under certain assumptions. However, we argue that these assumptions are not always reasonable, and they restrict our ability to integrate update with other forms of revision when reasoning about action.

1 Introduction

Reasoning about action and change has been a central focus of research in AI for many years, dating back at least to the origins of the situation calculus (McCarthy and Hayes 1969). For example, a planning agent must be able to predict the effects of its actions on the world in order to verify whether a potential plan achieves a desired goal. Actions can be viewed as effecting changes in the world, and agents must be able to change their beliefs about the world to reflect such considerations.

One of the most influential theories of belief change has been the *AGM theory* proposed by Alchourrón, Gärdenfors and Makinson (1985). Imagine an agent possesses a belief set or knowledge base KB. The AGM theory provides a set of postulates constraining the possible ways in which the agent can change KB in order to accommodate a new belief A. Notice that this *revision* of KB need not be straightforward, for the new belief A may conflict with beliefs in KB. It was pointed out by Winslett (1988) that the AGM theory is inappropriate for reasoning about changes in belief due to the evolution of a changing world. A new form of belief change dubbed *update* was proposed in full generality by Katsuno and Mendelzon (1991), who provided a set of postulates, distinct from the AGM postulates, that characterize this type of belief change.

Semantically, Katsuno and Mendelzon have shown that belief update can be viewed by positing a family of orderings over possible worlds, with each ordering being indexed by some world. The ordering associated with a specific world can be viewed intuitively as describing the most plausible ways in which that world can change. To update a knowledge base KB with some proposition A, the worlds admitted by KB are each updated by finding the most plausible change associated with that world satisfying A (we describe this formally below).

In this paper, we present an abductive view of update that breaks the Katsuno-Mendelzon semantics into smaller, more primitive parts. We argue that such a model provides a more natural perspective on belief update in response to changes in the world, and exploits information that is more readily available. In general, we take update to be a two stage process of explanation followed by prediction: first, an agent *explains* an observation by postulating some *plausible event* or events that could have caused that observation to hold, relative to its initial state of knowledge; second, an agent *predicts* the (further) consequences of these events, relative to this initial state. We formalize this notion in an abstract manner obtaining a class of *explanation-change* operators. We show that explanation-change satisfies some of the properties of update operators determined by the Katsuno-Mendelzon (KM) theory. Furthermore, if we make two additional assumptions our model determines a KM update operator. However, we will argue that these additional assumptions are not always appropriate. In particular, should we intend to use update to reason about action, and have the results of actions provide information about the state of the world, the general form of update has to be modified. This modification is pursued in (Boutilier 1993; Boutilier 1994b).

We also briefly describe and characterize a special class of update operators. Finally, we compare our construction to the model of update proposed by del Val and Shoham (1992). Proofs of the results can be found in (Boutilier 1994b).

2 The Semantics of Update

Katsuno and Mendelzon (1991) have proposed a general characterization of belief update. Update is distinguished from belief *revision* conceptually by viewing update as reflecting belief change in response to changes in the world, whereas revision is thought to be more appropriate for changing (possibly erroneous) beliefs about a static world. Update is described by Katsuno and Mendelzon with a set of postulates constraining acceptable update operators and a possible worlds semantics, which we review here.

We assume the existence of some knowledge base KB, perhaps the set of beliefs held by an agent about the current state of the world. We take our underlying logic to be propositional, based on a finitely generated language $\mathbf{L}_{CPL}$. We use W to denote the set of *possible worlds* (or models) suitable for this language.

If some new fact A is observed in response to some (unspecified) change in the world (i.e., some action or event occurrence), then the formula $KB \diamond A$ denotes the new belief set incorporating this change. The *KM postulates* (Katsuno and Mendelzon 1991) governing admissible update operators are

(U1) $KB \diamond A \models A$

(U2) If $KB \models A$ then $KB \diamond A$ is equivalent to KB

(U3) If KB and A are satisfiable, then $KB \diamond A$ is satisfiable

(U4) If $\models A \equiv B$ then $KB \diamond A \equiv KB \diamond B$

(U5) $(KB \diamond A) \wedge B \models KB \diamond (A \wedge B)$

(U6) If $KB \diamond A \models B$ and $KB \diamond B \models A$ then $KB \diamond A \equiv KB \diamond B$

(U7) If KB is complete then $(KB \diamond A) \wedge (KB \diamond B) \models KB \diamond (A \vee B)$

(U8) $(KB_1 \vee KB_2) \diamond A \equiv (KB_1 \diamond A) \vee (KB_2 \diamond A)$

A better understanding of the mechanism underlying update can be achieved by considering the possible worlds semantics described by Katsuno and Mendelzon, which they show to be equivalent to the postulates. For any proposition A, let $||A||$ denote the set of worlds satisfying A. Clearly, $||KB||$ represents the set of possibilities we are prepared to accept as the actual state of affairs. Since observation O is the result of some change in the actual world, we ought to consider, for each possibility $w \in ||KB||$, the most plausible way (or ways) in which w might have changed in order to make O true. To capture this intuition, Katsuno and Mendelzon postulate a family of preorders

$$\{\leq_w : w \in W\}$$

where each $\leq_w$ is a reflexive, transitive relation over W. We interpret each such relation as follows: if $u \leq_w v$ then u is at least as plausible a change relative to w as is v. Finally, a *faithfulness condition* is imposed: for every world w, the preorder $\leq_w$ has w as a minimum element; that is, $w <_w v$ for all $v \neq w$.

Naturally, the most plausible candidate changes in w that result in O are those worlds v satisfying O that are minimal in the relation $\leq_w$. The set of such minimal

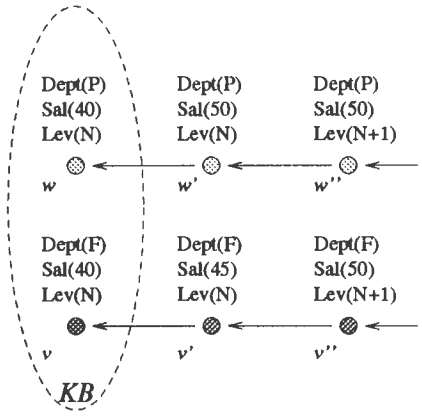


Figure 1: An Update Model

O-worlds for each relation $\leq_w$, and each $w \in ||KB||$, intuitively capture the situations we ought to accept as possible when updating KB with O. In other words,

$$||KB \diamond O|| = \bigcup_{w \in ||KB||} \{\min_{\leq_w}\{v : v \models O\}\}$$

where $\min_{\leq_w} X$ is the set of minimal elements (w.r.t. $\leq_w$) within X. Katsuno and Mendelzon show that such a formulation of update captures exactly the same class of change operators as the postulates; thus, we can treat this as an appropriate semantics for update.

As an example, consider the following scenario illustrating the application of the KM update semantics to database update. We know certain facts about an employee Fred: his salary is \$40,000, his job classification is level N, and so on. But, we are unsure whether he works for the Purchasing department or the Finance department. Thus, our KB admits two possibilities, w and v, reflecting this uncertainty (see Figure 1). If the orderings $\leq_w$ and $\leq_v$ are as indicated in the figure, then KB updated with the fact that Fred's salary is \$50,000 contains, among other things, the facts `Dept(P)` $\vee$ `Dept(F)`, `Dept(P)` $\supset$ `Level(N)` and `Dept(F)` $\supset$ `Level(N+1)`. This is due to the fact that the closest world to w with the new salary is w', while the closest to v is v''; hence, KB is determined by the set of worlds $\{w', v''\}$. This may reflect the fact that such a raise comes only with a promotion in Finance, whereas promotions are rare and raises more frequent in Purchasing.

3 Update as Explanation

3.1 Plausible Causes of Observations

The orderings upon which update semantics are based are interpreted as describing the most plausible manner in which that world might change. Given the role of update, this interpretation seems correct: worlds closer to w in the ordering $\leq_w$ are somehow more plausible states into which w might evolve. It seems reasonable then to update a KB by considering those most plausible changes. In our example above, if Fred is in Purchasing (world w), then a change of salary of this type is more

likely to come without a change in rank (w') than with a change in rank (w'').

While reasonable, it begs the question: why would one change be judged more plausible than another? Intuitively, it seems that there are certain *events* or *actions* that would *cause* a change in w, and that those leading to w' are more plausible than those leading to w''. For example, the event **RAISE** might be more probable than the event **PROMOTION** (at least, in Purchasing).

Given an observation `Sal(50000)` — in this case an update transaction — an agent might come to believe `Dept(P)` $\supset$ `Level(N)` (as we have in our example) as follows. Assuming `Dept(P)`, the most plausible event that might *cause* such a change in salary is **RAISE** (rather than **PROMOTION**). Thus **RAISE** is the best *explanation* for the observation. Adopting this explanation has, as a further consequence, that job rank (and department) stays the same; thus, belief in `Level(N)` remains. In contrast, **RAISE** (to \$50,000) is less likely than **PROMOTION** in the Finance department.[1] Thus, **PROMOTION** is the most plausible explanation for the observation, which has the additional consequence `Level(N+1)`. Thus, the two beliefs `Dept(P)` $\supset$ `Level(N)` and `Dept(F)` $\supset$ `Level(N+1)` hold in the updated belief state.

This leads to a very different view of update. When confronted with an observation or update O, an agent seeks an *explanation* of O, in terms of some external event that would have caused O had it occurred.[2] While many events might explain O in this way, some will be more plausible than others, and it will be those the agent adopts. Given such an explanation, one may then proceed to *predict* further consequences of these events, and produce the set of beliefs arising from the observation. With this point of view, the essence of update is captured by a two-step process: a) *explanation* of the observation in terms of some event(s); and b) *prediction* of the (additional) consequences of that event.

Before formalizing this idea, it is important to realize that this perspective is very natural. It is reasonable to suppose that an agent (or builder of a KB) has ready access to some description of the preconditions and effects of the possible events in a given domain. This assumption underlies all work in classical planning and reasoning about action, ranging from STRIPS (Fikes and Nilsson 1971) to the situation calculus (McCarthy and Hayes 1969; Reiter 1991) to more sophisticated probabilistic representations (Kushmerick, Hanks and Weld 1993; Dean et al. 1993). With such information, the predictions associated with explanations (event occurrences) can be easily determined. Furthermore, an ordering over the relative likelihood of possible events also seems something which an agent or system designer or user might easily postulate. This should certainly be easier to construct than a direct ordering over worlds according to their likelihood of "occurring." Indeed, we will show that such an ordering over worlds is *derivable* from this more readily available information.

This provides a possible interpretation of the update process, and in our view, a very natural one.[3] Furthermore, as we describe in the concluding section (and in detail in (Boutilier 1993)), by breaking update into two components, we will be able to extend the type of reasoning about action one can perform in this setting.

Using explanation for reasoning about action has been proposed by a number of people, especially within the framework of the situation calculus. Work on temporal projection and prediction failures often exploits the notion of explanation. For instance, Morgenstern and Stein (1988) propose a model where an observation that conflicts with the predicted effects of an agent's actions causes the agent to infer the existence of some external event occurrence. Shanahan (1993) proposes a model with a similar motivation, but adopts a truly abductive model (where candidate events are hypothesized rather than deduced from an observation). Our model will be rather different in several ways. First, explanations will be *conditional* (i.e., explaining events are conditioned on certain propositions). Second, the criteria used for adopting explaining events will be based on the relative plausibility of events. Third, we will not limit attention to any particular model of action (such as the situation calculus). Finally, our goal is to show how explanation can account for the *update* of a knowledge base. We should point out that Reiter (1992, and personal communication) has informally suggested that update can be viewed as explanation to events causing an observation. We will now proceed to show that this is, in fact, the case.

3.2 A Formalization

To capture update in terms of explanation, we require two ingredients missing from the Katsuno-Mendelzon account: a set of *events* that cause changes, and an *event ordering* that reflects the relative plausibility of different event occurrences.

We assume a finitely generated propositional language with an associated set of worlds W. Let E be a finite *event set*, the elements of which are primitive events. In general, $e \in E$ is a mapping $e : W \rightarrow 2^W$. For $w \in W$ and $e \in E$, we use $e(w)$ to denote the *result* of event e occurring in world w. This is a set of worlds, each of which is a possible *outcome* of e occurring at w. An event with more than one possible outcome is *nondeterministic*. A *deterministic* event is any $e \in E$ such that $e(w)$ is a singleton set for each $w \in W$. A *deterministic event set* is an event set all of whose events are deterministic. We assume that events are total functions on the domain W,

[1] In our example, we assume that a raise to \$45,000 is most likely (world v'), but that a higher raise is unlikely without a promotion.

[2] In this paper we will usually think of (external) *events* as the impetus for change, rather than *actions* over which the agent has direct control (or of which the agent has direct knowledge).

[3] This should not be taken as a criticism of update for requiring that a reasoning agent have an explicitly specified family of preorders at its disposal. One can reason about update with syntactic constraints or by any other means. The point is that, from a semantic point of view, the preorders and syntactic constraints seem to be *induced* by considerations about action effects and plausible event occurrences.

so that every event can be applied to each world.[4]

Typically, events are not specified as mappings of this type. Rather, for each event (or action), a list of conditions are provided that influence the outcome of the event. For each such condition, a set of effects is specified. An example of this is the classical situation calculus representation of actions (in the deterministic case). Another is the modified STRIPS representation presented in (Kushmerick, Hanks and Weld 1993). The key feature of these, and other representations, is that each action/event induces a function between worlds (or worlds and sets of worlds).[5] Thus, most action representations will fit within this abstract model.

As a further generalization, if events are nondeterministic, we might suppose that the possible outcomes are ranked by probability or plausibility. We set aside this complication (but see (Boutilier 1993)).

In order to explain certain observations by appeal to plausible event occurrences, we need some metric for ranking such explanations. We assume that the events in the set E are ranked by plausibility; hence, we postulate an indexed family of *event orderings*

$$\{\preceq_w : w \in W\}$$

over E. We take $e \preceq_w f$ to mean that event e is at least as plausible (or likely to occur) as event f in world w. We require that $\preceq_w$ be a preorder for each w, and will occasionally assume that $\preceq_w$ is a total preorder.

Putting these ingredients together, we have the following definitions:

Definition An *event model* is a triple $\langle W, E, \preceq \rangle$, where W is a set of worlds, E is a set of events (mappings $e : W \rightarrow 2^W$) and $\preceq$ is an indexed family of events orderings $\{\preceq_w : w \in W\}$ (where each $\preceq_w$ is a preorder over E).

Definition A *deterministic event model* is an event model where every $e \in E$ is deterministic (i.e., for all $w \in W$, $e(w) = \{v\}$ for some $v \in W$). A *total order event model* is an event model where each event ordering $\preceq_w$ is a total preorder over E.

Given an event model, an agent is able to incorporate a new piece of information through a process of explanation and prediction as discussed above. An explanation of an observation is some event e that, when applied to the world under investigation, possibly causes O. However, the agent should be interested only in the most plausible such events.

Definition Let O be some proposition and $w \in W$. The set of *weak explanations* of O relative to w is

$$Expl(O, w) = \min_{\preceq_w} \{e \in E : e(w) \cap ||O|| \neq \emptyset\}$$

[4] It is best to think of events as analogous to "action attempts." If the preconditions for the "successful" occurrence of the event are not true at a given world, then the effects can be null, or unpredictable or something like that. Allowing preconditions is a trivial and uninteresting addition for our purposes here.

[5] In the case of the situation calculus, dynamic logic or other temporal formalisms, one would require some solution to the frame problem. For example, the solution of Reiter (1991) induces just such a mapping.

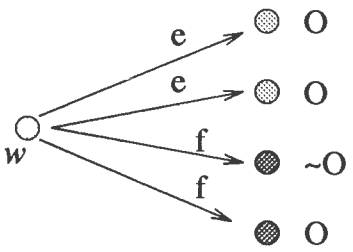


Figure 2: Weak and Strong Explanations

An event e is a *weak explanation* of O relative to w iff $e \in Expl(O, w)$. If $Expl(O, w) = \emptyset$, we say that O is *unexplainable* relative to w.

In other words, e explains O in a world w just when there is some possible result of e that satisfies O, and no more plausible event e' has this feature. Such explanations are called weak explanations because, before the observation O is made, an agent would not, in general, be able to *predict* that O would result from e. The agent merely knows that O is true in *some* possible outcome. A strong explanation is similar, but is predictive: *each* outcome of e satisfies O.

Definition The set of *strong explanations* of O relative to w is

$$\min_{\preceq_w} \{e \in E : e(w) \subseteq ||O||\}$$

The distinction between weak and strong explanations is very similar to that made between *consistency-based* diagnosis (Reiter 1987) and *predictive* (or *abductive*) diagnosis (Poole 1988). This distinction is illustrated in Figure 2. Both e and f are nondeterministic events. Event e strongly explains O, while f weakly explains O but does not strongly explain O. We are interested here in weak explanations, for these seem most appropriate when dealing with nondeterministic events. However, we note the following:

Proposition 1 *If e is a deterministic event, then e weakly explains O iff e strongly explains O.*

For a particular world w, $Expl(O, w)$ denotes those most plausible events that would cause O to be true. The possibilities admitted by such a set of explanations are the possible results of each of these events; that is:

Definition The *result* of O relative to w is the set of worlds

$$Res(O, w) = \bigcup \{e(w) \cap ||O|| : e \in Expl(O, w)\}$$

Note that if O is unexplainable relative to w, then $Res(O, w) = \emptyset$. Thus, that w might have evolved into a world satisfying O is impossible.

Taking a cue from the Katsuno-Mendelzon update semantics, the result of an observation with respect to a knowledge base KB is obtained by considering all plausible evolutions of each world $w \in ||KB||$. However, if O is unexplainable for some $w \in ||KB||$, we take O to be unexplainable relative to KB as a whole.

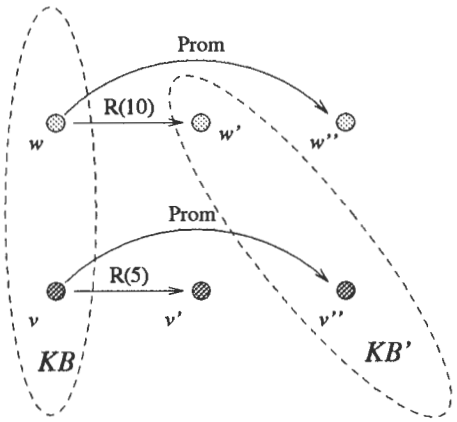


Figure 3: An Event Ordering

Definition The *result* of O relative to KB is the set of worlds

$$Res(O, KB) = \bigcup \{Res(O, w) : w \in ||KB||\}$$

If $Res(O, w) = \emptyset$ for some $w \in ||KB||$, we let $Res(O, KB) = \emptyset$.

The motivation for this last condition, that O must be explainable relative to every $w \in ||KB||$, comes from update semantics itself. In update, the updated KB is constructed by considering the possible evolution of *every* possibility admitted by KB. We might have allowed the result of O to be nontrivial even if some worlds could not evolve so as to satisfy O, and define $Res(O, KB)$ without this last condition. However, we adopt the current approach for two reasons: first, our goal is to pursue the analogy with update semantics; and second, when we drop this restriction, we intend to make this definition even weaker than we can by simply dropping the last condition. In (Boutilier 1993), we consider how to exclude both impossible and *implausible* evolutions. We elaborate on this in the concluding section.

With such a result function, we can now define the *explanation-change operator* relative to a given event model, which determines the consequences of adopting an observation.

Definition The *explanation-change operator* induced by an event model EM is $\diamond_{EM}$:

$$KB \diamond_{EM} O = \{A \in \mathbf{L}_{CPL} : Res(O, KB) \models A\}$$

In our example, we have two event types, Promotion and Raise. A `PROM` event (promotion of one level) ensures an employee's rank is increased and his salary is raised \$10,000. Events `RS(5)` and `RS(10)` raise salary \$5000 and \$10,000, respectively. We assume the following event orderings for each department:

Purchasing: `RS(10)` $\prec$ `PROM` $\prec$ `RS(5)`
Finance: `RS(5)` $\prec$ `PROM` $\prec$ `RS(10)`

This is illustrated in Figure 3, where shorter event arcs depict more plausible occurrences. The explanation relative to purchasing is a raise, while for finance it is a promotion. The updated KB' is determined by w' and v'' and induces the beliefs described earlier.

As another example, imagine that a warehouse control agent expects a series of trucks to pickup and deliver certain shipments, but at time $t1$ an expected truck A has not arrived. Assume that this might be explained by snow on Route 1 or a breakdown. If snow is the most plausible of the two events, the agent might reach further conclusions by predicting the consequences of that event; for example, trucks B and D will also be delayed since they use the same route. The proper explanation and subsequent predictions are crucial, for they will impact on the agent's decision regarding staffing, scheduling and so on. Notice also that such explanations are defeasible, which is reflected in the defeasibility of update: if A is late but B is on time, then snow is no longer plausible (therefore, e.g., D will not be delayed).

We should remark at this point that the intent of this model is to provide an abductive semantic model for update, not a computational model. Just as we do not expect actions or events to be represented as abstract functions between worlds, explanations will not typically be generated on a world by world basis. Usually, the same event will explain an observation for a large subset of the worlds with $||KB||$. In particular, we expect that $||KB||$ to be partitioned according to some small number of propositions (or conditions) for which a certain event is deemed to be a reasonable explanation. Indeed, these can naturally be viewed as *conditional explanations*, for example, "If Fred is in Finance, a `PROMOTION` must have occurred; but otherwise a `RAISE` must have occurred." How such conditional explanations should be generated will be intimately tied to the action or event representation chosen, and is beyond the scope of this paper.

3.3 Relationship to Update

We are interested in the question of whether the explanation-change operator satisfies the update postulates. As presented above, this is not the case.

Proposition 2 *Let $\diamond_{EM}$ be the explanation-change operator induced by some event model. Then $\diamond_{EM}$ satisfies postulates (U1), (U4), (U6) and (U7).*

There are two reasons why the remainder of the postulates are not satisfied in general, hence two assumptions that can be made to ensure that $\diamond_{EM}$ is an update operator.

The first difference in the explanation-change operator is reflected in the failure of (U2), which asserts that $KB \diamond A$ is equivalent to KB whenever KB entails A. A simple example illustrates why this cannot be the case is general. Consider a KB satisfied by a single world w where $w \models A$. Postulate (U2) requires that the observation of A induce no change in KB. However, it may be that the most plausible event in the ordering $\preceq_w$ is e, where $e(w) = \{v\}$ for some distinct world v. But assuming $v \models A$, then $KB \diamond_{EM} A$ is captured by v and is thus distinct from w. In order to conform to postulate (U2), we must make the assumption that no change in w is more plausible than change induced by some event. Formally, we postulate *null events* and make these most plausible.

Definition The *null event* is an event n, where $n(w) = \{w\}$ for all $w \in W$.

Definition Let $EM = \langle W, E, \preceq \rangle$ be an event model. EM is *centered* iff the null event $n \in E$ and, for each $w \in W$ and $e \in E$ ($e \neq n$) we have $n \prec_w e$.

Thus, a centered event model is one in which the null event is the most plausible event that could occur at any world. This seems to be the crucial assumption underlying postulate (U2).

Proposition 3 *Let $\diamond_{EM}$ be the explanation-change operator induced by some centered event model. Then $\diamond_{EM}$ satisfies postulates (U1), (U2), (U4), (U6) and (U7).*

This assumption of persistence of the truth of KB seems to be reasonable in many domains, but should probably be called into question as a general principle. It may be the case in a domain where change is the norm that, despite the fact that an observation is already believed, some change in KB should be forthcoming. In this sense, the more general nature of the explanation-change operator may be desirable.

Postulate (U3) is also violated by our model, and for a similar reason, so too are (U5) and (U8). For a given KB, we may have that $Res(O, w) = \emptyset$ for each $w \in \|KB\|$. In other words, there are no possible events that would cause an observation O to become true. The potential for such unexplainable observations clearly contradicts (U3), which asserts that $KB \diamond O$ must be consistent for any consistent O. The assumption underlying (U3) in update semantics seems to be the following: every consistent proposition is explainable, no matter how unlikely. In order to capture this assumption, we propose a class of event models called *complete*.

Definition Let $EM = \langle W, E, \preceq \rangle$ be an event model. EM is *complete* iff for each consistent proposition O and $w \in W$, O is explainable relative to w.

Proposition 4 *If EM is a complete event model then $Res(O, KB) \neq \emptyset$ for any consistent O and KB.*

Of course, this condition is sufficient to ensure (U5) and (U8) are satisfied as well.

Proposition 5 *Let $\diamond_{EM}$ be the explanation-change operator induced by some complete event model. Then $\diamond_{EM}$ satisfies postulates (U1), (U3), (U4), (U5), (U6), (U7) and (U8).*

The completeness of an event model refers, in fact, to the completeness of its event set E. If this set is rich enough to ensure that, for every world and observation, some event can make that observation hold, then the event model will be complete. Typically, domains will not be so well-behaved. However, the simple addition of a *miracle* event to an event set will ensure completeness. Intuitively, a miracle is some event which is less plausible than all others and whose consequences are entirely unknown.

Definition Let $EM = \langle W, E, \preceq \rangle$ be an event model. A *miracle* is an event m such that $m(w) = W$ for all $w \in W$, and $e \prec_w m$ for all $w \in W$ and $e \in E$ ($e \neq m$).

Proposition 6 *Let $EM = \langle W, E, \preceq \rangle$ be an event model. If E contains a miracle event, then EM is complete.*

If all observations must be explainable, and no observation is permitted to force an agent into inconsistency, then miracles are one embodiment of the required assumptions. The reasonableness of such a requirement can be called into question, however. Having unexplainable observations is, in general, a natural state of affairs. Rather than relying on miraculous explanations, the threat of an inconsistency can force an agent to reconsider the observation, its theory of the world, or both. As we will see in the concluding section, it is just this type of inconsistency that can force an agent to revise its beliefs about the world prior to the observation. Update postulate (U3) makes it difficult to combine update with revision in this way.

If we put together Propositions 3 and 5, we obtain the main representation result for explanation-change.

Theorem 7 *Let $\diamond_{EM}$ be the explanation-change operator induced by some complete, centered event model. Then $\diamond_{EM}$ satisfies update postulates (U1) through (U8).*

A useful perspective on the relationship between explanation change and update comes to light when one considers that the plausibility ordering on events quite naturally induces an indexed family of preorders of the type required in the Katsuno-Mendelzon update semantics.

Definition Let $EM = \langle W, E, \preceq \rangle$ be an event model. The plausibility ordering *induced by EM*, for each $w \in W$, is defined as follows: $v \leq_w u$ iff for any event e_u such that $u \in e_u(w)$, there is some event e_v (where $v \in e_v(w)$) such that $e_v \preceq_w e_u$.

Theorem 8 *Let $\{\leq_w : w \in W\}$ be the family plausibility orderings induced by some complete, centered event model EM. Then*

(a) Each relation $\leq_w$ is a faithful preorder over W.

(b) The change operation determined by $\{\leq_w : w \in W\}$ is an update operator.

(c) The update operator determined by $\{\leq_w : w \in W\}$ is equivalent to the explanation-change operator $\diamond_{EM}$.

If we have an event model where each event ordering is a total preorder, then the induced plausibility orderings over worlds are also preorders.

Proposition 9 *Let $EM = \langle W, E, \preceq \rangle$ be an event model such that $\preceq_w$ is a total preorder for each $w \in W$. Then each plausibility ordering $\leq_w$ induced by EM is a total preorder.*

Since such a circumstance may arise rather frequently, the properties of such *total update operators* are of interest. We can extend the Katsuno-Mendelzon representation theorem to deal with update operators of this type. The required postulate embodies a variant of the principle of rational monotonicity, cited widely in connection with nonmonotonic systems of inference and conditional logics (see, e.g., (Boutilier 1994a)).

(U9) If KB is complete, $(KB \diamond A) \not\models \neg B$ and $(KB \diamond A) \models C$ then $(KB \diamond (A \wedge B)) \models C$ then

Theorem 10 *An update operator $\diamond$ satisfies postulates (U1) through (U9) iff there exists an appropriate family of faithful* total *preorders $\{\leq_w : w \in W\}$ that induces $\diamond$ (in the usual way).*

As a final remark, we note that the converses of Theorems 7 and 8 are trivially and uninterestingly true. For any update operator $\diamond$, one can construct an appropriate set of events (and orderings) that will induce that operator. This not of interest, since the point of explanation-change is to provide a natural view of update, characterizable in terms of the events of an existing domain. The ability to construct such events to capture a particular update operator provides little insight into update. The appropriate perspective is to reject any update operator (in a given domain) that cannot be induced by the existing set of events (or event model).

4 Concluding Remarks

We have provided an abductive model for incorporating into an existing belief set observations that arise through the evolution of the world. While our model allows more general forms of change than KM-update, we can impose restrictions on our model to recover precisely the KM theory. However, these restrictions are inappropriate in many cases, calling into question the suitability of some of the update postulates.

Of particular concern, as emphasized earlier, is postulate (U3). This embodies the assumption that all observations are explainable in terms of some event. This is not always reasonable. For instance, in our database example we might have a transaction to update Fred's salary to \$90,000 when there is a salary cap of \$80,000 in Finance. Thus, no event could have caused such an occurrence if Fred is indeed in Finance. Far from being a miraculous occurrence, it suggests that Fred in actually in Purchasing. Thus the observation not only forces KB to be updated (reflecting a change in the world), but also revised (reflecting additional knowledge about the world.

Note that this is not an artifact of out definition of update, where we insist that an observation be explainable for *every* $w \in ||KB||$. One might argue that we should simply update those worlds for which explanations exist and ignore the others. This is reasonable, but it is no longer simply update; rather it is a combination of update and revision. Furthermore observations may often be unexplainable for every world in $||KB||$. For instance, suppose a solution is believed to be an acid and a litmus strip is dipped into it, which promptly turns blue. This is not explainable for any KB-world (it should turn red) in terms of event effects. Instead, the intuitive explanation (the solution is a base) requires that KB be revised before adopting the update observation. Finally notice that an observation need not be strictly unexplainable to force revision. Often an implausible explanation will suffice. For instance, a raise to \$90,000 might not be impossible in Finance, but just so implausible that the database is willing to accept the fact that Fred is in Purchasing.

Issues of this sort make postulate (U3) (and certain aspects of (U5) and (U8)) somewhat questionable, and provides further motivation for adopting an abductive view of update. This perspective is especially fruitful when combining the process of update (changing knowledge) with belief revision (gaining knowledge). A model that puts both components together in a broader abductive framework is described in (Boutilier 1993; Boutilier 1994b). Roughly, the logic for belief revision set forth in (Boutilier 1994c) is used to capture the revision process, but is combined with elements of dynamic logic (Harel 1984) to capture the evolution of the world due to action occurrences.

Other have presented models of update that, like ours and unlike the KM-model, have their basis in reasoning about action. del Val and Shoham (1992; 1993), using the situation calculus, show how one can determine an update operator by reasoning about the changes induced by a given action. Very roughly, when some KB is to be updated by an observation O, they postulate the existence of some action A_O^{KB} whose predicted effects, when applied to the "situation" embodied by KB, determine the form of the update operator. Most critically, the effect axiom for such an action states that O holds when A_O^{KB} is applied to KB, and other effects are inferred via persistence mechanisms.

This model differs from ours in a number of rather important ways. First, del Val and Shoham assume that the update formula O describes the occurrence of some action or event. This severely restricts the scope of update, which in general can accept arbitrary propositions. They provide no mechanism for explaining an observation using the specification of *existing* actions. In order to deal with arbitrary observations an action is "invented" for the purpose of causing any observation in any situation. Naturally, the effects of such new actions are not specified *a priori* in the domain theory. So they propose that the effect of invented actions is to induce minimal change in the knowledge base according to some persistence mechanism. However, the plausible cause of an observation O may carry with it, in general, other drastic (rather than minimal) changes in KB. This can only be accounted for by explaining an observation in terms of existing actions. A persistence mechanism is required primarily because existing action or event specifications are not employed.

Another drawback of this model is its failure to account for the possibility that any of a number of actions might have caused O, and that update should reflect the most plausible of these causes. Finally, there is an assumption that the update of KB is due to the occurrence of a (known) single action. As we have described above, this will usually not be the case. Conditional explanations, explanations that use different actions for different "segments" of KB, will be very common.

A related mechanism is proposed by Goldszmidt and Pearl (1992), who use qualitative causal networks to represent an action theory. Again, update formula are implicitly assumed to be propositions asserting the occur-

rence of some action or event. An observation O is incorporated by assuming some proposition $do(O)$ has become true, and using a forced-action semantics to propagate its effects. Explanations are not given in terms of existing actions.

We should point out that both theories adopt a theory of action that provides a representation mechanism for actions and effects, as well as incorporating a solution to the frame problem (implicitly in the case of Goldszmidt and Pearl). We have side-stepped such issues by focusing on the semantics of update. We are currently investigating various action representations, such as STRIPS and the situation calculus, and the means they provide for generating conditional explanations. This is partially developed in (Boutilier 1993; Boutilier 1994b), where we provide a representation for actions using a conditional default logic to capture the defeasibility and nondeterminism of action effects, and use elements of dynamic logic to capture the evolution of the world. Action theories such as those exploited in (del Val and Shoham 1992; Goldszmidt and Pearl 1992) might also be used to greater advantage.

References

Alchourrón, C., Gärdenfors, P., and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530.

Boutilier, C. 1993. Explaining observations in reasoning about action. (manuscript).

Boutilier, C. 1994a. Conditional logics of normality: A modal approach. *Artificial Intelligence*. (in press).

Boutilier, C. 1994b. Two types of explanation in reasoning about action. Technical report, University of British Columbia, Vancouver. (Forthcoming).

Boutilier, C. 1994c. Unifying default reasoning and belief revision in a modal framework. *Artificial Intelligence*. (in press).

Dean, T., Kaelbling, L. P., Kirman, J., and Nicholson, A. 1993. Planning with deadlines in stochastic domains. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 574–579, Washington, D.C.

del Val, A. and Shoham, Y. 1992. Deriving properties of belief update from theories of action. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 584–589, San Jose.

del Val, A. and Shoham, Y. 1993. Deriving properties of belief update from theories of action (ii). In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 732–737, Chambery, FR.

Fikes, R. E. and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

Goldszmidt, M. and Pearl, J. 1992. Rank-based systems: A simple approach to belief revision, belief update, and reasoning about evidence and actions. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 661–672, Cambridge.

Harel, D. 1984. Dynamic logic. In Gabbay, D. and Guenthner, F., editors, *Handbook of Philosophical Logic*, pages 497–604. D. Reidel, Dordrecht.

Katsuno, H. and Mendelzon, A. O. 1991. On the difference between updating a knowledge database and revising it. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 387–394, Cambridge.

Kushmerick, N., Hanks, S., and Weld, D. 1993. An algorithm for probabilistic planning. Technical Report 93-06-04, University of Washington, Seattle.

McCarthy, J. and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502.

Morgenstern, L. and Stein, L. A. 1988. Why things go wrong: A formal theory of causal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 518–523, St. Paul.

Poole, D. 1988. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*, pages 359–380. Academic Press, San Diego.

Reiter, R. 1992. On specifying database updates. Technical Report KRR-TR-92-3, University of Toronto, Toronto.

Shanahan, M. 1993. Explanation in the situation calculus. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 160–165, Chambery, FR.

Winslett, M. 1988. Reasoning about action using a possible models approach. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 89–93, St. Paul.

Acknowledgements

Discussions with Ray Reiter have helped to clarify my initial thoughts on update. Thanks to Richard Dearden and David Poole for helpful discussions on this topic and to Alvaro del Val for well-considered comments on an earlier draft of this paper. This research was supported by NSERC Research Grant OGP0121843.

Can Situated Robots Play Soccer?

Michael K. Sahota and Alan K. Mackworth
Laboratory for Computational Intelligence
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada, V6T 1Z4
sahota@cs.ubc.ca, mack@cs.ubc.ca

Abstract

The goal of creating an integrated cognitive robot is still only a tantalizing dream. Current artificial intelligence and robotics research is highly divergent with little or no commonality among specialized subfields. New rich task domains are needed to pose the right challenges to extant theories and promote convergence. We propose soccer-playing as such a task since it requires situated robotics, perception, real-time decision making, planning, plan recognition, learning and multi-robot coordination and control. The technology to perform real-time vision and build autonomous robots is available; the Dynamite testbed has been built to perform experiments with multiple robots. A soccer tournament has been carried out using the testbed to evaluate aspects of the proposed reactive deliberation robot architecture. The results raise new issues and problems for research on robotic agents operating in dynamic environments.

1 Introduction

One of the dreams of Artificial Intelligence is the construction of integrated cognitive robots. Such robots must be able to integrate perception, reasoning, and action. These robots should be able to operate in the real world, which is dynamic and uncertain, not just in highly restricted environments such as factories. If building real robots is still part of the dream of AI, then we need to develop tools and theories to accomplish this goal.

Unfortunately, current research in AI is highly divergent with little or no overlap between specialized subfields such as computational vision, knowledge representation, robotics, and learning. Each group has its own conferences and journals, and when they do all meet at a single conference, they diverge in parallel sessions. The version of divide-and-conquer that we have been playing, namely, functional decomposition, is not now the best strategy.

For significant progress to be made on the AI dream, researchers must work on common tasks. But which tasks? It is clear that any science must close its eyes to most of the allures and mysteries of nature and choose a highly circumscribed fragment of reality to examine. Indeed, the key experimental task domain may well be an abstraction of the world; but we must take care to preserve the key problems and not abstract them away. For example, Galileo chose, as his blocks world, bodies sliding on a friction-free inclined plane in a vacuum; Newton considered point masses of infinite density. The danger in selecting a problem domain is that researchers must steer a course between the Scylla of enunciating a vacuous general theory of an artificial world and the Charybdis of implementing a collection of quick and dirty hacks that work, after a fashion, on an overly complex domain not properly abstracted, delimited or understood.

There have been a number of task domains that have served to focus AI research since its inception. Chess, the blocks world, video games, Tweety, the Yale Shooting Problem and many others have all served to motivate and focus the efforts of communities of researchers. We should realize that the choice of task domain is a theory-laden decision; that decision should be taken explicitly by the research community.

The Good Old Fashioned AI and Robotics (GOFAIR) [Mackworth, 1993] research paradigm has shaped the area of robotics since the time of the robot Shakey [Nilsson, 1984]. Some of the fundamental assumptions made of the world were that there is only one agent, that the environment is static unless the agent changes it, that actions are discrete and are carried out sequentially and that the world the robot inhabits can be accurately and exhaustively modeled by the robot. These assumptions proved to be overly restrictive and ultimately sterile. In the usual dynamic of the scientific dialectic, a new movement has emerged as the antithesis to GOFAIR: Situated or Nouvelle AI, which we will call the Situated Agent approach.

The Situated Agent paradigm is loosely characterized by the guiding principles set forth by Brooks: situatedness, embodiment, intelligence and emergence [Brooks, 1991]. The key idea of situatedness and embodiment is that researchers in AI should consider embodied agents that are connected to a larger world that provides the context for their activity. The essence of intelligence and emergence is that the intelligence of an agent can be judged by the qual-

ity of its interaction with its environment. The motivation for these principles is to direct research toward more realistic tasks and architectures and away from the Scylla of ungrounded theories.

A paradigmatic domain is needed to test and develop the competing GOFAIR and Situated Agent approaches. It must be suitable for testing extant theories and be sufficiently rich to bring the many threads in AI back together.

2 Why Soccer as a Task Domain?

We propose that playing soccer be a paradigmatic task domain since it breaks with nearly all of the restrictive assumptions on which GOFAIR is based and meets the standards proposed in the Situated Agent approach. The soccer domain can be characterized by the following:

- ☐ Neutral, friendly, and hostile agents
- ☐ Interagent cooperation
- ☐ Real-time interaction
- ☐ Dynamic environment
- ☐ Real and unpredictable world
- ☐ Objective performance criteria
- ☐ Repeatable experiments

The GOFAIR assumptions do not hold in the soccer world. The one agent assumption is violated: there are cooperating agents on the robot's team, competing agents on the other team, and neutral agents such as the referee and the weather. The world is not completely predictable: it is not possible to predict precisely where the ball will go when it is kicked, even if all the relevant factors are known. The simplifying assumption of discrete sequential actions is violated: continuous events such as a player running to a position and the ball moving through the air occur concurrently.

Soccer meets the standards of the Situated Agent approach. In soccer, robot agents are embodied and are situated in an unfolding game. Although it is still true that the intelligence of an agent can be judged from the dynamics of interaction with the environment, soccer also provides *objective performance criteria*.

The ability to score and prevent goals and the overall score of the game are objective measures of success. These measures allow explicit comparisons of alternative controller designs. The effects of chance can be factored out by carrying out repeated experiments. With objective criteria and repeatability, short-term and long-term learning strategies, as well as experiments in automatic evolution of controllers, become feasible. The availability of objective criteria is a critical feature of soccer that distinguishes it, along with the aspect of a real and unpredictable environment, from many of the other task domains proposed for driving the new research paradigm.

Soccer as a task domain is sufficiently rich to support research integrated from many branches of AI. In addition to the obvious potential of the soccer domain for research in perception and robotics, there are many other areas of AI that are applicable: reasoning under uncertainty, on-line reasoning, resource-bounded reasoning, planning, decision theory, qualitative physics, plan recognition, learning, and multi-agent theory.

Soccer is not the real world, but a suitably circumscribed fragment of it. Soccer is an appropriate abstraction of the world to challenge research in AI to focus on achievable tasks, and to drive the development of relevant theories.

3 Dynamite: A Testbed for Multiple Mobile Robots

The Dynamite testbed provides a practical platform for testing theories in the soccer domain using multiple mobile robots. The testbed consists of a fleet of radio controlled vehicles that perceive the world through a shared perceptual system [Barman *et al.*, 1993]. In an integrated environment with dataflow and MIMD computers, vision programs can monitor the position and orientation of each robot while planning and control programs can generate and send out motor commands. This approach allows umbilical-free behaviour and very rapid, lightweight fully autonomous robots.

The mobile robot bases are commercially available radio controlled vehicles. We have two controllable 1/24 scale racing-cars, each 22 cm long, 8 cm wide, and 4 cm high excluding the antenna. The testbed (244 cm by 122 cm in size) with two cars and a ball is shown in Figure 1. The cars have each been fitted with two circular colour markers to allow the vision system to identify their position and orientation. The ball is the small object between the cars.

The hardware used in this system is shown in Figure 2. There is a single colour camera mounted in a fixed position above the soccer field. The video output of the camera is transmitted to special-purpose video processing DataCube hardware in Figure 2. The DataCube is a dataflow computer which has been programmed to classify image pixels into different colour classes at video rate (60 Hz). This information is transmitted to a network of transputers which form a MIMD computer. Additional vision processing is performed on the transputers to find the position, in screen coordinates, of the centroid of each coloured blob and to transform these positions from screen to world coordinates. The vision subsystem is called the Vision Engine [Little *et al.*, 1991]. The Vision Engine produces the absolute position of all the objects on the soccer field; the orientation of each car is also reported. This is done at 60 Hz with an accuracy in position of approximately 1 mm.

The reasoning and control components of a vehicle can be implemented on any number of transputers out of the available pool. Currently, each vehicle is controlled by a distributed user program running on two transputer nodes. An arbitrary number of nodes, labeled 1 to n in Figure 2, can be used in parallel to control independent vehicles. The movement of all vehicles is controlled through radio transmitters attached to a single shared transputer node. Commands are transmitted to the vehicles at a rate of 60 Hz.

Figure 1 Robot Players on the Soccer Field

A physics-based real-time graphics simulator for the Dynamite world is also available for testing and developing reasoning and control programs.

A feature of the Dynamite testbed is that it is based on the "remote brain" approach to robotics. The testbed avoids the technical complexity of configuring and updating on-board hardware and makes fundamental problems in robotics and artificial intelligence more accessible. We have elected not to get on-board the on-board computation bandwagon, since the remote (but untethered) brain approach allows us to focus on scientific research without devoting resources to engineering compact electronics.

4 A Robot Architecture for Dynamic Domains

Most extant theories of robot architectures do not directly address the problems posed by dynamic environments. In a changing world, an agent must be able to generate intelligent behaviour in real-time. The soccer domain is a good testing ground for theories that address these issues since it is a highly dynamic environment. In this section, an architecture targeted towards dynamic environments, reactive deliberation, is described.

Much of the previous work on architectures for dynamic environments has been addressed by two distinct schools. Architectures in the situated behaviour school [Brooks, 1986; Agre and Chapman, 1987; Kaelbling and Rosenschein, 1990] typically allow frequent changes in the actions of the robot, yet restrict the allowable computational models. The planning school [Nilsson, 1984; Firby, 1992; Gat, 1992] allows unrestricted computational models, yet the commitment to arbitrary length plans hinders the ability of the agent to change its goals and actions in response to unanticipated changes in the environment.

The problem of deciding what to do next has also been addressed in decision theory [Kanazawa and Dean, 1989],

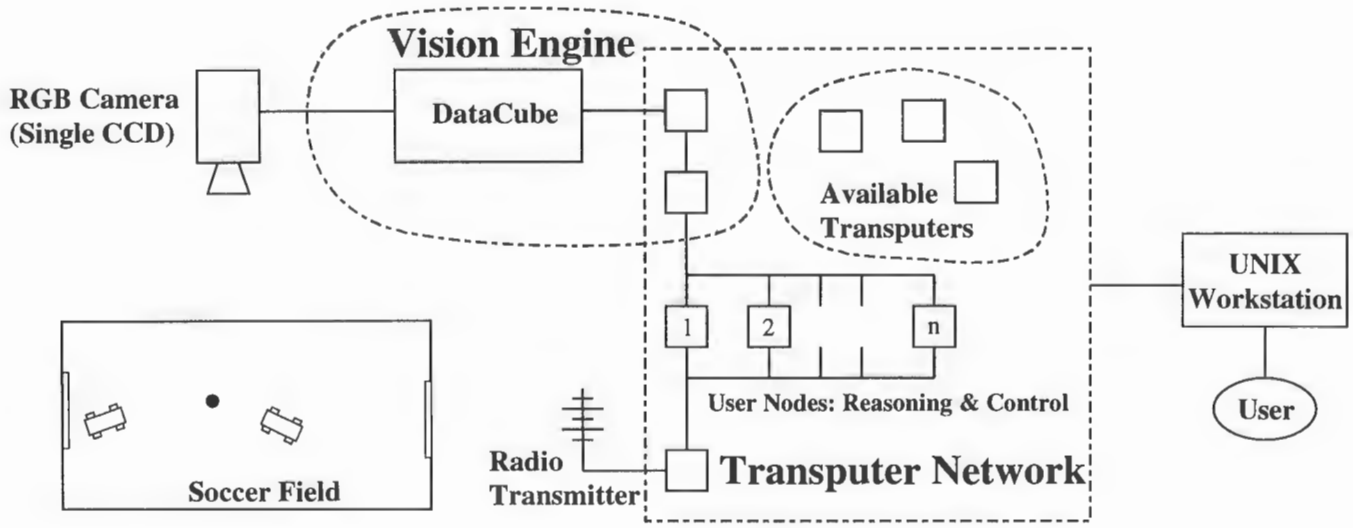


Figure 2 The Dynamite Hardware Setup

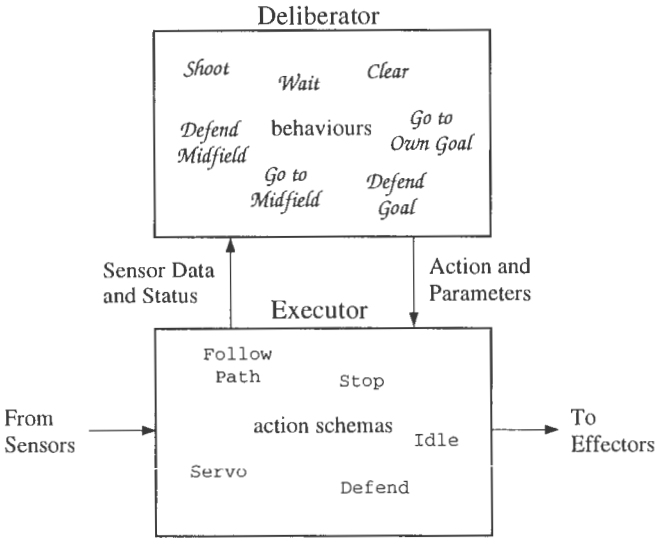


Figure 3 The Reactive Deliberation Controller

Maes' dynamics of action selection [Maes, 1990], and Minsky's mental proto-specialists [Minsky, 1986]. Decision theoretic tools are limited in their ability to handle continuous variables and perform sophisticated spatial reasoning. The dynamics of action selection performed poorly in simulations [Tyrrell, 1993] in part due to a reliance of the model on predicate inputs. Minsky's arguments against mental proto-specialists (that bid against one another for control of the agent) neglect to include the external state of the world as a valid basis for decisions.

Reactive deliberation is a robot architecture that combines responsiveness to the environment with intelligent decision making [Sahota, 1993; Sahota, 1994]. Even deliberation must be to some extent be reactive to respond to changes in the environment. Although the name is apparently an oxymoron, it is consistent with Artificial Intelligence nomenclature (cf. Reactive Planning).

Under reactive deliberation, the robot controller is partitioned into a deliberator and an executor; the distinction is primarily based on the different time scales of interaction. Informally, the deliberator decides what to do and how to do it, while the executor interacts with the environment in real-time. These components run asynchronously to allow the executor to interact continuously with the world and the deliberator to perform time consuming computations. A structural model illustrating the partition with examples can be seen in Figure 3. The deliberator is responsible for generating a single action, whereas other planning-based architectures generate a complete plan (i.e. sequences of actions). This distinction helps focus the deliberative activities on the immediate situation.

The executor is composed of a collection of action schemas. An *action schema* is a robot program that interacts with the environment in real-time to accomplish specific actions. Only one action schema is enabled at a time and it interacts with the environment through a tight feedback loop. The active schema receives run-time parameters from the deliberator that fully define its activity.

The focus of the deliberator is on an effective mechanism for selecting actions or goals in a timely manner. A central feature of reactive deliberation is that the deliberator is composed of concurrently active modules called *behaviours* that represent the goals of the robot. The notion of a behaviour is used in the sense of Minsky's mental proto-specialists [Minsky, 1986] with some important distinctions. In reactive deliberation, each behaviour computes an action and generates a bid reflecting how suitable it is in the current situation. The most appropriate behaviour, and hence action, is determined in a distributed manner through inter-behaviour bidding. Some examples of behaviours are: shoot ball, defend goal, go to midfield, clean floor, and deliver mail.

A *behaviour* is a robot program that computes an action that may, if executed, bring about a specific goal. Behaviours propose actions whereas action schemas perform actions. Each behaviour must perform the following: 1) select an action schema, 2) compute run-time parameters for the schema (plan the action), and 3) generate a bid describing how appropriate the action is.

Behaviours in reactive deliberation have a number of features. Different computational models can be used within behaviours to provide flexibility in the design of robot controllers. Inter-behaviour bidding is an effective mechanism for goal arbitration [Tyrrell, 1993] and can also be accomplished in a distributed computing environment. Another important property is that behaviours can be used as a mechanism for distributing computational resources.

Reactive deliberation is not a panacea for robotic architectural woes. A further disclaimer is that it is an incomplete robot architecture since it focuses on the issues related to dynamic domains and ignores a number of issues such as perceptual processing and the development of world models. The proposal is orthogonal to those issues. However, it makes explicit the need to evaluate the actions and goals of the robot at a rate commensurate with changes in the environment.

5 Some Experimental Results

Several controllers based on reactive deliberation have been implemented to allow robots to compete in complete one-on-one games of soccer [Sahota, 1993]. Current functionality includes various simple offensive and defensive strategies, motion planning, ball shooting and playing goal. The robots can drive under accurate control at speeds up to 1 m/s, while simultaneously considering alternate actions. We have produced a 10 minute video that documents these features.

As documented in [Sahota, 1993], a series of experiments, soccer games, called the Laboratory for Computational Intelligence (LCI) Cup were performed using the Dynamite testbed. The most elaborated reactive deliberation controller competed with subsets of itself to provide,

through the scores of the games, an objective utility measure for some of the architectural features of reactive deliberation and the behaviour themselves. Through the results of the LCI Cup the importance of modifying goals in response to changes in the environment has been shown. Further, the results demonstrate that the architectural elements in reactive deliberation are sufficient for real-time intelligent control in dynamic environments.

The reactive deliberation architecture provides a first step towards an integrated intelligent agent for dynamic environments. The current version of the controller can only play adequately in one-on-one soccer. Even in this restricted task domain, there are many unresolved problems. There are several important issues that need to be further addressed in building robot agents, such as:

- Real-time decision making — Reasoning about the world and selecting appropriate actions in real-time.
- Planning — Efficiently computing motion plans, predicting future world states, and reasoning about actions in an uncertain world.
- Plan recognition — Identifying the goals, actions and plans of other agents.
- Modeling — Acquiring implicit or explicit models of the robot and the environment.
- Learning — Changing behaviour at many levels through tuning models and refining actions using objective performance criteria.
- Multi-agent theory — Determining how agents can cooperate to accomplish group tasks.
- Robot architectures — Integrating all of the above components in new organizational forms.

We have shown that the Dynamite testbed is a useful abstraction of the soccer domain that can be used to test and develop many theories. However, it has a significant limitation. Off-board perception through an overhead camera leads to the pervasive use of world coordinates. The convenience of using a world model bypasses many important issues in robot vision and sensory robotics. For soccer experiments to address these issues in situated perception, a new testbed with on-board sensing will have to be developed.

6 Conclusions

Soccer has been proposed as a task for the development and unification of divergent theories in Artificial Intelligence. Soccer captures a number of essential properties of the real world including dynamics, real-time requirements, and cognitive functions. To perform experiments with soccer, the Dynamite testbed has been constructed with support for multiple mobile robots. A theory of robot architecture, reactive deliberation, has been applied to the soccer domain using the Dynamite testbed with demonstrated success. The results suggest that a wide range of theories from decision theory to robot control need further development to be successful in domains like this. This paper can be viewed as a challenge to researchers to apply their theories to the soccer domain to determine whose team of agents will win the Robot Soccer World Cup.

The question posed in the title, "Can Situated Robots Play Soccer?" has at least four possible answers: "Yes", "No", "Don't Know", and "Don't Care". We claim to have provided evidence for "Yes". But, one could argue for "No" based on the limitations of our experiments or our theories. "Don't Know" now seems inappropriate. "Don't Care" is a response that ignores the current theoretical and experimental needs of the field. Not only *can* situated robots play soccer but they also *should*!

Acknowledgments

We are grateful to Rod Barman, Keiji Kanazawa, Stewart Kingdon, Jim Little, Dinesh Pai, Heath Wilkinson and Ying Zhang for help with this. This work is supported, in part, by the Canadian Institute for Advanced Research, the Natural Sciences and Engineering Research Council of Canada and the Institute for Robotics and Intelligent Systems Network of Centres of Excellence.

References

[Agre and Chapman, 1987] Philip Agre and David Chapman. Pengi: An implementation of a theory of activity. In *AAAI-87*, pages 268–272, 1987.

[Barman *et al.*, 1993] R. Barman, S. Kingdon, J. Little, A. K. Mackworth, D.K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang. Dynamo: real-time experiments with multpile mobile robots. In *Proceedings of Intelligent Vehicles Symposium*, pages 261–266, 1993.

[Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

[Brooks, 1991] Rodney A. Brooks. Intelligence without reason. In *IJCAI-91*, pages 569–595, 1991.

[Firby, 1992] R. James Firby. Building symbolic primitives with continuous control routines. In *First International Conference on Artificial Intelligence Planning Systems*, pages 62–69, 1992.

[Gat, 1992] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *AAAI-92*, pages 809–815, 1992.

[Kaelbling and Rosenschein, 1990] Leslie Pack Kaelbling and Stanley J Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 35–48. M.I.T. Press, 1990.

[Kanazawa and Dean, 1989] Keiji Kanazawa and Thomas Dean. A model for projection and action. In *IJCAI-89*, pages 49–54, 1989.

[Little *et al.*, 1991] J. Little, R. Barman, S. Kingdon, and J. Lu. Computational architectures for responsive vision: the

vision engine. In *Proceedings of Computer Architectures for Machine Perception*, pages 233–240, 1991. Paris.

[Mackworth, 1993] Alan Mackworth. On seeing robots. In A. Basu and X. Li, editors, *Computer Vision: Systems, Theory, and Applications*, pages 1–13. World Scientific Press, 1993.

[Maes, 1990] Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 49–70. M.I.T. Press, 1990.

[Minsky, 1986] Marvin Minsky. *The Society of Mind*. Simon & Schuster Inc., 1986.

[Nilsson, 1984] Nils Nilsson. Shakey the robot. Technical Report 323, SRI International, 1984. Collection of Earlier Technical Reports.

[Sahota, 1993] Michael K. Sahota. Real-time intelligent behaviour in dynamic environments: Soccer-playing robots. Master's thesis, University of British Columbia, 1993.

[Sahota, 1994] Michael K. Sahota. Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. In *Proceedings of AAAI-94*, 1994. Forthcoming.

[Tyrrell, 1993] Toby Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, Edinburgh University, 1993.

Will The Robot Do The Right Thing?

Ying Zhang and Alan K. Mackworth*
Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada v6T 1Z4

Abstract

Robots are generally composed of multiple sensors, actuators and electromechanical parts. The overall behavior of a robot is emergent from coordination among its various parts and its interaction with its environment. Designing a 'correct' robot which does 'the right thing' in a given environment is an important and challenging problem. The question posed in the title is decomposed into two questions. First, what is the right thing? Second, how does one guarantee the robot will do it? We answer these questions in this paper by establishing a formal approach to the design and analysis of robotic systems and behaviors.

1 Motivation and Introduction

Building control systems for intelligent, reliable, robust and safe autonomous robots working in complex environments is an increasingly important challenge for research in electrical and mechanical engineering, and computer science.

Robots are generally composed of multiple sensors, actuators and electromechanical parts. Robots should be reactive as well as purposive systems, closely coupled with their environments; they must deal with inconsistent, incomplete and delayed information from various sources. Such systems are usually complex, hierarchical and physically distributed. Each component functions according to its own dynamics. The overall behavior of a system is emergent from coordination among its various parts and its interaction with its environment. We call the integration of a robot and its environment a *robotic system*, and the relation on the state of a robot and its environment over time the *robotic behavior*.

The current trend for developing intelligent robots is to combine AI techniques with traditional control theory [Schoppers, 1991]. However, most of this work is *ad hoc*; there is no well defined interface between the higher level (AI) and the lower level (control). The coordination between these levels is not fully understood, and the behavior of the whole system cannot be analyzed. One fundamental problem is the mismatch of the underlying computational models. AI is based on off-line computational models and control is based on on-line computational models. (The distinction between off-line and on-line models is analogous to the distinction between functions and processes.)

In this paper, we advocate a formal approach to modeling a robotic system. We have developed a formal model, Constraint Nets (CN), for general dynamic systems [Zhang and Mackworth, 1994a]. CN is an abstraction of general dynamic systems so that a system with discrete as well as continuous time, and asynchronous as well as synchronous event structures can be modeled in a unitary framework. Using aggregation operators, a system can be modeled hierarchically in CN; therefore, the dynamics of the environment as well as the dynamics of the robot can be modeled individually and then integrated. Based on abstract algebra and topology, CN supports multiple levels of abstraction, so that a system can be analyzed at different levels of detail. With a rigorous formalization, CN provides a programming semantics for the design of robot control systems.

We believe that the intelligence of an agent should be judged by the quality of the agent's interaction with the environment [Brooks, 1991]. The intelligence of an agent is measured by its ability to accomplish difficult tasks in complex, hazardous or uncertain environments. However, because there is, as yet, no rigorous definition for intelligent behaviors, we shall use the concept of *desired behaviors*.

In this paper, we advocate a formal approach to specifying desired behaviors and to verifying the relationship between a dynamic system and its behavior specification. Since robotic behaviors are the relationships between robots and their environments *over time*, the specification language should at least be able to represent temporal behaviors: states of a system over time. Various forms of temporal logics [Emerson, 1990] have been proposed in both the systems [Manna and Pnueli, 1992; Lamport, 1991; Ostroff, 1989; Alur and Henzinger, 1989] and AI [Allen, 1990; Shoham, 1988; McDermott, 1990; Rosenschein, 1985] communities. We have adopted an automaton-based specification language, called $\forall$-automata [Manna and Pnueli, 1987], which is capable of representing a large class of temporal properties

*Shell Canada Fellow, Canadian Institute for Advanced Research

such as safety, liveness (recurrence, persistence, stability or controllability), goal achievement (reachability) and bounded response. Furthermore, a system modeled by a constraint net can be verified against its desired behavior specification by a general verification method.

The rest of the paper is organized as follows. Section 2 depicts the structure of robotic systems and the specification of robotic behaviors, illustrated by two running examples: a hand coordinator and a maze traveler. Section 3 gives the formal model for robotic systems, the Constraint Net model, and demonstrates constraint net modeling via the examples. Section 4 presents the formal specification language and its relationship with the Constraint Net model. Section 5 describes the formal verification method. Section 6 concludes the paper and points out some related research.

2 Robotic Systems and Behaviors

From a systemic point of view, a robotic system is a coupling of a robot to its environment, while the robot is an integration of a plant and its controller (Fig. 1). Basically, the roles of these three subsystems can be characterized as follows:

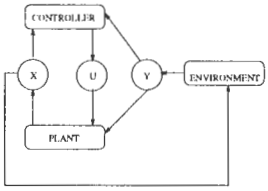


Figure 1: A robotic system

- *Plant*: a plant is a set of entities which must be controlled to achieve certain behaviors. For example, a robot arm with multiple joints, a car with throttle and steering, an airplane or a nuclear power plant can be considered as the *plant* of a robotic system.
- *Controller*: a controller is a set of sensors and actuators together with software/hardware computational systems which sense the states of the plant (X) and the environment (Y), and compute desired inputs (U) to actuate the plant. For example, an analog circuit, a program in a digital computer, various motors and sensors can be considered as parts of the *controller* of a robotic system.
- *Environment*: an environment is a set of entities beyond the control of the controller, with which the plant may interact. For example, obstacles to be avoided, objects to be reached, and rough terrain to be traversed can be considered as the *environment* of a robotic system.

We introduce two running examples to illustrate the general structure of robotic systems.

Example 2.1 The Hand Coordinator: Suppose a two-handed robot is required to fit caps on jars on an automated assembly line. The robot must pick up a jar and hold it with one hand and then fit a cap on the jar with its other hand. However, the hands work asynchronously at their own speed; for example, jars or caps may occasionally be unavailable, but we can assume that the acts of jar picking and cap fitting take some constant time. We will design a hand coordinator so that the right hand will cap only if the left hand is holding a jar; the left hand will put down the jar and pick up a new one only if the right hand has done the capping.

The robotic system, shown in Fig. 2, consists of the hand coordinator, and the left and right hands.

Figure 2: The hand coordinator system

The whole system should work as follows. Whenever there are more jars available, the left hand will request permission ($R1$) to pick up a jar, and the coordinator will grant the request ($C1$) if the previous jar has been capped. On the other hand, whenever there are more caps available, the right hand will request permission ($R2$) to cap a jar, and the coordinator will grant the request ($C2$) if the left hand is holding a jar in place.

Example 2.2 The Maze Traveler: Consider a maze composed of separated $\top$-shaped obstacles of bounded size placed in one of four orientations on an unbounded plane. A simple robot (Fig. 3 (a)) is required to traverse the maze from west to east (Fig. 3 (b)).

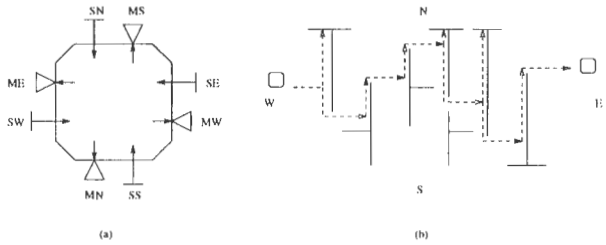


Figure 3: (a) A simple robot (b) A simple maze

In this example, the plant is the body of the robot which can move in one of four directions; the environment is the maze; and the controller connects sensing signals to motor commands (Fig. 4). For example, when the north sensor SN is on, the robot is touching a wall directly to its north; when the east motor ME is on, the robot moves east, if it is not blocked.

While the model of a robotic system states how the system is composed and how the system works formally, the specification of a robotic behavior represents what should the robot do overall. For the hand coordinator, one desired behavior is that the acts of jar picking and cap fitting should interleave. For the maze traveler, one desired behavior is that the robot should move to the east persistently. Given a formal model of a designed system and a formal specification of a desired behavior,

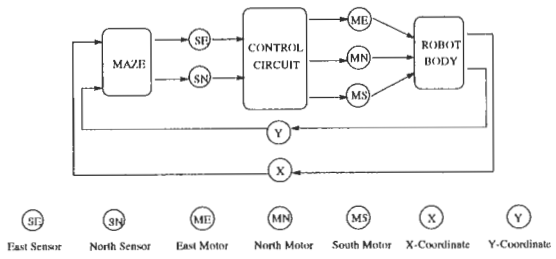


Figure 4: The maze traveler robotic system

one should be able to prove that the model does satisfy the specification, that is, that the robot will do the right thing.

Therefore, 'do the right thing' here does not necessarily mean that the robot has rationality built in [Russell and Wefald, 1991]; it simply means that the robot does what it is designed to do. Furthermore, the specification language we propose focuses only on temporal aspects; probability and stochastic analysis will be incorporated into this modeling framework in the future.

3 Model for Robotic Systems

In this section, we introduce the Constraint Net model and characterize its composite structure and modularity. The formal semantics of the model, based on the fixpoint theory of continuous algebras, has been presented in [Zhang and Mackworth, 1994a].

3.1 Dynamic systems

Since a robotic system is a dynamic system in general, we start with some basic concepts of dynamic systems. Let a *time structure* be a totally ordered set, which could be intervals of reals or ordered events, with a metric topology. Let a *domain* be a set of values, which could be numbers, symbols or strings. Both time and domains can be either continuous or discrete. In this paper, we mainly discuss discrete systems. However, all definitions here works for the general case.

- *Trace*: A trace $v : \mathcal{T} \rightarrow A$ is a function from a time structure $\mathcal{T}$ to a domain A. The set of traces is denoted by *trace space* $A^{\mathcal{T}}$. Traces can be transformed from one to another via *transductions*.
- *Transduction*: A transduction is a function from input traces to output traces which satisfies the causal relationship between its inputs and outputs, viz. the output values at any time are determined by the input values up to that time. Transductions can be considered as transformational processes. For example, a temporal integration is a typical transduction in continuous time; a state automaton defines a transduction in discrete time. Clearly, transductions are closed under functional composition. There are two basic types of transductions: *transliterations* and *delays*.
- *Transliteration*: A transliteration $f_{\mathcal{T}}$ is a pointwise extension of function f on time structure $\mathcal{T}$, that is, the output value at any time is the function of the input value at that time only. Let v be the input trace then we have $f_{\mathcal{T}}(v) = \lambda t.f(v(t))$. Intuitively, a transliteration is a transformational process without memory or internal state, for example, a combinational circuit. We use f to denote the transliteration $f_{\mathcal{T}}$ if no ambiguity arises.
- *Unit delay*: A unit delay $\delta(v_0)$ is a transduction defined mainly for discrete time structures, such that the output value at initial time 0 is v_0 and the rest of the output values are the input values at the previous time:

$$\delta(v_0)(v) = \lambda t. \begin{cases} v_0 & \text{if } t = 0 \\ v(pre(t)) & \text{otherwise} \end{cases}$$

where $pre(t)$ indicates the previous time point adjacent to t in the total order. The definition of $pre(t)$ can be generalized to arbitrary time structures [Zhang and Mackworth, 1994a]. A unit delay acts as a unit memory in discrete dynamic systems.

Any discrete dynamic system can be modeled using only transliterations and unit delays. A hybrid dynamic system, composed of both discrete and continuous components, can be modeled by event-driven transductions and transport delays, in addition to these two types of transductions [Zhang and Mackworth, 1994a].

3.2 Constraint nets

A dynamic system can be modeled by a constraint net. Influenced by some dataflow-like models [Ashcroft, 1986; Lavignon and Shoham, 1990; Benveniste and LeGuernic, 1990; Caspi *et al.*, 1987], the Constraint Net model is developed on an abstract dynamics structure, with abstract time and domains.

Intuitively, a constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections. A location can be regarded as a wire, a channel, a variable, or a memory location, whose value may change over time. Each transduction is a causal mapping from input traces to output traces. Connections relate locations with ports of transductions.

Syntactically, a *constraint net* is a triple $CN = \langle Lc, Td, Cn \rangle$, where Lc is a finite set of *locations*, each of which is associated with a domain; Td is a finite set of labels of *transductions*, each of which is associated with a set of *input ports* and an *output port*; Cn is a set of *connections* between locations and ports of transductions, with the following restrictions: (1) there is at most one output port connecting to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated.

A location l is an *output location* of a transduction F iff there is a connection between the output port of F and l; l is an *input location* of F iff there is a connection between an input port of F and l. A location is an *output* of the constraint net if it is an output location of a transduction otherwise it is an *input*. The set of input locations of a constraint net CN is denoted by $I(CN)$, the set of output locations is denoted by $O(CN)$. A constraint net is *open* if there is an input location otherwise it is *closed*.

A constraint net is represented by a bipartite graph where locations are depicted by circles, transductions are depicted by boxes and connections are depicted by arcs. We have seen examples of constraint nets in Fig. 1, Fig. 2 and Fig. 4.

Semantically, a constraint net is a set of equations, $\vec{o} = \vec{F}(\vec{i}, \vec{o})$, where each left-hand side is an individual output location and each right-hand side is an expression composed of transductions and locations. The semantics is defined as a solution of the set of equations [Zhang and Mackworth, 1994a]. If CN is a constraint net with time structure $\mathcal{T}$ and domain A_l for $l \in Lc$, the semantics is a transduction from input traces to output traces: $[\![CN]\!] : \times_{I(CN)} A_i^{\mathcal{T}} \rightarrow \times_{O(CN)} A_o^{\mathcal{T}}$ where $\times_L A_l^{\mathcal{T}}$ denotes the product of a family of trace spaces.

A constraint net can be hierarchically organized. A *module* is a triple $\langle CN, I, O \rangle$, denoted by $CN(I, O)$, where CN is a constraint net, $I \subseteq I(CN)$ and $O \subseteq O(CN)$ are subsets of the input and output locations of CN; $I \cup O$ defines the *interface* of the module. Locations $I(CN) - I$ and $O(CN) - O$ are called *hidden input* and *hidden output locations* respectively.

Graphically, a module is depicted by a box with rounded corners. If $[\![CN]\!]$ is the semantics of CN, the semantics of $CN(I, O)$ is the semantics of CN projected onto the interface, i.e. $[\![CN(I,O)]\!] = \Pi_{I \cup O}[\![CN]\!] = \{(\vec{i}, \vec{o}) | o = [\![CN]\!]_o(\vec{u}, \vec{i}), \vec{u} \in \times_U A_l^{\mathcal{T}}\}$ where $U = I(CN) - I$ is the set of hidden input locations. If $U \neq \emptyset$, i.e. $I \subset I(CN)$, $[\![CN(I,O)]\!]$ is a relation between input and output traces in general, rather than a function. Thus, while more powerful, and simpler, than most inherently nondeterministic models, nondeterminism can be modeled with hidden inputs, and probabilistic and stochastic analysis can be incorporated (if we provide random distributions on hidden inputs).

Generally speaking, modularity provides a kind of abstraction: hidden inputs capture nondeterminism and hidden outputs encapsulate internal state.

Furthermore, a complex module can be constructed from simple ones with aggregation operators [Zhang and Mackworth, 1994a], and the semantics has compositional properties.

Given a system modeled as a module $CN(I, O)$, the behavior of the system can be formally defined as a set of observable input/output traces $\{\vec{v} \in \times_{I \cup O} A_l^{\mathcal{T}} | \vec{v} \in [\![CN(I,O)]\!]\}$. We also use $[\![CN(I,O)]\!]$ to denote the behavior of the system.

So far, we have briefly presented the Constraint Net model (CN). CN is as powerful as the existent computational models, in either discrete sequential computations (Turing Machines) or continuous analog computations (smooth non-hypertranscendental functions [Shannon, 1941]) [Zhang, 1994]. CN is able to represent a hybrid system, consisting of a non-trivial mixture of discrete and continuous components. For the two running examples given in the previous section, we focus here only on their discrete models. Examples of hybrid system modeling can be found in [Zhang and Mackworth, 1993b].

In general, a robotic system is modeled as an integration of a plant module, a control module and an environment module (Fig. 1), and each of which may be further decomposed into a hierarchy of modules. The overall behavior of the system is not determined by any one of the modules, but emerges from the coupling of the interaction among all the components. Formally, the behavior of the system is the solution of the following equations:

$$\begin{aligned} X &= PLANT(U, Y), \\ U &= CONTROLLER(X, Y), \\ Y &= ENVIRONMENT(X). \end{aligned}$$

As we can see here, a robot, composed of a plant and a controller, is an open system in general, and a robotic system, with a robot coupled to its environment, is a closed system.

Now we illustrate the constraint net modeling using the two examples: the hand coordinator and the maze traveler.

Example 3.1 The Hand Coordinator: The hand coordinator can be designed using negated Muller C-elements [Sutherland, 1989], since the desired behavior of the hand coordinator is similar to a buffer synchronizer.

Consider the request and grant signals as events, transitions from 0 to 1 or 1 to 0. The Muller C-element acts as the 'and' for events: if both of its inputs have the same value, the output and its next state are copies of that value, otherwise the output and its next state are unchanged.

A Muller C-element can be modeled by a module $C(\{i_1, i_2\}, o)$ composed of a transliteration c and a unit delay:

$$o = c(i_1, i_2, q), q = \delta(0)(o)$$

where

$$c(i_1, i_2, q) = \begin{cases} i_1 & \text{if } i_1 = i_2 \\ q & \text{otherwise.} \end{cases}$$

We use the standard 'and' logic symbol with a 'C' inside it to represent Muller C-elements and 'bubbles' on input or output ports to represent inversions.

Assume that jar picking and cap fitting each take constant time. Two negated Muller C-elements and two delay elements are used to synchronize events in the controller (Fig. 5).

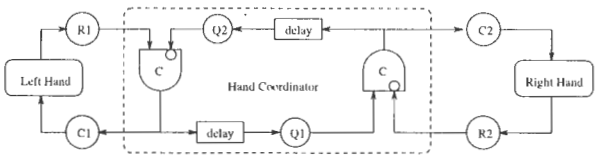


Figure 5: The hand coordinator

Example 3.2 The Maze Traveler: The controller for the maze traveler can be built using 'and' and 'not' gates with a flip-flop memory unit (Fig. 6).

The flip-flop $FF(\{i_1, i_2\}, o)$ is composed of a transliteration ff and a unit delay:

$$o = ff(i_1, i_2, q), q = \delta(0)(o)$$

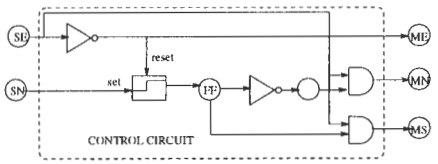


Figure 6: The controller for the maze traveler

where

$$ff(i_1, i_2, q) = \begin{cases} 1 & \text{if } i_1 = 1 \text{ (set)} \\ 0 & \text{else if } i_2 = 1 \text{ (reset)} \\ q & \text{otherwise.} \end{cases}$$

The control outputs to the robot body are: $ME = \neg SE, MN = \neg FF \wedge SE$ and $MS = FF \wedge SE$.

The robot body $BODY(\{ME, MN, MS\}, \{X, Y\})$ is composed of a transliteration f_{xy} and a unit delay:

$$\begin{aligned} \langle NX, NY \rangle &= f_{xy}(ME, MN, MS, X, Y), \\ \langle X, Y \rangle &= \delta(x_0, y_0)(NX, NY) \end{aligned}$$

where

$$f_{xy}(ME, MN, MS, X, Y) = \langle X + ME, Y + MN - MS \rangle,$$

i.e. the next $\langle X, Y \rangle$ position is displaced by a grid point depending on the motion commands.

Even though sensor and command domains are Boolean $\{0, 1\}$, notice that this robotic system is not finite, since the domain of both X and Y is the integers.

Will these robots do the right thing? We have given informal descriptions of what these two robotic systems are supposed to do in the previous section. Enthusiastic readers will check by hand that they seem to do the 'right thing'. However, even though a constraint net model gives a precise definition of what the behavior of the system is, it is infeasible to generate the behavior and check all the traces of the behavior. In the next two sections, we will present a behavior specification language for representing the desired behaviors of a system and a verification method for ensuring that the behavior of the system does satisfy its specification.

4 Specification for Robotic Behaviors

While modeling focuses on the underlying structure of a system, the organization and coordination of components or subsystems, the overall behavior of the modeled system is not explicitly expressed. However, for many situations, it is important to specify some global properties and guarantee that these properties hold in this design. For example, the acts of jar picking and cap fitting must interleave, the maze traveler should persistently move east. In this section, we present an automaton-based language for formally specifying robotic behaviors, and establish the relationship between a constraint net and a behavior specification.

A trace $v : \mathcal{T} \rightarrow A$ is a generalization of a sequence. In fact, when $\mathcal{T}$ is the set of natural numbers, v is an infinite sequence. A set of sequences defines a conventional formal language. If we take the abstract behavior of a system as a language, a specification can be represented as an automaton, and verification checks the inclusion relation between the language of the system and the language accepted by the automaton.

There is always a trade-off between the power of representation, i.e., the class of languages the type of automaton can accept, and the power of analysis, i.e. the computability of checking the acceptance of traces. We would like the type of automaton to be powerful enough to state certain temporal and real-time properties, yet simple enough to have formal, semi-automatic or automatic verifications. We have adopted $\forall$-automata [Manna and Pnueli, 1987] for our purpose.

$\forall$-automata are non-deterministic finite state automata over infinite sequences. These automata were proposed as a formalism for the specification and verification of temporal properties of concurrent programs. It has been shown that $\forall$-automata have the same expressive power as Buchi automata [Thomas, 1990] and the extended temporal logic (ETL) [Wolper, 1983], which are strictly more powerful than the linear propositional temporal logic [Thomas, 1990; Wolper, 1983]. More importantly, there is a formal verification method. We have been able to generalize $\forall$-automata for accepting general traces [Zhang and Mackworth, 1994b]. In this paper, we focus only on discrete systems and infinite sequences.

Let an *assertion* be a logical formula defined on states of a dynamic system, i.e. any assertion α on a given state s, denoted $\alpha(s)$, will be evaluated to either $true, s \models \alpha$, or $false, s \not\models \alpha$.

A *$\forall$-automaton* $\mathcal{A}$ is a quintuple $\langle Q, R, S, e, c \rangle$ where Q is a finite set of *automaton-states*, $R \subseteq Q$ is a set of *recurrent states* and $S \subseteq Q$ is a set of *stable states*. With each $q \in Q$, we associate an assertion $e(q)$, which characterizes the *entry condition* under which the automaton may start its activity in q. With each pair $q, q' \in Q$, we associate an assertion $c(q, q')$, which characterizes the *transition condition* under which the automaton may move from q to q'. R and S are the generalization of *accepting* states to the case of infinite inputs. We denote by $B = Q - (R \cup S)$ the set of *non-accepting (bad)* states.

For simplicity, let time be the set of natural numbers $\mathcal{N}$ and $v : \mathcal{N} \rightarrow A$ be a sequence. A *run* of $\mathcal{A}$ over v is a mapping $r : \mathcal{N} \rightarrow Q$ such that (1) $v(0) \models e(r(0))$, and (2) for all $n > 0$, $v(n) \models c(r(n-1), r(n))$.

A $\forall$-automaton is called *complete* iff the following requirements are met:

- $\bigvee_{q \in Q} e(q)$ is valid.
- For every $q \in Q$, $\bigvee_{q' \in Q} c(q, q')$ is valid.

These two requirements guarantee that any sequence has a run over it, and that any partial run can always be extended to an infinite sequence. We will restrict ourselves to complete automata. This is not a real restriction, since any automaton can be transformed to a complete automaton by introducing an additional error state $q_E \in B$, with the corresponding entry condition and transition conditions [Manna and Pnueli, 1987].

If r is a run, let $Inf(r)$ be the set of automaton-states appearing infinitely many times in r, that is $Inf(r) =$

$\{q|\forall n \exists m > n, r(m) = q\}$. A run r is defined to be *accepting* iff:

1. $Inf(r) \cap R \neq \emptyset$, i.e. *some* of the states appearing infinitely many times in r belong to R, or
2. $Inf(r) \subseteq S$, i.e. *all* the states appearing infinitely many times in r belong to S.

A $\forall$-automaton $\mathcal{A}$ *accepts* a sequence v, written $v \models \mathcal{A}$, iff *all* possible runs of $\mathcal{A}$ over v are accepting. A $\forall$-automaton $\mathcal{A}$ *accepts* a discrete time system $CN(I,O)$, written $CN(I,O) \models \mathcal{A}$, iff for all $v \in [\![CN(I,O)]\!]$, $v \models \mathcal{A}$.

One of the advantages of using automata as a specification language is the graphical representation. It is useful and illuminating to represent $\forall$-automata by diagrams. The basic conventions for such representations are the following:

- The automaton-states are depicted by nodes in a directed graph.
- Each initial state is marked by a small arrow, called the *entry arc*, pointing to it.
- Arcs, drawn as arrows, connect some of the states.
- Each recurrent state is depicted by a diamond shape inscribed within a circle.
- Each stable state is depicted by a square inscribed within a circle.

Nodes and arcs are labeled by assertions. A node or an arc that is left unlabeled is considered to be labeled with *true*. The labels define the entry conditions and the transition conditions of the associated automaton as follows:

- Let $q \in Q$ be a node in the diagram. If q is labeled by ψ and the entry arc is labeled by φ, the entry condition $e(q)$ is given by: $e(q) = \varphi \wedge \psi$. If there is no entry arc, $e(q) = false$.
- Let q, q' be two nodes in the diagram. If q' is labeled by ϕ, and arcs from q to q' are labeled by $\varphi_i, i = 1..n$, the transition condition $c(q,q')$ is given by: $c(q,q') = (\varphi_1 \vee \ldots \vee \varphi_n) \wedge \psi$. If there is no arc from q to q', $c(q,q') = false$.

A diagram representing an incomplete automaton is interpreted as a complete automaton by introducing an error state and associated entry and transition conditions.

This type of automaton is powerful enough to specify various qualitative behaviors. Some typical desired behaviors are shown in Fig. 7. Figure 7(a) accepts a sequence which satisfies $\neg G$ only finitely many times, Figure 7(b) accepts a sequence which never satisfies B, and Figure 7(c) accepts a sequence which will satisfy S in the finite future whenever it satisfies R.

Now we can formally specify the desired behaviors for the hand coordinator and the maze traveler.

Example 4.1 The Hand Coordinator: Let the acts of jar picking and cap fitting be controlled by the events at $C1$ and $C2$ respectively. Let $E(X)$ be an assertion denoting that there is an event at X. If $E(C1)$, the robot picks up a jar, and if $E(C2)$, the robot fits the cap.

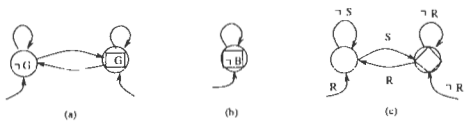


Figure 7: $\forall$-automata: (a) goal achievement or reachability (b) safety (c) bounded response

One desired behavior for the hand coordinator is that $E(C1)$ and $E(C2)$ must interleave and $E(C1)$ always precedes $E(C2)$. This behavior can be represented by a $\forall$-automaton in Fig. 8 (a).

Example 4.2 The Maze Traveler: Let ME be an assertion denoting that the east motor is on, or $ME = 1$. One desired behavior for the maze traveler is the *liveness* property represented by a $\forall$-automaton in Fig. 8 (b), meaning that the robot will persistently move east.

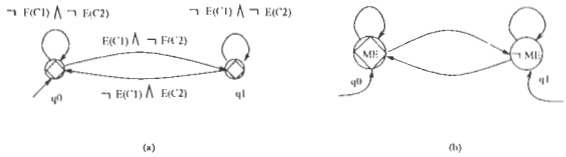


Figure 8: The specification of (a) the hand coordinator (b) the maze traveler

5 A Formal Verification Method

In this section, we present a verification method modified from [Manna and Pnueli, 1987] with concurrent programs replaced by discrete constraint nets.

Any discrete constraint net CN is composed of two types of transductions, transliterations and unit delays. Therefore CN can be represented by two sets of *domain equations*, each of the form $l_0' = l$, if l_0 is an output location of a unit delay from the input location l, or $l_0 = f(l_1, \ldots, l_n)$, if l_0 is an output location of a transliteration f from the input location tuple $\langle l_1, \ldots, l_n \rangle$.

The domain equations for the hand coordinator and the maze traveler are as follows.

Example 5.1 The Hand Coordinator: For simplicity, assume that the delays in the hand coordinator in Fig. 5 are unit delays. The set of equations for the hand coordinator is:

$$C1 = c(R1, \neg Q2, Q1), \;\; Q1' = C1,$$

$$C2 = c(Q1, \neg R2, Q2), \;\; Q2' = C2.$$

Example 5.2 The Maze Traveler: The set of equations for the maze traveler is:

$$FF = ff(SN, \neg SE, Q), Q' = FF,$$

$$ME = \neg SE, MN = \neg FF \wedge SE, MS = FF \wedge SE,$$

$$\langle X', Y' \rangle = f_{xy}(ME, MN, MS, X, Y)$$

A state s of CN is a mapping from locations to domains, i.e. $s \in \times_{Lc} A_l$ where $Lc = I(CN) \cup O(CN)$. A pair of states $\langle s, s' \rangle$ is said to be *consistent* with CN, denoted $CN(s, s')$, iff for every equation of the form $l_0 = f(l_1, \ldots, l_n)$, $s(l_0) = f(s(l_1), \ldots, s(l_n))$ and $s'(l_0) = f(s'(l_1), \ldots, s'(l_n))$, and for every equation of the form $l'_0 = l$, $s'(l_0) = s(l)$.

Let φ and ψ be assertions on states of a constraint net. We write $\{\varphi\}CN\{\psi\}$ to denote that the consecutive condition: $\varphi(s) \bigwedge CN(s, s') \rightarrow \psi(s')$ is valid.

Let Θ be an assertion indicating the initial state of CN, and $\mathcal{A} \equiv \langle Q, R, S, e, c \rangle$ be a $\forall$-automaton. A set of assertions $\{\alpha_q\}_{q \in Q}$ is called a set of *invariants* for CN and $\mathcal{A}$ iff

- *Initiality*: $\forall q \in Q. \Theta \wedge e(q) \rightarrow \alpha_q$.
- *Consecution*: $\forall q, q' \in Q. \{\alpha_q\}CN\{c(q, q') \rightarrow \alpha_{q'}\}$.

Given that $\{\alpha_q\}_{q \in Q}$ is a set of invariants for CN and $\mathcal{A}$ and W is a *well-founded set*, i.e. any decreasing sequence of W is finite, a set of partial functions $\{\rho_q\}_{q \in Q} : \times_{Lc} A_l \rightarrow W$ is called a set of *ranking* functions for CN and $\mathcal{A}$ iff the following conditions are satisfied:

- *Definedness*: $\forall q \in Q.\ \alpha_q \rightarrow \exists w. \rho_q = w$.
- *Non-increase*: $\forall q \in Q, q' \in S.$

$$\{\alpha_q \wedge \rho_q = w\}CN\{c(q, q') \rightarrow \rho_{q'} \leq w\}.$$

- *Decrease*: $\forall q \in Q, q' \in B.$

$$\{\alpha_q \wedge \rho_q = w\}CN\{c(q, q') \rightarrow \rho_{q'} < w.\}$$

We conclude that if the following requirements are satisfied the validity of a $\forall$-automaton $\mathcal{A}$ over a constraint net CN is proved:

(I) Associate with each automaton-state $q \in Q$ a state formula α_q, such that $\{\alpha_q\}_{q \in Q}$ is a set of invariants for CN and $\mathcal{A}$.

(R) Associate with each automaton-state $q \in Q$ a partial function $\rho_q : \times_{Lc} A_l \rightarrow W$, such that $\{\rho_q\}_{q \in Q}$ is a set of ranking functions for CN and $\mathcal{A}$.

As in [Manna and Pnueli, 1987], the verification rules (I) and (R) are sound and complete for a complete $\forall$-automaton $\mathcal{A}$ and a discrete constraint net CN, i.e. $\mathcal{A}$ accepts CN iff there exist a set of invariants and ranking functions.

Now we can prove that the hand coordinator in Fig. 5 does satisfy its specification in Fig. 8(a), and the maze traveler controlled by the control circuit in Fig. 6 does satisfy its specification in Fig. 8(b).

Example 5.3 The Hand Coordinator: The $\forall$-automaton in Fig. 8(a) is not complete. To make it complete, we add another automaton-state $q_E \in B$ with $e(q_E) = false$, $c(q_0, q_E) = E(C2)$, $c(q_1, q_E) = E(C1)$, $c(q_E, q_i) = false$ for $i = 0, 1$, and $c(q_E, q_E) = true$.

Let $E(C1)$ be $C1 \neq Q1$ and $E(C2)$ be $C2 \neq Q2$. Let the initial condition Θ be $C1 = C2$. It is easy to check that $C1 = C2$, $C1 \neq C2$ and $false$ are invariants for q_0, q_1 and q_E respectively. Therefore the verification rule (I) is satisfied.

Since $q_0, q_1 \in R$ and the invariant of the only bad state q_E is $false$, the verification rule (R) is trivially satisfied.

As a result, we have proved that the acts of jar picking and cap fitting, controlled by the hand coordinator, do interleave.

Example 5.4 The Maze Traveler: Let q_0, q_1 in Fig. 8(b) be associated with ME and $\neg ME$ respectively. The verification rule (I) is trivially satisfied.

Suppose the maximum length of an obstacle is L. Associate with each automaton-state the same function $\rho : \{0,1\} \times \{0,1\} \times \{0,1\} \times \mathcal{Z} \rightarrow \{0,1\} \times \mathcal{D}$ where $\mathcal{Z}$ is the set of integers and $\mathcal{D}$ is the interval $[0, L+1]$ of natural numbers. Let ρ be defined as:

$$\rho(ME, MN, MS, Y) = \begin{cases} \langle 1, 1+L \rangle & \text{if } ME = 1 \\ \langle 1, DN - Y \rangle & \text{if } MN = 1 \\ \langle 0, Y - DS \rangle & \text{if } MS = 1 \end{cases}$$

where DN (DS) is the Y-coordinate of the north (south) end of the current maze block. Obviously $DN - Y$ and $Y - DS \leq L$. The order on $\{0,1\} \times \mathcal{D}$ is defined as: $\langle 0, - \rangle < \langle 1, - \rangle$ and $\langle X, Y_1 \rangle \leq \langle X, Y_2 \rangle$ iff $Y_1 \leq Y_2$. $\{0,1\} \times \mathcal{D}$ is a well-founded set since L is finite. With this function and the well-founded set, any transition that ends up at $q_1 \in B$ would lead to a decrease. Therefore, ρ is a ranking function.

As a result, we have proved that the maze traveler does move east infinitely often, escaping any finite maze of that type.

We should notice that the verification method is formal but not necessarily amenable to automation if the domains of the constraint net are not finite. In fact, there is no verification algorithm, in general, since the discrete constraint net is powerful enough to simulate a Turing machine and the specification language is rich enough to state the halting problem. However, an automatic or semi-automatic theorem prover can be used for proving the validity of the formulas derived from this method.

6 Conclusion and Related Work

Will the robot do the right thing? One can guarantee the answer 'yes' by modeling the robotic system, including the environment when necessary, at an appropriate level of abstraction and proving that the model satisfies the desired behavior specification. In this paper, we have illustrated a formal approach to the modeling, specification and verification of discrete robotic systems.

We have done some further related work on this subject, (1) designing a verification algorithm for finite systems [Zhang and Mackworth, 1994c], (2) extending $\forall$-automata to timed $\forall$-automata to deal with real-time responses [Zhang and Mackworth, 1994c], and (3) extending ranking functions to Liapunov functions to deal with continuous time and domains [Zhang and Mackworth, 1994b]. We have also worked on control synthesis based on constraint satisfaction using Liapunov functions [Zhang and Mackworth, 1993a]. In fact, the problems of synthesis and verification are coupled in the design and analysis of robotic systems.

Acknowledgement

We wish to thank the anonymous referees for their constructive comments on the paper. This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

References

[Allen, 1990] J. F. Allen. Towards a general theory of action and time. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 464 – 479. Morgan Kaufmann Publishers Inc., 1990.

[Alur and Henzinger, 1989] R. Alur and T. A. Henzinger. A really temporal logic. In *30th Annual Symposium on Foundations of Computer Science*, pages 164 – 169, 1989.

[Ashcroft, 1986] E. A. Ashcroft. Dataflow and eduction: Data-driven and demand-driven distributed computation. In J. W. deBakker, W.P. deRoever, and G. Rozenberg, editors, *Current Trends in Concurrency*, number 224 in Lecture Notes on Computer Science. Springer-Verlag, 1986.

[Benveniste and LeGuernic, 1990] A. Benveniste and P. LeGuernic. Hybrid dynamical systems theory and the SIGNAL language. *IEEE Transactions on Automatic Control*, 35(5), May 1990.

[Brooks, 1991] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1 – 3), January 1991.

[Caspi *et al.*, 1987] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *ACM Proceeding of Principles of Programming Languages*, 1987.

[Emerson, 1990] E. Emerson. Temporal and modal logic. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics. Elsevier, The MIT Press, 1990.

[Lamport, 1991] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Systems Research Center, Palo Alto, California, December 1991.

[Lavignon and Shoham, 1990] J. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[Manna and Pnueli, 1987] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *Proc. 14th Ann. ACM Symp. on Principles of Programming Languages*, pages 1–12, 1987.

[Manna and Pnueli, 1992] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[McDermott, 1990] D. McDermott. A temporal logic for reasoning about processes and plans. In James Allen, James Hendler, and Austin Tate, editors, *Readings in Planning*, pages 436 – 463. Morgan Kaufmann Publishers Inc., 1990.

[Ostroff, 1989] J. S. Ostroff. *Temporal Logic For Real-Time Systems.* John Wiley & Sons Inc., 1989.

[Rosenschein, 1985] S. J. Rosenschein. Formal theories of knowledge in AI and robotics. *New Generation Computing*, 1985.

[Russell and Wefald, 1991] S. Russell and E. Wefald. *Do the Right Thing: studies in limited rationality.* MIT Press, 1991.

[Schoppers, 1991] M. Schoppers, editor. *Communications of ACM.* ACM, August 1991. Special Section on Real-Time Knowledge-Based Control Systems.

[Shannon, 1941] C. E. Shannon. Mathematical theory of the differential analyzer. *Journal of Mathematics and Physics*, 20:337 – 354, 1941.

[Shoham, 1988] Y. Shoham. *Reasoning about Change.* MIT Press, 1988.

[Sutherland, 1989] I. E. Sutherland. Micropipeline. *Communication of ACM*, 32(6), June 1989.

[Thomas, 1990] W. Thomas. Automata on infinite objects. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science.* MIT Press, 1990.

[Wolper, 1983] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72 – 99, 1983.

[Zhang and Mackworth, 1993a] Y. Zhang and A. K. Mackworth. Constraint programming in constraint nets. In *First Workshop on Principles and Practice of Constraint Programming*, pages 303–312, 1993.

[Zhang and Mackworth, 1993b] Y. Zhang and A. K. Mackworth. Design and analysis of embedded real-time systems: An elevator case study. Technical Report 93-4, Department of Computer Science, University of British Columbia, February 1993.

[Zhang and Mackworth, 1994a] Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems, 1994. Working Paper. Department of Computer Science, UBC.

[Zhang and Mackworth, 1994b] Y. Zhang and A. K. Mackworth. Specification and verification of constraint-based dynamic systems. In *Second Workshop on Principles and Practice of Constraint Programming*, May 1994.

[Zhang and Mackworth, 1994c] Y. Zhang and A. K. Mackworth. Specification and verification of discrete dynamic systems using timed ∀-automata, 1994. Working Paper. Department of Computer Science, UBC.

[Zhang, 1994] Y. Zhang. A foundation for the design and analysis of robotic systems and behaviors, 1994. PhD thesis, forthcoming. Department of Computer Science, UBC.

Searching With Abstractions: A Unifying Framework and New High-Performance Algorithm[1]

R.C. Holte, C. Drummond, M.B. Perez
Computer Science Department
University of Ottawa
Ottawa, Ontario, CANADA K1N 6N5
{holte , cdrummon , mbperez}@csi.uottawa.ca

R.M. Zimmer, A.J. MacDonald
Electrical Engineering Department
Brunel University
Uxbridge, Middlesex, ENGLAND UB8 3PH
{Robert.Zimmer , Alan.MacDonald}@brunel.ac.uk

Abstract

This paper presents a common algorithmic framework encompassing the two main methods for using an abstract solution to guide search. It identifies certain key issues in the design of techniques for using abstraction to guide search. New approaches to these issues give rise to new search techniques. Two of these are described in detail and compared experimentally with a standard search technique, classical refinement. The "alternating opportunism" technique produces shorter solutions than classical refinement with the same amount of search, and is a more robust technique in the sense that its solution lengths are very similar across a range of different abstractions of any given space.

1 Introduction

Heuristic search is ubiquitous in AI. A particular form of heuristic search, state-space search, is the cornerstone of many AI systems, including most planning and problem-solving systems. Consequently, techniques for speeding up heuristic search, or for automatically generating or improving heuristics, are of central importance to AI.

Abstraction is a widely studied means of speeding up state-space search. Instead of directly solving a problem in the original search space, the problem is mapped into and solved in an "abstract" search space. The abstract solution is then used to guide the search for a solution in the original space.

Abstraction very often produces impressive performance improvements (e.g. (Knoblock,1991)), but it is not guaranteed to speedup search. The guidance provided by an abstract solution is not even guaranteed to reduce the amount of search in the original space; if the guidance is positively misleading, it will increase the amount of search in the original space (e.g. (Holte et al., 1992)). Even if abstraction does speedup search in the original space, there are overhead costs associated with abstraction: the cost of creating an abstract space, and the costs associated with finding one or more abstract solutions and using them to guide search in the original space. The former cost can be amortized if there are many searches of the same space and the same abstract space is used each time. The speedup in the original space that results from using abstraction must more than compensate for these costs in order for a net speedup to be achieved.

Research on abstraction aims to find methods for creating and using abstract spaces that reliably speedup search without undue degradation in solution quality (i.e. the length of the solution). Most research on abstraction has investigated different methods for creating an abstract space; in this paper we investigate different methods for using an abstract solution to guide search.

There are two principal methods for using an abstract solution to guide search. In one method, the length of the abstract solution is used as a heuristic estimate of the distance to the goal (Pearl,1984; Prieditis and Janakiraman, 1993). In the other method, the individual steps of the abstract solution are used as a sequence of subgoals whose solutions link together to form the final solution (Minsky,1963; Sacerdoti,1974; Knoblock,1991; Yang and Tenenberg,1990). In the latter method, the abstract solution serves as a skeleton for the final solution; the process of

[1] This research was supported in part by an operating grant from the Natural Sciences and Engineering Research Council of Canada and in part by the EEC's ESPRIT initiative (project "PATRICIA").

"fleshing out" the abstract solution is called "refinement".

Until now, these two methods have been seen as mutually exclusive. This paper presents a computational framework in which the two methods are seen to be very similar. As algorithms, their differences are "minor", in the sense of being small in number and highly localized (i.e. independent of one another and of other aspects of the algorithms). Consequently, hybrids of the two methods can easily be constructed. But there is a much more important consequence. The algorithmic differences correspond to specific issues that arise in designing techniques for using abstract solutions. Having clearly identified these issues for the first time, it becomes immediately apparent that there are numerous promising alternatives to the existing methods. Two of these alternatives, called "path-marking" and "alternating opportunism", are examined in this paper. These techniques are experimentally compared with the classical refinement technique on a range of problems and abstraction methods. "Alternating opportunism" emerges as the best of the three techniques compared. It is between 3 and 12 times faster than breadth-first search, and very reliably produces near-optimal solutions.

Section 2 describes the method used to create abstractions, and the parameters of this method varied in the experiment. Section 3 describes the general computational framework encompassing both standard methods for using an abstract solution, and presents the three specific techniques studied in the experiment. Section 4 describes the experimental setup and results.

2 The "Star" Method of Abstraction

In most AI search systems, a search space is defined implicitly, typically in the STRIPS notation (Fikes and Nilsson,1971)). A state is defined to be a set of sentences in a formal language (containing constants, variables, predicate symbols, etc.). The successor relation between states is represented by operators that map one state to another by adding to or deleting from the set of sentences (i.e. the state) to which it is applied. Each operator has preconditions, stated in the formal language, specifying to which states the operator may be applied. An abstract search space is created by removing symbols from the formal language and/or the definitions of the operators and states (e.g. see (Knoblock et al., 1991)).

By contrast, in our system a search space is represented by an explicit graph. A state is a node in the graph; the successor relation between states is represented by edges connecting each node to its successors. In this way of representing search spaces, search space SSA is an abstraction of search space SSB iff there is a graph homomorphism from SSB to SSA.

In a graph homomorphism, each state in the abstract space, SSA, corresponds to one or more states in SSB. We view the abstract state as a class "containing" the corresponding states from SSB. Henceforth the terms "class" and "abstract state" will be used interchangeably, and the term "state" will mean a state in the original space. In the figures a class will be indicated by drawing a circle around the states it contains.

Each edge in the abstract space connects one class to another. There must be an edge in the abstract space corresponding to each edge in the original space. Thus, if there is an edge in the original space from state S1 to state S2, then there must be an edge in the abstract space from the class containing S1 to the class containing S2. If S1 and S2 are in the same class, an edge from S1 to S2 corresponds to an identity edge in the abstract space. Identity edges are not drawn in the figures.

The use of an explicit graph has the obvious drawback that it is feasible only for relatively small search spaces (the largest we have studied to date had 50,000 states). But it has the very great advantage, for research purposes, of flexibility and generality. It permits a very wide range of different abstraction-creating and abstraction-using techniques to be easily implemented and investigated. The ultimate aim of our research is to develop techniques that operate on implicit graphs; and indeed, the principles underlying new search techniques described in this paper can be used in the ordinary STRIPS representation as readily as in the explicit graph representation.

The standard STRIPS-based definition of abstraction is a highly restricted type of graph homomorphism. Some of the limitations and weaknesses of this type of graph homomorphism are discussed in (Holte et al.,1993). To overcome them, we have been developing alternative methods of abstraction.

The "star" method of abstraction was first investigated in (Mkadmi,1993). Each class consists of a "hub" state and all the neighbours of the hub within a given distance, R, called the "radius" of abstraction. The classes are built one at a time; once a state is included in a class it is ineligible to be included in any other. The process is repeated until all states have been assigned to a class (it may happen that a class contains just one state). Then edges are added between classes, as described above, to complete the construction of the abstract space.

As with any abstraction method, the star method can be applied recursively to the abstract space it creates in order to construct a "hierarchy" of abstract spaces. In our current implementation, successively more abstract search spaces are added to the hierarchy until the trivial search space is produced (the trivial search space consists of just one class). For simplicity, the discussion will speak as if there were only two levels in the hierarchy, the original search space (which is always at the bottom of the hierarchy) and one abstract search space. But all of the discussion applies equally to any two adjacent levels in a larger abstraction hierarchy.

The two main decisions in using the star method are: how to choose the hub states, and what value of R to use. These are parameters that will be varied in the experiments below. We will consider two ways of choosing hub states: choose a random state, and choose the state having the greatest number of immediate neighbours. The radius will be varied from 2 (which means a class contains only the hub and its

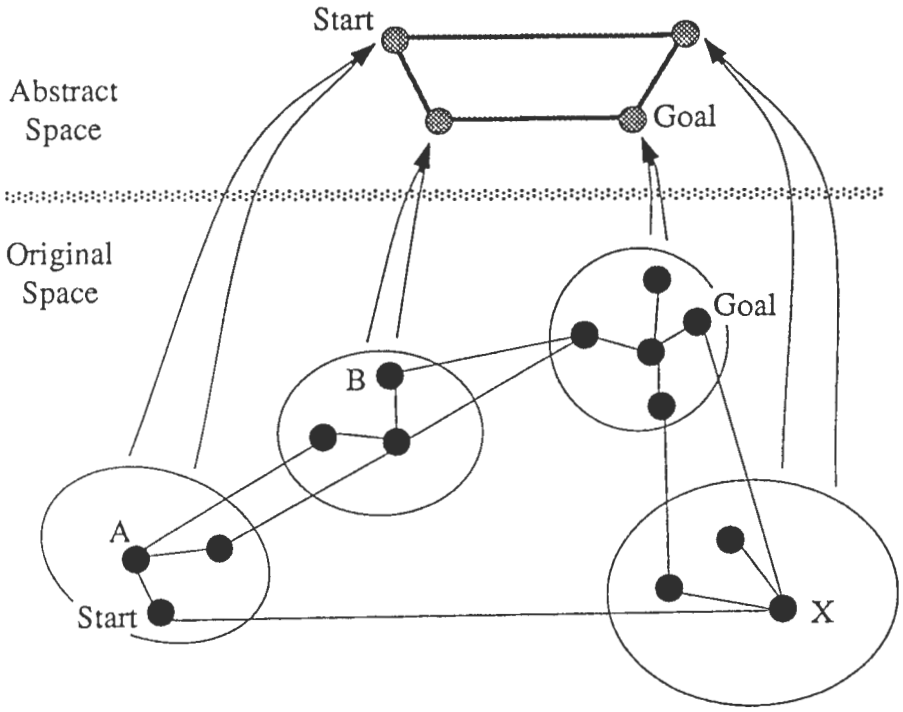


Figure 1.

immediate neighbours) to 9.

The value of R has great impact on the performance of search systems that use abstraction. For example, the total amount of search done at all levels of abstraction is bounded above by $W(R)\times(2R)^A$, where A is the number of levels in the abstraction hierarchy and W(R) is the amount of search done to solve a problem when the start and goal are both in the same class of radius R (Mkadmi,1993). Solution length is also affected by R, typically decreasing as R increases. Certain overhead costs increase as R increases, but others decrease. Generally speaking, the choice of R affects – in antagonistic ways – the quantity and quality of the guidance that an abstract solution provides for search in the original space. When R is small, there are few states in each class. Consequently, knowing which classes are in the abstract solution provides information about very few states, but the information is specific to those states and therefore is highly reliable. When R is large there are many states in each class: the abstract solution provides information about many more states, but the information is not very specific and so is indiscriminate, possibly even misleading.

3 Search Methods That Use Abstractions

There are two main ways that abstract solutions can be used to guide search. "Heuristic" methods are based on the observation that distance (number of edges in the shortest path) between two states is greater than or equal to the distance, in the abstract space, between the corresponding classes. For example, in Figure 1, the distance between Start and Goal in the original space is 3 (the shortest path passes through state X); the distance between the corresponding classes is 2. Distance in the abstract space is therefore an admissible heuristic and can be used in the A* algorithm (Hart et al., 1968) to find optimal solutions.

When the A* algorithm visits state S it computes h(S), an estimate of the distance from S to a goal state. If h(S) is defined to be the abstract distance from the class containing S to a goal class, computing h(S) involves searching in the abstract space. The result of this search is a shortest path, i.e. a sequence of classes, connecting S's class to a goal class. In Figure 1, when A* visits state A, it would compute h(A) by finding a shortest path between the class containing A and the goal class. In this example there are two shortest paths. h(A) would be 2, the length of whichever of the two was actually found.

We note, however, that the abstract path found in the course of computing h(A) provides the information needed to compute h(-) for many other states. Suppose, in the example, that the path found is the one including B's class. Firstly, for every state S in the same class as A, h(S)=h(A). Secondly, this path also allows us to compute h(B), because it includes a shortest path from B's class to the goal class. And, of course, it allows us to compute h(S) for every S in the goal class. In general, one abstract shortest path enables the computation of h(S) for every state S contained in every class on the path.

This h(-) information is easily cached with each state.

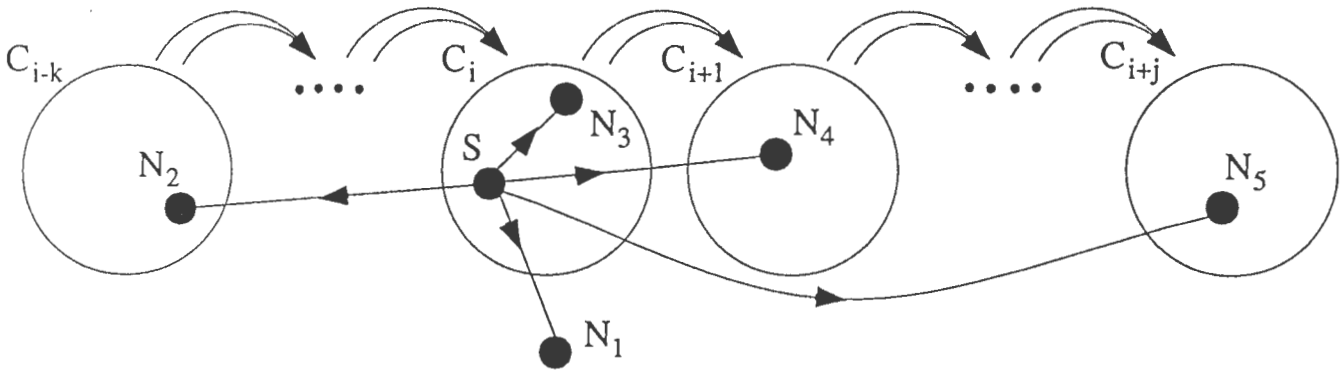


Figure 2.

Now when A* visits state S it only needs to search in the abstract space if the class containing S has not occurred on any of the abstract paths previously computed. Although this may seem like a minor implementation detail, it provides a conceptual link to the other method of using abstract paths to guide search.

"Refinement" methods find one abstract path from Start to Goal and use each class in this path as a subgoal when searching in the original space. The Start state, S1, is, by definition, in the first class, C1, on this path. A path (in the original space) is sought, typically using breadth-first search, from S1 to any state, S2, in C2, the second class on the abstract path. In searching for this path, only states in C1 are considered: all other states are disregarded (this is the graph equivalent of "goal protection"). In our example (Figure 1), a path in the original space is sought from Start to a state in the same class as B. In searching for this path, state X, and other states in X's class are ignored, because that class is not on the abstract path.

Having reached S2 in class C2, a path wholly within C2 is sought from S2 to any state, S3, in C3, the third class in the abstract path. This process, called refinement, is repeated for each successive class on the abstract path until a path in the original space has been constructed from Start to a state, Sg, in Cg, the class containing the goal state. Then, as the final step, a path wholly within Cg is sought from Sg to the actual goal state.

Figure 2 shows a portion of the abstract path $C_1 ... C_n$ being refined and a typical intermediate situation during refinement, in which we are searching forward from state S, in class C_i, looking for any state in class C_{i+1}. The five successors of S illustrate the different possibilities for a state's successors. N_1 is not in a class on the abstract path; it will be ignored. N_2 is in a previous class; it too will be ignored. N_3 and N_4 are in the current and next class, respectively. N_4 would be preferred to N_3, and would result in C_{i+1} becoming the current class. But if S happened not to have any successors in the next class, states in the same class (N_3) would be pursued during refinement. State N_5 is in a class more than 1 ahead of the current class. If the abstract path is a shortest path, such states cannot exist. However, refinement techniques (unlike the heuristic methods described above) are not guaranteed to find shortest paths, so this situation certainly can arise. In classical refinement, this opportunity to jump ahead is ignored. Techniques that exploit such opportunities we call "opportunistic".

Thus, when a state is visited during refinement, it is necessary to know whether or not it is in a class on the abstract path, and, if so, the position of the class in the abstract path. This is precisely the information that would be cached when h(Start) is computed by the implementation of the heuristic method described above. It is clear therefore that the two methods for using abstract paths are very similar, differing only in two respects:

(1) the refinement method restricts itself to states that are part of the first abstract path computed, whereas the heuristic method considers all states (and may have to find additional abstract paths to do so);
(2) the refinement method does not take into account the distance of a state from the start state, whereas the heuristic method adds this to the abstract distance to the goal in order to compute a state's overall "score".

These two classical methods are not the only ways in which the information derived from an abstract path can be used to guide search. The following are two novel variants of the classical refinement technique.

Path-Marking

Classical refinement derives its efficiency from two sources: from ignoring states that are not part of the abstract path, and from associating a position in the abstract path with each state. The path-marking technique uses only the first of the sources: it does ordinary breadth-first search (in the original space) but ignores all states that are not in classes on the abstract path. This is guaranteed to find the shortest possible refinement of the abstract path.

Alternating Search Direction

While searching in the abstract space, many classes may be visited that, in the end, are not on the abstract path. For these classes the distance to a goal class is not known; that distance is known only for classes on the abstract path. However, the distance to the abstract start class is known for all classes visited during the search, because the start class is where the search began. This information has no utility if search in the original space is in the same direction as search in the abstract space (e.g. from start to goal). But, if search in the original space proceeds in the opposite direction, from goal to start (using the inverse of the successor relation), then distance from the start is precisely what is need to guide search. If there are several levels of abstraction, search direction alternates from one level to the next.

Using this technique, search is not confined to states that are on the abstract path: heuristic distance information is available for every state in every class visited during the abstract search. This means that the solutions found by alternating search direction could be shorter than those found using the path-marking technique. Such solutions would not, of course, be refinements of the abstract path in the normal sense.

Our implementation of this alternating-direction technique is "opportunistic", as defined above.[2] Like classical refinement, alternating opportunism never moves "away" from the destination: once it finds a state whose heuristic distance from the destination is D, it ignores all states whose heuristic distance is greater than D. For this reason, the solutions found by alternating opportunism could be longer than those found by the path-marking technique.

4 Experimental Comparison

This experiment compares three search techniques — classical refinement (abbreviated CR), path-marking (PM), and alternating opportunism (AO). The two performance measures of interest are the length of the solution found, and the amount of "work" required to find a solution.

We originally measured work in terms of CPU time, but this proved extremely sensitive to low-level programming details of no significance to the algorithms themselves. In this paper, work is measured by counting the number of edges traversed during search (at all levels of abstraction) and the number of "overhead" operations performed during search (for example, to pass the heuristic distance information from one level of abstraction to the next). As these two counts have roughly comparable units, they can sensibly be added to give a composite "total work" figure. Overhead costs associated with creating the abstraction are not included in the "work" measure because our aim is to compare different techniques for using abstractions; the cost of creating the abstractions is the same for them all.

Also of interest is the robustness of a search technique: to what extent does good performance depend upon the abstract space that is used. To investigate this, abstractions were created using several different radii and two different methods for choosing the hub states (see section 2).

Four different search spaces, derived from well-known puzzles, were used in the experiment. These are described below. For each space, 100 pairs of states were chosen randomly. Each pair <S1,S2> gives rise to two problems: <start=S1,goal=S2> and <start=S2,goal=S1>. The same 200 problems were used for every different combination of system and abstraction-parameter settings. All the results shown are averages over these 200 problems.

A simple breadth-first search system was also run on the 200 test problems for each search space. Its performance figures allow us to compare the solutions found using abstraction to the optimal solutions, and to measure how much work is being saved by using abstraction to guide search.

4.1 Search Spaces

Towers of Hanoi. The 7-disk version has $3^7 = 2187$ states. Each state (except for the 3 extreme "corners") has 3 successors, but the effective branching factor is considerably less than 3 because of the structure of the space. The maximum distance between two states is $2^7 = 128$.

5-puzzle. This is a 2×3 version of the 8-puzzle. The state space comprises two unconnected regions each containing 360 states. We have connected the space by adding a single edge between one randomly chosen state in each of the two regions. Two-thirds of the states have only 2 successors, which means the branching factor at these states is effectively 1 (because every edge has an inverse, so one of the 2 successors will be the state from which the current one was reached). The other states have 3 successors.

Blocks World. There are 6 distinct blocks, numbered 1 to 6, each of which is either on the "table" or on top of another block. There is a "hand" that can hold one block at a time and execute one of four operations: put the block being held onto the table, put it down on top of a specific stack of blocks, pick up a block from the table, and pick up the block on top of a specific stack. With 6 blocks there are 7057 states. Unlike the other search spaces, in the blocks world the branching factor varies considerably from one state to another, depending as it does on the number of stacks in the state. The maximum distance between two states is 11.

Permutation. A state is a permutation of the integers 1–7; there are 7! = 5040 states. There are 6 operators numbered 2 to 7. Operator N reverses the order of the first N integers in the current state. For example, applied to the state [3,2,5,6,1,7,4] operator 4 produces [6,5,2,3,1,7,4]. Operator 7 reverses the whole permutation. All operators are

[2] we have also implemented a variation of classical refinement that is opportunistic. This was not used in the experiments because with star abstraction and the types of graphs used in the experiments, it can be proved that the heuristic distance must decrease by 1 if search is confined to the abstract path.

TABLE 1. Solution Length.
States with the most neighbours are used as the hubs of the abstract classes.
The optimal solution length is shown in brackets.

	Towers of Hanoi (66)			5-puzzle (21)			Blocks World (9.5)			Permutation (6.2)		
radius	CR	PM	AO	CR	PM	AO	CR	PM	AO	CR	PM	AO
2	98	88	80	29	27	25	14.7	13.2	11.2	11.6	10.6	8.3
3	101	77	76	27	24	24	11.5	11.0	10.8	9.5	9.3	8.0
4	80	72	75	24	23	23	12.1	11.6	11.2	9.7	8.9	8.1
5	82	72	76	29	25	25	11.9	11.2	11.3	8.1	7.3	7.4
6	82	72	77	27	26	25	11.6	10.5	10.8	7.1	6.4	6.8
7	79	69	75	26	25	24	10.3	9.7	10.2	7.3	6.4	6.8
8	73	69	71	25	25	24	9.5	9.5	9.5	6.2	6.2	6.2
9	78	68	74	26	25	24	9.5	9.5	9.5	6.2	6.2	6.2

TABLE 2. Total Work (#edges + overhead).
States with the most neighbours are used as the hubs of the abstract classes.
The work done by ordinary breadth-first search is shown in brackets.

	Towers of Hanoi (3058)			5-puzzle (802)			Blocks World (3936)			Permutation (5570)		
radius	CR	PM	AO	CR	PM	AO	CR	PM	AO	CR	PM	AO
2	894	1048	767	299	362	255	724	1073	762	512	868	460
3	867	903	711	251	308	238	1200	1339	1227	616	762	609
4	776	903	751	294	333	301	902	1127	889	750	1191	790
5	752	927	740	314	371	300	1268	1767	1395	1407	2443	2527
6	777	944	765	335	392	321	2124	2906	2146	3055	3926	5454
7	802	950	828	333	392	325	3424	4258	4149	5739	7053	6416
8	932	1079	938	348	413	346	5915	5915	5973	7982	7982	7982
9	916	1082	946	386	469	389	5973	5973	5973	7982	7982	7982

applicable in every state, so each state has 6 successors. The maximum distance between two states is 14.

4.2 Results

Tables 1 and 2 show the solution length and total work results when states having the most immediate neighbours are used as the hubs of abstract classes. The solutions found using abstraction are relatively short, within 30% of the optimal length in most cases and never more than double the optimal length. Total work is impressively small for small radii, in most cases, but for large radii search using abstraction is not cost-effective. This is particularly evident in the Blocks World and Permutation spaces, whose maximum distance between states is small: a radius of 7 or greater causes almost all states to be put into the same class, making abstraction of little value. Only in the Towers of Hanoi space, where the maximum distance between two states is large, does abstraction with large radii pay off. In the comparisons that follow, data for radius ≥ 7 for the Blocks World and Permutation spaces will be ignored.

The experiment shows that PM's solutions are roughly 10% shorter than CR's. Regarding total work, it is possible for PM to do less than CR, if their solutions are different lengths. In the experiment, this never happened: PM always did more work, sometimes much more. In most circumstances, the 10% improvement in solution length PM provides is probably not sufficient to justify the additional work.

CR produces its poorest solutions when radius=2. It was hoped that PM would find much shorter solutions in this case, but the experiment reveals that its solutions are just 10% shorter, as usual. Since PM produces the optimal refinement of an abstract solution, one may conclude that the relatively poor performance when radius=2 is an inherent property of the the general strategy of using a single abstract solution to guide search.

AO does not follow the same general strategy, and its solutions are 10-15% shorter than PM's when radius=2. For larger radii, AO and PM give solutions of similar lengths – PM's are slightly shorter in the Towers of Hanoi space, AO's in the Permutation space. However, AO always does much less work than PM. AO does the same amount of work as CR and produces shorter solutions, sometimes very much shorter. The single exception is the Permutation space, where AO begins to degenerate to breadth-first search slightly sooner (radius=5) than CR.

The same patterns arise when random states are used as the hubs when constructing abstract classes (see Tables 3 and 4). The solutions found by all techniques have increased in length but CR's have increased more than PM's which, in turn, have increased more than AO's. Consequently, the difference in solution lengths has become

TABLE 3. Solution Length.
Randomly chosen states are used as the hubs of the abstract classes.
The optimal solution length is shown in brackets.

	Towers of Hanoi (66)			5-puzzle (21)			Blocks World (9.5)			Permutation (6.2)		
radius	CR	PM	AO	CR	PM	AO	CR	PM	AO	CR	PM	AO
2	91	80	75	32	29	26	27.2	21.3	12.7	15.6	12.0	8.4
3	95	81	83	30	29	26	22.0	19.1	13.1	11.4	10.0	8.1
4	82	72	77	27	25	25	19.7	16.5	13.6	10.1	9.6	8.5
5	86	74	78	28	25	24	14.3	13.6	12.1	9.5	8.5	8.6
6	95	86	79	26	26	24	15.6	14.8	13.0	9.0	7.7	8.0
7	88	81	79	25	25	25	17.0	16.2	13.8	6.9	6.4	6.6

TABLE 4. Total Work (#edges + overhead).
Randomly chosen states are used as the hubs of the abstract classes.
The work done by ordinary breadth-first search is shown in brackets.

	Towers of Hanoi (3058)			5-puzzle (802)			Blocks World (3936)			Permutation (5570)		
radius	CR	PM	AO	CR	PM	AO	CR	PM	AO	CR	PM	AO
2	847	1021	765	316	411	285	1310	1526	1070	806	1015	817
3	794	957	763	282	338	264	571	783	546	800	978	806
4	734	870	738	269	337	264	699	1027	738	811	1250	1038
5	776	924	763	280	360	265	902	1093	896	1258	2491	1456
6	931	1111	830	306	363	303	808	1048	780	3142	5064	3674
7	942	1106	916	292	354	302	964	1297	964	6295	7008	6566

greater (except, perhaps, for the Towers of Hanoi). It is now the case that the shortest of CR's solutions for any radius is comparable in length to AO's longest solution.

For radii other than 2, total work has decreased in most cases; dramatically so in the Blocks World. But all the search techniques have benefited roughly equally: CR and AO still do comparable amounts of work, and PM does significantly more.

AO's solution lengths are remarkably insensitive to the manner in which hub states are chosen and to the choice of radius (as long as the radius is not so large as to cause AO to degenerate to breadth-first search). This robustness of the search technique is important because it relieves the abstraction-constructing system of the responsibility of ensuring good solutions. The abstraction-construction system can therefore focus on other issues; for example, it could attempt to construct abstract spaces that were "meaningful" to a human in the sense that each class has a succinct description.

6 Conclusions

This paper has provided a common algorithmic framework encompassing the two main methods of using an abstract solution to guide search. In doing so, it has identified certain key issues in the design of techniques for using abstraction to guide search. The clear identification of these issues is important for research in abstraction, because doing so sharply focuses research. In response to these issues, two new new search techniques have been developed — path-marking and alternating opportunism. These have been compared experimentally with a standard search technique, classical refinement. Path-marking is guaranteed to find the optimal refinement of a given abstract path; the experiments show that classical refinement, in general, produces refinements whose length are within 10% of the optimal refinement. However, the optimal refinement of a somewhat arbitrarily chosen abstract path is not necessarily close to being an optimal solution. The alternating opportunism technique is based on a generalization of the notion of "refinement of an abstract path". It produces shorter solutions than classical refinement with the same amount of search. It is also a more robust technique, in the sense that its solution lengths are very similar across a range of different abstractions of any given space.

This study has been carried out with a system in which search spaces are represented as explicit graphs. However, this is incidental to the general framework and specific techniques developed and compared in the paper. Path-marking, alternating search direction, and opportunism could all be implemented as search techniques in a traditional STRIPS-style search system.

References

[Fikes and Nilsson, 1971] Fikes, R. and N.J. Nilsson (1971), "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence, vol.2, pp.189-208.

[Hart et al., 1968] Hart, P.E., N.J. Nilsson, and B. Raphael (1968), "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on Systems Science and Cybernetics, vol.4(2), pp.100-107.

[Holte et al., 1993] Holte, R.C., R. Zimmer, and A.J. Macdonald (1993), "A Study of the Representation-Dependency of Abstraction Techniques", ML'93 workshop on Knowledge Compilation and Speedup Learning, June 1993. (unpublished)

[Holte et al., 1992] Holte, R.C., R. Zimmer, and A. MacDonald (1992), "When does Changing Representation Improve Problem-Solving Performance ?", in M. Lowry (ed.), Proceedings of the Workshop on Change of Representation and Problem Reformulation, NASA Ames technical report FIA-92-06. May 1992.

[Knoblock, 1991] Knoblock, C.A. (1991). Automatically Generating Abstractions for Problem Solving. tech. report CMU-CS-91-120, Computer Science Dept., Carnegie-Mellon University.

[Knoblock et al., 1991] Knoblock, C.A., J.D. Tenenberg, and Q. Yang (1991), "Characterizing Abstraction Hierarchies for Planning", Proc. AAAI, pp.692-697.

[Minsky, 1963] Minsky, M. (1963), "Steps Toward Artificial Intelligence", in Computers and Thought, E. Feigenbaum and J. Feldman (eds.), McGraw-Hill, pp.406-452.

[Mkadmi, 1993] Mkadmi, T. (1993). Speeding Up State-Space Search by Automatic Abstraction. Master's Thesis. Computer Science Dept., University of Ottawa.

[Pearl, 1984] Pearl, J. (1984), Heuristics, Addison-Wesley.

[Prieditis and Janakiraman, 1993] Prieditis, A., and B. Janakiraman (1993), "Generating Effective Admissible Heuristics by Abstraction and Reconstitution", Proc. AAAI, pp.743-748.

[Sacerdoti, 1974] Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, vol. 5(2), pp. 115-135.

[Yang and Tenenberg, 1990] Yang, Q. and J.D. Tenenberg (1990). Abtweak: Abstracting a nonlinear, least commitment planner. *Proc. AAAI'90*, pp. 204-209.

An Argument for Indexical Representations in Temporal Reasoning*

Yves Lespérance and **Hector J. Levesque**†
Department of Computer Science
University of Toronto
Toronto, ON, Canada M5S 1A4
{lesperan, hector}@ai.utoronto.ca

Abstract

This paper discusses the need for indexicals in a representation language. It has been claimed that the cost of updating a knowledge base containing indexicals would be prohibitive and thus that a robot should use its internal clock to eliminate indexicals from its representations. We criticize this view and give an example of a commonplace temporal reasoning/planning problem that can only be solved in a representation formalism that includes both indexical and absolute terms and supports reasoning using both. We show that the example can be formalized within our theory of knowledge and action. We argue that rather than trying to find restricted settings where indexical knowledge can be reduced to objective knowledge, one should investigate when and how planning and temporal knowledge base update can be performed efficiently in the presence of indexicals.

1 Introduction

To someone accustomed to the objective point of view of science or mathematics, indexicality (context-sensitivity) may appear as little more than an artifact of natural language. One may thus claim that while using indexical descriptions (e.g., *now, three hours ago, one meter in front of me*) is often convenient, in practice, indexical knowledge can always be understood objectively. One reason for wishing that this claim were true has to do with the fact that the semantic content of indexical representations depends on the context, so if the context changes you may have to adjust the representations to keep the semantic content unchanged. For instance, if an agent's knowledge base describes some facts as holding 'now', then at the next time step, it should describe these facts as holding 'one time step before now'.[1] Haas [1991] points out that the cost of adjusting a knowledge base that contains indexical time references for the passage of time would be high, if implemented in the obvious way. He proposes that a robot use its internal clock to eliminate all occurrences of 'now' in its representations.

This proposal and the general strategy of trying to eliminate indexical representations are misguided.[2] First, humans do not have internal clocks that they can use in the way a robot can, and they do not always know what time it is. A system that is interacting with humans will need to model this (e.g. to remind a user that his meeting is just starting). Even if we limit ourselves to simple robotics contexts, it seems unlikely that the internal clocks of robots could always be guaranteed to be accurate. In such cases, Haas's scheme leads to indexical information being misrepresented. Moreover, Haas's robot cannot even represent the fact that his internal clock is incorrect; it could not monitor for this and plan to get its clock reset when appropriate. Also, for very simple (e.g. insect-like) robots, the cost of fitting them with internal clocks and setting them may be too high. Finally, as Haas recognizes, it is not at all clear that the cost of updating a temporal knowledge base that contains indexicals need be prohibitive; for example, if all occurrences of 'now' are replaced by a new constant and the fact that this new constant is equal to 'now' is added, then only this single assertion need be updated as time passes.

In the next section, we will give an example of a commonplace temporal reasoning/planning problem that can only be solved in a representation formalism that *includes both indexical and non-indexical concepts and supports reasoning using both.* The example involves an agent that does not initially know what time it is; he must keep track of time in relative terms (using a timer), but later convert this indexical knowledge into absolute knowledge for communication to another agent. We will

*This research received financial support from the Institute for Robotics and Intelligent Systems (Canada), the Information Technology Research Center (Ontario, Canada), and the Natural Science and Engineering Research Council (Canada).

†Fellow of the Canadian Institute for Advanced Research

[1] Subramanian and Woodfill [1989] prove that such a transformation is truth-preserving within their indexical situation calculus framework.

[2] Philosophers such as Perry [1979] have long argued that indexical knowledge cannot be eliminated. But the examples they cite to support their position can appear farfetched. Here we try to make the point that this does apply to the practice of building intelligent systems.

show in detail in Section 4 that this example can be formalized within the theory of knowledge and action that we proposed in [Lespérance and Levesque, 1994; Lespérance, 1991]. We will return to our discussion of whether indexical knowledge can be eliminated in the final section.

2 The Example

The example goes as follows. Imagine that you arrive at home and are greeted by the following note:

> "The turkey is ready to go into the oven (at 325°F). I will be home in time to take it out, but leave me a message before you go telling me at what time you put it in."

Unfortunately, the only timing devices you have are the one-hour timer on the stove, and a radio station that announces the time at least every 30 minutes. You also want to put the turkey to roast as early as possible and be done as quickly as possible.

It is not too hard to see what needs to be done. A reasonable plan is to put the turkey in the oven, set the timer to one hour, then listen to the radio until the time is announced while keeping track of the roasting time with the timer, and finally calculate the time the turkey started roasting, and leave a message to that effect.[3]

Obviously, a single example by itself is not much of an argument for anything. However, the example illustrates a simple but extremely common situation where two very different notions of time need to be dealt with: indexical or relative time, as determined by the timer ("the turkey went in 20 minutes ago"), and absolute or objective time, as calculated from the radio announcement ("the turkey went in at 12:45pm"). The key point is that either notion of time *by itself* is inadequate to model the situation. To see this, observe that the problem cannot be solved without the radio, and yet all it provides is information like "It is now 1:05pm." To make sense of this essential piece of information, we need to be able to relate the two separate conceptions of time. In purely indexical or purely absolute terms, the information is meaningless.

3 Overview of the Formalism

Before presenting a formalization of the example, we briefly review a formal theory of indexical knowledge, action and ability (see [Lespérance and Levesque, 1994; Lespérance, 1991] for a more detailed presentation.) The goal is to be able to express attributions of indexical knowledge, for example, that Rob knows that he himself was holding a cup five minutes ago.[4] In such cases, what is known is a "proposition" that is relative. It may be relative to the knower, or to the time of the knowing, or perhaps to other aspects of the context. To handle this, our language includes two special terms: **self**, which denotes the current agent, and **now**, which denotes the current time; these terms are called *primitive indexicals*. Non-logical (domain-dependent) symbols may also depend on the current agent and time for their interpretation, for example, $\exists x \textsc{Holding}(x)$ may express the fact that the current agent is currently holding something — we say that such symbols are *non-primitive indexicals*. Our semantics handles this by interpreting terms and formulas with respect to *indices*, which consist of a possible-world (modeling the objective circumstances), and an agent and a time (modeling the context).

The language used is a many-sorted first-order modal language with equality called *LIKA* (Language of Indexical Knowledge and Action). It includes terms of four different sorts: terms for ordinary individuals (as usual), temporal terms, agent terms, and action terms. For each of these sorts, there are both variables and function symbols (i.e., functions whose values are of the proper sort); as usual, constants are taken to be 0-ary function symbols.

The atomic formulas include predications using predicate symbols and terms, written $R(\theta_1, \ldots \theta_n)$, which are used to assert that $\theta_1, \ldots \theta_n$ stand in static relation R at the current time for the current agent. We also have equality expressions $(\theta_1 = \theta_2)$, between terms of the same sort, as well as expressions of temporal precedence $(\theta_1^t < \theta_2^t)$. We assume that time is linearly ordered. Finally, $\mathbf{Does}(\theta^d, \theta^t)$ is used to assert that the current agent does action θ^d starting from the current time and ending at time θ^t.

Non-atomic formulas may be composed using the standard boolean connectives and quantifiers as well as a set of modal operators. $\mathbf{At}(\theta^t, \varphi)$ means that φ holds at time θ^t, that is, when θ^t is taken to be the current time. $\mathbf{By}(\theta^a, \varphi)$ means that φ holds when θ^a is taken to be the current agent. $\Box\varphi$ means that φ is historically necessary at the current time, that is, that φ now holds in all possible courses of events that are identical to the current one up to the current time. We also introduce a dual to $\Box$: $\Diamond\varphi \stackrel{\text{def}}{=} \neg\Box\neg\varphi$.

$\mathbf{Know}(\varphi)$ is used to represent the fact that the current agent knows at the current time that φ. If φ contains indexical elements, $\mathbf{Know}(\varphi)$ should be taken as attributing indexical knowledge, that is, knowledge the agent has about himself and the current time. For example, $\mathbf{Know}(\textsc{Holding}(x))$ could mean that the agent knows that *he himself* is *currently* holding the object denoted by x. The semantics for **Know** is a simple generalization of the standard possible-world scheme. The knowledge accessibility relation K is taken to hold over indices rather than plain possible worlds. Informally, $\langle\langle \mathsf{w}, \mathsf{a}, \mathsf{t}\rangle, \langle \mathsf{w}', \mathsf{a}', \mathsf{t}'\rangle\rangle \in \mathsf{K}$ if and only if as far as agent a at time t in world w knows, it may be the case that w' is the way the world actually is *and* he is a' *and* the current time is t'. Thus, we allow an agent to be uncertain not only about what world he is in, but also about who he is and what time it is. We assume that the knowledge accessibility relation K is reflexive and transitive, meaning that **Know** obeys the principles of modal logic S4. We also assume that agents have perfect memory and

[3] It would be more efficient to listen to the radio *as* one is putting the turkey in, but we do not want to deal with true concurrency here.

[4] As well, we want to be able to distinguish this from having objective knowledge, such as Rob's knowing that Rob was holding a cup at some specified time, say, 4:37pm.

always know what actions they have done.

For the example to follow, it is convenient to make two general assumptions beyond those embodied in the logic. First, we assume that the domain of times is (or is isomorphic to) the integers, that is, we assume that the facts about integer arithmetic that we need are valid and that constants and function symbols representing arithmetic operations are rigid. Secondly, we assume that all actions are "solid", in the sense that any overlapping instances of an action (type) must be the same instance (have the same endpoints) [Shoham, 1987]. This ensures that we can refer to things like "the starting time of the action I have just done".

To talk more easily about a wider class of actions, it is useful to extend the use of **Does** to a new syntactic category, that of *action expressions*. These include action terms as above, which represent simple actions, **noOp**, which represents the empty action and takes no time, $(\delta_1;\delta_2)$, which represents the sequential composition of the actions δ_1 and δ_2, and $\mathbf{if}(\varphi,\delta_1,\delta_2)$, which represents the action that consists in doing action δ_1 if the condition φ holds, and in doing action δ_2 otherwise. Formulas of the form $\mathbf{Does}(\delta,\theta^t)$ where δ is an action expression, can be thought of as abbreviations that reduce to formulas where **Does** ranges only over the simple action terms, in the obvious way. We also define a bounded form of "while loop" as an action expression as follows:

$$\mathbf{while}_{\mathsf{k}}(\varphi,\delta) \stackrel{\text{def}}{=} \begin{cases} \mathbf{noOp} & \text{if } \mathsf{k}=0 \\ \mathbf{if}(\varphi,(\delta;\mathbf{while}_{\mathsf{k}-1}(\varphi,\delta)),\mathbf{noOp}) & \text{if } \mathsf{k}>0, \mathsf{k}\in\mathbb{N} \end{cases}$$

Let us also define some dynamic-logic-style operators that will be used in our formalization of ability. $\mathbf{AfterNec}(\delta,\varphi)$, which is intended to mean "φ must hold after δ", is defined inductively as follows:

$\mathbf{AfterNec}(\theta^d,\varphi) \stackrel{\text{def}}{=} \Box\forall v^t(\mathbf{Does}(\theta^d,v^t) \supset \mathbf{At}(v^t,\varphi))$, where v^t is a temporal variable that does not occur free in φ

$\mathbf{AfterNec}(\mathbf{noOp},\varphi) \stackrel{\text{def}}{=} \varphi$

$\mathbf{AfterNec}((\delta_1;\delta_2),\varphi) \stackrel{\text{def}}{=}$
$\mathbf{AfterNec}(\delta_1,\mathbf{AfterNec}(\delta_2,\varphi))$

$\mathbf{AfterNec}(\mathbf{if}(\varphi_c,\delta_1,\delta_2),\varphi) \stackrel{\text{def}}{=}$
$(\varphi_c \supset \mathbf{AfterNec}(\delta_1,\varphi)) \wedge$
$(\neg\varphi_c \supset \mathbf{AfterNec}(\delta_2,\varphi))$

Also, let $\mathbf{PhyPoss}(\delta) \stackrel{\text{def}}{=} \neg\mathbf{AfterNec}(\delta,\mathbf{False})$. $\mathbf{PhyPoss}(\delta)$ is intended to mean that it is "physically possible" for **self** to do action δ next (even though he may not be able to do it because he does not know what primitive actions δ stands for). **True** (**False**) stands for some tautology (contradiction).

Our formalization of ability, based on that of Moore [1980], says that the agent is able to achieve the goal φ by doing action δ, formally $\mathbf{Can}(\delta,\varphi)$, if and only if he can do action δ and knows that after doing δ, the goal φ must hold:

$\mathbf{Can}(\delta,\varphi) \stackrel{\text{def}}{=} \mathbf{CanDo}(\delta) \wedge \mathbf{Know}(\mathbf{AfterNec}(\delta,\varphi))$

$\mathbf{CanDo}(\delta)$ is defined inductively as follows:[5]

$\mathbf{CanDo}(\theta^d) \stackrel{\text{def}}{=}$
$\exists v^d\,\mathbf{Know}(v^d=\theta^d)\wedge\mathbf{Know}(\mathbf{PhyPoss}(\theta^d))$, where action variable v^d does not occur free in θ^d

$\mathbf{CanDo}(\mathbf{noOp}) \stackrel{\text{def}}{=} \mathbf{True}$

$\mathbf{CanDo}(\delta_1;\delta_2) \stackrel{\text{def}}{=} \mathbf{Can}(\delta_1,\mathbf{CanDo}(\delta_2))$

$\mathbf{CanDo}(\mathbf{if}(\varphi,\delta_1,\delta_2)) \stackrel{\text{def}}{=}$
$(\mathbf{Know}(\varphi)\wedge\mathbf{CanDo}(\delta_1)) \vee$
$(\mathbf{Know}(\neg\varphi_c)\wedge\mathbf{CanDo}(\delta_2))$

Note that we eliminate Moore's requirement that the agent know who he is; instead, we require indexical knowledge (see [Lespérance and Levesque, 1994] for a discussion of why this is better). Also, as we will see in the next section, the fact that our account of ability is based on a more expressive temporal logic allows it to deal with actions whose prerequisites or effects involve knowledge of absolute times and knowing what time it is.

4 Formalizing the Example

Let us formalize the example and prove that the agent is able to achieve his goal by executing the proposed plan given some reasonable assumptions about his initial knowledge. Our formalization will be rather simplistic, but could easily be made more accurate and general. We define the agent's plan as follows:

GETDINNERGOING $\stackrel{\text{def}}{=}$
STARTROASTING; SETTIMER(1H);
LISTENUNTILTIMEANNOUNCED;
LOOKATTIMER; LEAVEMSG

We use abbreviations such as 1H (1 hour) and 30MIN (30 minutes) for readability; it is assumed that such abbreviations stand for the corresponding number of seconds. The complex action LISTENUNTILTIMEANNOUNCED is defined below. Let ANNOUNCEDTIME mean that the time has been announced on the radio during the last step of LISTEN:[6]

ANNOUNCEDTIME $\stackrel{\text{def}}{=}$
$\exists t_s\exists t_i(\mathbf{DoesFromTo}(\text{LISTEN},t_s,\mathbf{now})\wedge$
$t_s \le t_i \le \mathbf{now} \wedge \mathbf{At}(t_i,\text{ANNOUNCINGTIME}))$

LISTENUNTILTIMEANNOUNCED stands for repeatedly doing LISTEN until the time is announced (at least once and at most some very large number of times vIn):

LISTENUNTILTIMEANNOUNCED $\stackrel{\text{def}}{=}$
LISTEN; $\mathbf{while}_{\mathsf{vIn}}(\neg\text{ANNOUNCEDTIME}, \text{LISTEN})$

[5] This way of defining **Can** is preferable to the one in [Lespérance, 1991; Lespérance and Levesque, 1990] as it separates the knowledge prerequisites involving the goal from the rest. The definitions of **AfterNec** and **PhyPoss** given here are also changed; they now behave exactly as their dynamic logic [Goldblatt, 1987] counterparts do.

[6] $\mathbf{DoesFromTo}(\delta,\theta^t_s,\theta^t_e)$ means that the agent does action δ from time θ^t_s to time θ^t_e; formally: $\mathbf{DoesFromTo}(\delta,\theta^t_s,\theta^t_e) \stackrel{\text{def}}{=} \exists v^t_e(v^t_e=\theta^t_e\wedge\mathbf{At}(\theta^t_s,\mathbf{Does}(\delta,v^t_e)))$, provided that v^t_e does not occur anywhere in θ^t_s, θ^t_e, or δ.

Let us now formalize the actions involved in the plan. First note that we must specify an appropriate limit on the duration of the actions, otherwise they might take so much time as to prevent the goal from being achieved. But note also that we cannot have these actions take a fixed length of time that is known to the agent, for otherwise, he could use them to measure time and dispense with the timer. What we will do is specify that basic actions take a indeterminate amount of time that must be between given bounds. So we specify the effects of the STARTROASTING action as follows:

Assumption 1 (Effects of STARTROASTING)

$$\models \forall t_s \forall t_e(\mathbf{DoesFromTo}(\text{STARTROASTING}, t_s, t_e) \supset t_e = (t_s + 1\text{MIN}) \pm 30\text{S} \wedge \mathbf{At}(t_e, \text{ROASTING}) \wedge \forall t(t_s < t < t_e \supset \mathbf{At}(t, \neg\text{ROASTING})))$$

This says that setting the turkey to roast takes one minute plus or minus 30 seconds (i.e., between half a minute and a minute and a half), that the turkey is roasting afterwards, and that it is not roasting while the action is being done.

For the action of setting the timer, we have the following:

Assumption 2 (Effects of SETTIMER)

$$\models \forall t_s \forall t_e \forall n(\mathbf{DoesFromTo}(\text{SETTIMER}(n), t_s, t_e) \supset t_e = (t_s + 15\text{S}) \pm 5\text{S} \wedge \mathbf{At}(t_e, \text{TIMERVAL} = n))$$

This says that doing SETTIMER(n) takes 15 seconds plus or minus 5 seconds, and that afterwards, the timer is set to show that the time left is n seconds.

For the action of listening to the radio (LISTEN), we assume that it takes 10 seconds plus or minus 4 seconds, and that afterwards, the agent either knows that the time has not been announced during the action, or knows that it has been announced and knows what time it is within a margin of error of 7 seconds:

Assumption 3 (Effects of LISTEN)

$$\models \forall t_s \forall t_e(\mathbf{DoesFromTo}(\text{LISTEN}, t_s, t_e) \supset t_e = (t_s + 10\text{S}) \pm 4\text{S} \wedge \mathbf{At}(t_e, \mathbf{Know}(\neg\text{ANNOUNCEDTIME}) \vee (\mathbf{Know}(\text{ANNOUNCEDTIME}) \wedge \exists t \mathbf{Know}(\mathbf{now} = t \pm 7\text{S}))))$$

It is assumed that the time is announced on the radio at least every half hour.

For the action of looking at the timer, we assume that it takes 5 seconds plus or minus two seconds and that afterwards, the agent knows what duration is left on the timer:

Assumption 4 (Effects of LOOKATTIMER)

$$\models \forall t_s \forall t_e(\mathbf{DoesFromTo}(\text{LOOKATTIMER}, t_s, t_e) \supset t_e = (t_s + 5\text{S}) \pm 2\text{S} \wedge \mathbf{At}(t_e, \exists n \mathbf{Know}(n = \text{TIMERVAL})))$$

Finally, we need to specify the effects of leaving a message. Let $\text{TSR}(\theta^t)$ stand for the claim that the turkey has been roasting since time θ^t and was not roasting prior to that:

$$\text{TSR}(\theta^t) \stackrel{\text{def}}{=} \forall v^t(\theta^t \leq v^t \leq \mathbf{now} \supset \mathbf{At}(v^t, \text{ROASTING})) \wedge \mathbf{At}(\theta^t - 1\text{S}, \neg\text{ROASTING}),$$

where v^t is not free in θ^t

We will assume that LEAVEMSG, that is, leaving a message about what time the turkey started roasting, takes one minutes plus or minus 30 seconds, and that afterwards, there must be a message on the table stating that the turkey started roasting at some time t_m that is within ϵ seconds of the time at which the turkey actually started roasting:

Assumption 5 (Effects of LEAVEMSG)

$$\models \forall t_s \forall t_e(\mathbf{DoesFromTo}(\text{LEAVEMSG}, t_s, t_e) \supset t_e = (t_s + 1\text{MIN}) \pm 30\text{S} \wedge \mathbf{At}(t_e, \exists t_r \exists t_m(\text{MSGONTBL}(t_m) \wedge \text{TSR}(t_r) \wedge t_m = t_r \pm \epsilon)))$$

For this example, the error bound ϵ can be made as tight as 14 seconds, but nothing depends crucially on this.

Now, let us specify the physical prerequisites of the actions. We assume that it is physically possible for the agent to do STARTROASTING whenever the turkey is not roasting:

Assumption 6 (Prerequisites of STARTROASTING)

$$\models \neg\text{ROASTING} \supset \mathbf{PhyPoss}(\text{STARTROASTING})$$

We also assume that it is physically possible for the agent to set the stove timer to any duration between 0 and 1 hour (the formal statement is similar to the one above). As well, it is assumed that LISTEN, LOOKATTIMER, and LEAVEMSG are always physically possible.

We must also specify the conditions under which agents know how to perform the basic actions. We assume that one always knows how to do STARTROASTING:

Assumption 7 (STARTROASTING is known)

$$\models \exists d \mathbf{Know}(d = \text{STARTROASTING})$$

Similarly, we assume that agents always know how to do LOOKATTIMER, LISTEN, and SETTIMER(n) (for any n). Finally, we assume that the agent must know how to do LEAVEMSG if he knows when the turkey started roasting within a margin of error of ϵ:

Assumption 8 (LEAVEMSG is known)

$$\models \exists t_m \mathbf{Know}(\exists t_r(\text{TSR}(t_r) \wedge t_r = t_m \pm \epsilon)) \supset \exists d \mathbf{Know}(d = \text{LEAVEMSG}))$$

We also have various frame assumptions that specify what remains unchanged as actions are done. First, we assume that actions other than STARTROASTING have no effects on whether or not the turkey is roasting.[7] Secondly, we assume that for any action other than setting the timer, the time shown by the timer must accurately reflects the passage of time during the action:

Assumption 9 (Frame assumption about timer values)

$$\models \forall t_s \forall t_e \forall d \forall n \forall t_i(\mathbf{DoesFromTo}(d, t_s, t_e) \wedge \forall m(d \neq \text{SETTIMER}(m)) \wedge \mathbf{At}(t_s, n = \text{TIMERVAL}) \wedge t_s < t_i \leq t_e \supset \mathbf{At}(t_i, \text{TIMERVAL} = \text{MAX}(0, n - (t_i - t_s))))$$

[7] This frame assumption (and the subsequent ones) should not really be taken to hold for all actions. But given the limited domain under consideration, this causes no harm. Such assumptions would probably be best specified as default statements.

Finally, we have unique name assumptions for all the actions introduced.

Given this formalization, our framework now allows us to prove the following proposition, which says that if the agent knows that the turkey is not yet roasting, then by doing the action GETDINNERGOING, he is able to achieve the goal that there be a message on the table telling when the turkey started roasting within ϵ seconds of accuracy and that the time after the action be less than 32 minutes and 12 seconds after the turkey started roasting:

Proposition 1

$$\models \mathbf{Know}(\neg \text{ROASTING}) \supset \mathbf{Can}(\text{GETDINNERGOING}, \exists t_r \exists t_m (\text{MSGONTBL}(t_m) \wedge \text{TSR}(t_r) \wedge t_r = t_m \pm \epsilon \wedge \mathbf{now} - t_r \leq 32\text{MIN}12\text{S}))$$

A sketch of the proof is provided in appendix.

5 Discussion

Let us return to our discussion of the claim that indexical knowledge can be reduced to objective knowledge. In our semantics for knowledge, indexical terms and formulas are treated as relative to an agent and a time; for example, knowing that something is **here** amounts to knowing that something is at one's position at the current time. Given this, it is clear that if one knows who one is and knows what time it is (we are taking this to require knowing a standard name), then anything that one knows in an indexical way is also known in an objective way. But is it reasonable to assume that an agent always knows who he is and what time it is?

As argued in section 1, the temporal part of this question must clearly be answered negatively.[8] Humans do not always know what time it is and computers need to model this. And even robots sometimes need to get their internal clocks reset. Work on reactive agent architectures supplies other reasons for wanting a formalism that can represent indexical knowledge. As pointed out by Agre and Chapman [1987], the world can change in unexpected ways and reasoning about change can be very costly; in some cases it is better to rely on perception to get fresh information at every time step rather than try to update a representation of the world; in such cases, the problem of updating indexical representations does not arise. And as Rosenschein and Kaelbling [1986] have shown, it is legitimate to ascribe knowledge to agents even when they have no explicit representation of this knowledge. In such cases, one needs a formalism that distinguishes between indexical and objective knowledge just to accurately model the agent's thinking. The output of the agent's perception module says nothing about time, and even if the agent has a correct internal clock, he may have no need to time-stamp his knowledge. We want a formalism that makes the distinctions required to model this.

[8] We also think that it should not be assumed that agents always know who they are; see [Lespérance and Levesque, 1994; Lespérance, 1991; Grove and Halpern, 1991] for arguments.

Thus, rather than trying to find restricted settings where indexical knowledge can be reduced to objective knowledge, we think it would be more productive to investigate *when and how planning and temporal knowledge base update can be performed efficiently in the presence of indexicals.* Once one allows for agents not knowing what time it is, then examples like the one formalized here easily come to mind, examples that require both indexical and absolute terms for their representation and the ability to relate them in reasoning. A formalism along the lines of ours is required for handling such cases. Grove and Halpern's logic of knowledge [Grove and Halpern, 1991] handles some aspects of indexicality, but does not deal with time; so it cannot handle the kinds of situations discussed here. Subramanian and Woodfill's version of the situation calculus [Subramanian and Woodfill, 1989] handles aspects of indexicality, but not knowledge; thus, it cannot account for knowledge acquisition actions.

Let us conclude by discussing various areas in which our framework could be extended or improved. It may be possible to develop more convenient constructs for specifying of the temporal constraints associated with actions. We are currently reformulating our framework into an extended version of the situation calculus, to incorporate a solution to the frame problem described in [Scherl and Levesque, 1993]. We are also developing a more general account of the notions of "ability to achieve a goal" and "knowing how to execute a plan" [Levesque *et al.*, 1994]. Other important issues are how default information could be specified, and identifying restrictions on domain theories and queries that guarantee tractability or decidability.

Acknowledgements

We would like to thank Andy Haas for his comments on the ideas advanced in this paper.

A Outline of the Proof of Proposition 1

The following lemmas are the main steps in proving proposition 1. One proves the proposition by "chaining" these lemmas using the following properties of **Can**:

$$\text{If } \models \varphi_i \supset \mathbf{Can}(\delta_2, \varphi_e), \text{ then } \models \mathbf{Can}(\delta_1, \varphi_i) \supset \mathbf{Can}((\delta_1; \delta_2), \varphi_e).$$

$$\models \mathbf{Can}(\delta, \varphi) \supset \mathbf{Can}(\delta, \mathbf{Know}(\varphi))$$

The first lemma shows that given his initial knowledge, the agent is able to set the turkey to roast:

Lemma 1

$$\models \mathbf{Know}(\neg \text{ROASTING}) \supset \mathbf{Can}(\text{STARTROASTING}, \text{TSR}(\mathbf{now}))$$

The proof uses assumptions 1, 6, and 7.

The second lemma shows that once he has set the turkey to roast, the agent is able to start measuring the roasting time by setting the timer to one hour. Let us first define MRT, meaning that the agent is measuring the roasting time:

$$\text{MRT} \stackrel{\text{def}}{=} \exists t (\text{TSR}(t) \wedge t = (\mathbf{now} - (1\text{H} - \text{TIMERVAL}) - 15\text{S}) \pm 5\text{S})$$

We can then establish that:

Lemma 2

$\models$**Know**(TSR(**now**)) $\supset$
Can(SETTIMER(1H), MRT $\wedge$ TIMERVAL = 1H)

The proof uses assumption 2, the assumptions that SETTIMER is always known and physically possible, the frame assumption for ROASTING, and the unique name assumption for actions.

Then, we show that once he gets in that state, by doing the iterative action LISTENUNTILTIMEANNOUNCED the agent can find out what time it is within 7 seconds of accuracy, with the timer still measuring the roasting time:

Lemma 3

$\models$**Know**(MRT $\wedge$ TIMERVAL = 1H) $\supset$
Can(LISTENUNTILTIMEANNOUNCED,
MRT $\wedge$ 30MIN $-$ 14S $<$ TIMERVAL $\wedge$
$\exists t$**Know**(**now** $= t \pm$ 7S))

To prove this, we need the following two sublemmas. Let us define ATSTS, meaning that the time has been announced on the radio since the timer was set:

ATSTS $\stackrel{\text{def}}{=} \exists t$(**now** $-$ (1H $-$ TIMERVAL) $\leq t \leq$ **now**
$\wedge$ **At**(t, ANNOUNCINGTIME))

The first sublemma states that if the agent knows that he is measuring the roasting time with the timer and that the time has not been announced on the radio since he set the timer, then by doing LISTEN, he is able to either find out that the time has been announced during the action and know what it is (within 7 seconds of accuracy), or know that it has not been announced during the action and since he set the timer, while continuing to measure the roasting time with the timer:

Lemma 4

$\models \forall m($
Know(MRT $\wedge$ 30MIN $<$ TIMERVAL $\leq m \wedge \neg$ATSTS) $\supset$
Can(LISTEN, MRT $\wedge$
([**Know**(ANNOUNCEDTIME) $\wedge \exists t$**Know**(**now** $= t \pm$ 7S)
$\wedge$ 30MIN $-$ 14S $<$ TIMERVAL] $\vee$
[**Know**($\neg$ANNOUNCEDTIME) $\wedge \neg$ATSTS $\wedge$
30MIN $<$ TIMERVAL $\leq m - 6$S])))

This is proven using assumptions 3 and 9, the assumptions that LISTEN is always possible and known, the frame assumption for ROASTING, the assumption that the time is announced at least every thirty minutes, and the unique name assumption. From the above lemma, we can then prove the following by induction over the bound on the number of iterations n:

Lemma 5

For all $n \in \mathbb{N}$,
$\models$ **Know**(MRT) $\wedge$
([**Know**(ANNOUNCEDTIME $\wedge$ 30MIN $-$ 14S $<$ TIMERVAL)
$\wedge \exists t$**Know**(**now** $= t \pm$ 7S)]
$\vee$ **Know**($\neg$ANNOUNCEDTIME $\wedge \neg$ATSTS $\wedge$
30MIN $<$ TIMERVAL $\leq$ 30MIN $+ n$6S)) $\supset$
Can(**while**$_n$($\neg$ANNOUNCEDTIME, LISTEN),
MRT $\wedge$ 30MIN $-$ 14S $<$ TIMERVAL $\wedge$
$\exists t$**Know**(**now** $= t \pm$ 7S))

This says that if the agent either knows that the time has been announced during the previous LISTEN step and knows what it is (within 7 seconds), or knows that it has not been announced during the action and since he set the timer, while measuring the roasting time, then by repeatedly doing LISTEN until the time is announced, he is able to find out what time it is (within 7 seconds) while continuing to measure the roasting time. Lemma 3 is then proven by "chaining" the two results above.

Returning to the proof of the proposition, we then show that once he has found out the time, the agent can find out at what time the turkey started roasting by looking at the timer:

Lemma 6

$\models$**Know**(MRT $\wedge$ 30MIN $-$ 14S $<$ TIMERVAL) $\wedge$
$\exists t$ **Know**(**now** $= t \pm$ 7S) $\supset$
Can(LOOKATTIMER,
$\exists t_m$**Know**($\exists t_r$(TSR(t_r) $\wedge\ t_r = t_m \pm$ 14S $\wedge$
now $- t_r \leq$ 30MIN42S)))

This is shown using assumptions 4 and 9, the assumptions that LOOKATTIMER is always possible and always known, the frame assumption for ROASTING, and the unique name assumption.

Finally, we show that once he has found out when the turkey started roasting, by doing LEAVEMSG, the agent can ensure that there is a message on the table telling when the turkey started roasting (within 14 seconds of accuracy):

Lemma 7

$\models \exists t_m$**Know**($\exists t_r$(TSR(t_r) $\wedge\ t_r = t_m \pm$ 14S $\wedge$
now $- t_r \leq$ 30MIN42S)) $\supset$
Can(LEAVEMSG,
$\exists t_r \exists t_m$(TSR(t_r) $\wedge$ MSGONTBL(t_m) $\wedge\ t_r = t_m \pm$ 14S
$\wedge$ **now** $- t_r \leq$ 32MIN12S))

The proof uses assumptions 5 and 8, the assumption that LEAVEMSG is always physically possible, the frame assumption for ROASTING, and the unique name assumption.

References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Seattle, WA, July 1987. American Association for Artificial Intelligence, Morgan Kaufmann Publishing.

[Goldblatt, 1987] Robert Goldblatt. *Logics of Time and Computation.* CSLI Lecture Notes No. 7. Center for the Study of Language and Information, Stanford University, Stanford, CA, 1987.

[Grove and Halpern, 1991]
Adam J. Grove and Joseph Y. Halpern. Naming and identity in a multi-agent epistemic logic. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 301–312, Cambridge, MA, 1991. Morgan Kaufmann Publishing.

[Haas, 1991] Andrew R. Haas. Indexical expressions and planning. Unpublished manuscript, Department of Computer Science, State University of New York, Albany, NY, 1991.

[Lespérance and Levesque, 1990] Yves Lespérance and Hector J. Levesque. Indexical knowledge in robot plans. In *Proceedings of the Eight National Conference on Artificial Intelligence*, pages 868–874, Boston, August 1990. American Association for Artificial Intelligence, AAAI Press/MIT Press.

[Lespérance and Levesque, 1994] Yves Lespérance and Hector J. Levesque. Indexical knowledge and robot action — a logical account. To appear in *Artificial Intelligence*, 1994.

[Lespérance, 1991] Yves Lespérance. *A Formal Theory of Indexical Knowledge and Action.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, ON, January 1991. Also published as technical report CSRI-248.

[Levesque *et al.*, 1994]
Hector J. Levesque, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Knowledge, action, and ability in the situation calculus. In preparation, 1994.

[Moore, 1980] Robert C. Moore. Reasoning about knowledge and action. Technical Report 191, AI Center, SRI International, Menlo Park, CA, October 1980.

[Perry, 1979] John Perry. The problem of the essential indexical. *Noûs*, 13:3–21, 1979.

[Rosenschein and Kaelbling, 1986] Stanley J. Rosenschein and Leslie P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In Joseph Y. Halpern, editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 83–98, Monterey, CA, 1986. Morgan Kaufmann Publishing.

[Scherl and Levesque, 1993] Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, July 1993. AAAI Press/The MIT Press.

[Shoham, 1987] Yoav Shoham. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33(1):89–104, 1987.

[Subramanian and Woodfill, 1989] Devika Subramanian and John Woodfill. Making the situation calculus indexical. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 467–474, Toronto, ON, May 1989. Morgan Kaufmann Publishing.

Evaluating the Tradeoffs in Partial-Order Planning Algorithms*

Craig A. Knoblock
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
knoblock@isi.edu

Qiang Yang
University of Waterloo
Computer Science Department
Waterloo, Ont., Canada N2L 3G1
qyang@logos.uwaterloo.ca

Abstract

Most practical partial-order planning systems employ some form of goal protection. However, it is not clear from previous work what the tradeoffs are between the different goal-protection strategies. Is it better to protect against all threats to a subgoal, some threats, or no threats at all? In this paper, we consider three well-known planning algorithms, SNLP, NONLIN, and TWEAK. Each algorithm makes use of a different goal-protection strategy. Through a comparison of the three algorithms, we provide a detailed analysis of different goal protection methods, in order to identify the factors that determine the performance of the systems. The analysis clearly shows that the relative performance of the different goal-protection methods used by the systems, depends on the characteristics of the problems being solved. One of the main determining factors of performance is the ratio of the number of negative threats to the number of positive threats. We present an artificial domain where we can control this ratio and show that in fact the planners show radically different performance as the ratio is varied. The implication of this result for someone implementing a planning system is that the most appropriate algorithm will depend on the types of problems to be solved by the planner.

1 Introduction

There has been a great deal of work recently on comparing total and partial order planning systems [Barrett and Weld, 1992; Minton et al. 1991], but little has been done in comparing different partial order planners themselves. There are a variety of design decisions that must be made in order to build a general planner. This paper focuses on one of these design choices – the choice of a protection strategy. In particular, we compare the protection strategy employed in three basic planning algorithms, SNLP, NONLIN and TWEAK.

On the surface, the three planners are quite different. However, on a careful examination one can find that they mainly differ in which conditions they protect. During planning, an inserted plan step can interact with previously inserted steps. If a goal is achieved by one plan step, then later it could be *threatened* by other steps. A goal is *protected* by removing all threats by imposing additional constraints on a plan whenever a threat is detected. Among the three planners, TWEAK protects nothing, NONLIN protects against all negative threats, and SNLP protects against both negative and positive threats.

The use of goal protection in SNLP prevents the planner from generating redundant plans and thereby could potentially reduce the size of the search space. However, enforcing the goal protection has a cost. In this paper, we show that none of the planners is always a winner. In some domains our planner based on TWEAK greatly outperforms both a planner based on NONLIN [1] and SNLP. In other domains, SNLP and NONLIN perform much better than TWEAK. The challenge is to identify the features of the domains where each planner is expected to perform well, so that practitioners can balance the protection methods based on the application domain.

In the following sections, we first review the three algorithms. Then we present an analysis of the algorithms to identify their relative merits. We also report on two critical domain features that have the greatest impact on the performance of the planners. Finally, we present empirical results on an artificial domain to support the analysis.

*The first author is supported by Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency under contract no. F30602-91-C-0081. The second author is supported in part by grants from the Natural Science and Engineering Research Council of Canada, and ITRC: Information Technology Research Centre of Ontario. The views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion or policy of DARPA, RL, NSERC, ITRC, or any person or agency connected with them.

[1] For convenience we will simply refer to them as TWEAK and NONLIN.

2 Comparison of the Algorithms

This section presents the SNLP, TWEAK, and NONLIN planning algorithms. First, we present the SNLP algorithm based on the algorithm descriptions of McAllester and Rosenblitt's Find-Completion algorithm [McAllester and Rosenblitt, 1991] and Barrett and Weld's POCL algorithm [Barrett and Weld, 1992]. We start with this algorithm because we can build on the elegant algorithm description and implementation provided in previous work. Then, we describe the changes necessary to transform the SNLP algorithm into algorithms that implement NONLIN [Tate, 1977] and TWEAK [Chapman 1987].

2.1 The SNLP Algorithm

In the planning algorithms that we consider below, we follow the notations used by Barrett and Weld [Barrett and Weld, 1992]. A plan is a 3-tuple, represented as $\langle S, O, B\rangle$, where S is a number of steps, O is a set of ordering constraints, and B the set of variable binding constraints associated with a plan. A step consists of a set of preconditions, an add list, and a delete list. The binding constraints specify whether two variables can be bound to the same constant or not.

The core of SNLP is the recording of the *causal links* for why a step is introduced into a plan, and for protecting that purpose. If a step S_i adds a proposition p to satisfy a precondition of step S_j, then $S_i \xrightarrow{p} S_j$ denotes the causal link. An operator S_k is a threat to $S_i \xrightarrow{p} S_j$ if S_k can possibly add or delete a literal q that can possibly be bound to p. For convenience, we also refer to the pair $(S_k, S_i \xrightarrow{p} S_j)$ as a threat. In addition, we define an operator S_k to be a *positive threat* to $S_i \xrightarrow{p} S_j$, if S_k can possibly be between S_i and S_j, and S_k adds a literal q that can possibly be bound to p. Likewise, S_k is a *negative threat* if it can possibly be between S_i and S_j, and deletes a literal q that can possibly be bound to p.

The following algorithm which is an adaptation of McAllester and Rosenblitt's Find-Completion algorithm [McAllester and Rosenblitt, 1991] and Barrett and Weld's POCL algorithm [Barrett and Weld, 1992], has been shown to be sound, complete, and systematic (never generates redundant plans). Let the notation codesignate(R) denote the codesignation constraints imposed on a set of variable pairs R. For example, if $R = \{(x_i, y_i) \mid i = 1, 2, \ldots k\}$, then codesignate($R$) = $\{x_i = y_i \mid i = 1, 2, \ldots k\}$. Similarly, noncodesignate(R) denotes the set of non-codesignation constraints on a set R of variable pairs. The parameters of the algorithm are: *S=Steps, O=Ordering constraints, B= Binding constraints, G= Goals, T=Threats, and L=Causal links.*

Algorithm SNLP($\langle S, O, B\rangle, T, G, L$)

1. **Termination:** If G and T are empty, report success and stop.
2. **Declobbering:** A step s_k *threatens* a causal link $s_i \xrightarrow{p} s_j$ when it occurs between s_i and s_j, and it adds or deletes p. If there exists a threat $t \in T$ such that t is a threat between a step s_k and a causal link $s_i \xrightarrow{p} s_j \in L$, then:
   - Remove the threat by adding ordering constraints and/or binding constraints using promotion, demotion, or separation. For completeness, all ways of resolving the threat must be considered.
     - **Promotion:** $O' = O \bigcup \{s_k \prec s_i\}$, $B' = B$
     - **Demotion:** $O' = O \bigcup \{s_j \prec s_k\}$, $B' = B$
     - **Separation:** $O' = O \bigcup \{s_i \prec s_k\} \bigcup \{s_k \prec s_j\}$. Let q be the effect of s_k that threatens p and let P be the set of binding pairs between q and p. $B' = B \bigcup \sigma$, where $\sigma \in \{\alpha \mid \alpha = \text{noncodesignate}(s) \bigcup \text{codesignate}(P - s)$, where $s \subseteq P \wedge s \neq \emptyset\}$.[2]
   - **Recursive invocation:** SNLP($\langle S, O', B'\rangle, T - \{t\}, G, L$)
3. **Goal selection:** Let p be a proposition in G, and let S_{need} be the step for which p is a precondition.
4. **Operator selection:** Let S_{add} be an existing step, or some new step, that adds p before S_{need}. If no such step exists or can be added then backtrack. Let $L' = L \bigcup \{S_{\text{add}} \xrightarrow{p} S_{\text{need}}\}$, $S' = S \bigcup \{S_{\text{add}}\}$, $O' = O \bigcup \{S_{\text{add}} \prec S_{\text{need}}\}$, *and* $B' = B \bigcup$ the set of variable bindings to make S_{add} add p. Finally, update the goal set: $G' = (G - \{p\}) \bigcup$ preconditions of S_{add}, if new. For completeness, all ways of achieving the step must be considered.
5. **Threat identification:** Let $T' = \{t \mid$ for every step s_k that is a positive or negative threat to a causal link $s_i \xrightarrow{p} s_j \in L'$, $t = (s_k, S_i \xrightarrow{p} S_j)\}$.
6. **Recursive invocation:** SNLP($\langle S', O', B'\rangle, T', G', L'$).

2.2 The NONLIN Algorithm

SNLP is a descendant of NONLIN [Tate, 1977], so the algorithms are quite similar and differ mainly in which threats they protect against and how they perform separation. These two differences stem from the added constraints on SNLP that are used to ensure systematicity. NONLIN also provides some additional capabilities such as hierarchical task-network decomposition, but these capabilities are orthogonal to the point of this paper and are not considered.

The first change to the SNLP algorithm is in the threat identification step. In contrast to SNLP, only the negative threats are added to the list T':

Threat identification: Let $T' = \{t \mid$ for every step s_k that is a negative threat to a causal link $s_i \xrightarrow{p} s_j \in L'$, $t = (s_k, S_i \xrightarrow{p} S_j)\}$.

The second change is that to perform separation, there is no requirement that promotion, demotion and separation are made mutually exclusive. In this case, separation simply entails that one or more of the possible bindings are forced not to codesignate, but imposes no ordering constraints.

[2] The possible binding constraints are mutually exclusive, since systematicity requires that the search space is partitioned into non-overlapping parts.

Separation: $O' = O$. Let q be the effect of s_k that possibly codesignates with p and let P be the set of binding pairs between q and p. $B' = B \bigcup \sigma$, where $\sigma \in \{\alpha \mid \alpha = \text{noncodesignate}(e), \text{where } e \in P\}$.

As we will see in the experimental results section, the differences in performance of goal protection methods employed by SNLP and NONLIN are relatively minor.

2.3 The TWEAK Algorithm

The primary difference between TWEAK and the two previous algorithms is that instead of building explicit causal links for each condition established by the planner, TWEAK uses what is called the Modal Truth Criterion [Chapman 1987] to check the truth of each precondition in the plan. This difference results in four changes from the SNLP algorithm and only three changes from the NONLIN algorithm. The differences are in termination, separation, goal selection, and threat identification. Each of these are discussed in turn.

Since TWEAK does not maintain explicit causal links for each precondition, it must test the truth of all of the preconditions in the plan to determine when the plan is complete. It does this using the Modal Truth Criterion check [Chapman 1987]. This algorithm takes $O(n^3)$ time, as compared with the $O(1)$ time termination routine of SNLP. We will refer to the algorithm that implements the Modal Truth Criterion as `mtc`. This algorithm returns true if a given plan is complete and otherwise returns a precondition of some step in the plan that does not necessarily hold.

Termination: If mtc($\langle S, O, B \rangle$) is true, report success and stop.

Similar to NONLIN, there is no requirement that all of the separation constraints are mutually exclusive. Thus, TWEAK uses the same method for separation as NONLIN.

Separation: $O' = O$. Let q be the effect of s_k that possibly codesignates with p and let P be the set of binding pairs between q and p. $B' = B \bigcup \sigma$, where $\sigma \in \{\alpha \mid \alpha = \text{noncodesignate}(e), \text{where } e \in P\}$.

Since TWEAK does not maintain an explicit set of causal links, there is no explicit record of which preconditions much be achieved. Thus, goal section is done using the `mtc` algorithm. The `mtc` returns a precondition of a step in the plan that is not necessarily true.

Goal Selection: Let p be the precondition of step S_{need} returned by the `mtc` procedure.

Finally, unlike both SNLP and NONLIN, TWEAK makes no attempt to protect all of the previously established preconditions against either negative or positive threats. TWEAK does, however, ensure that at each step all negative threats to the most recently built causal link are removed. However, after a precondition is established and threats are removed, it can be clobbered again. In such a case, TWEAK will have to re-establish the condition.

Threat identification: Let $l_{new} = S_{\text{add}} \xrightarrow{p} S_{\text{need}}$, which is the causal link constructed in step 4. Let $T' = \{t | \text{for every step } s_k \text{ that is a } \textit{negative} \text{ threat to } l_{new}, t = (s_k, l_{new})\}$.

As we stated above, the `mtc` routine for the termination check is more expensive than that for SNLP. However, this does not mean that TWEAK is less efficient than SNLP, since in many cases, TWEAK will explore fewer nodes. In the next section, we consider the major factors that affect the search space, and present a complexity analysis of the three algorithms.

3 Analyzing the Algorithms

3.1 Algorithm Complexities

Let eb be the effective branching factor and ed the effective depth of the search tree. In both algorithms, eb is the maximum number of successor plans generated either after step 2, or after step 5, while ed is the maximum number of plan expansions in the search tree from the initial plan state to the solution plan state. Then with a breadth-first search, the time complexity of search is

$$O(eb^{ed} * T_{node}),$$

where T_{node} is the amount of time spent per node.

We next analyze the complexity of the algorithms by fleshing out the parameters eb, ed and T_{node}. In this analysis, let P denote the maximum number of preconditions or effects for a single step, let N denote the total number of operators in an optimal solution plan, and let A be either the SNLP, NONLIN, or TWEAK algorithm.

To expand the effective branching factor eb, we first define the following additional parameters. We use b_new for the number of new operators found by step 4 for achieving p, b_old for the number of existing operators found by step 4 for achieving p, and r_t for the number of alternative constraints to remove one threat. The effective branching factor of search by either algorithm is then

$$eb = \max\{(\ b\_new + \ b\_old_A), r_{t_A}\},$$

since each time the main routine is followed, either step 2 is executed for removing threats, or step 3 –6 is executed to build causal links. If step 2 is executed, r_t successor states are generated, but otherwise, (b_new + b_old) successor plan states are generated.

Next, we expand the effective depth ed. In the solution plan, there are $N * P$ number of (p, S_{need}) pairs, where p is a precondition for step S_{need}. Let f_A be the fraction of the $N * P$ pairs chosen by step 3. For each pair (p, S_{need}) chosen by step 3, step 5 accumulates a set of threats to remove. Let t_A be the number of threats generated by step 5. Finally, let v be the total number of times any fixed pair (p, S_{need}) is chosen by step 3. Then we have

$$ed_A = f_A * N * P * t_A * v_A.$$

A summary of the parameters can be found in Table 3.1.

For SNLP, each pair (p, S_{need}) must be visited exactly once. Therefore, $f_{snlp} = 1$ and $v_{snlp} = 1$. Also, SNLP examines every causal link in the current plan in step 4. Thus, in the average case, the amount of time per node is half of the total number of links in the solution plan, i.e., $N * P/2$. Thus, the average time complexity for SNLP is:

$$O(\max(\ b\_new + \ b\_old_{snlp}, r_{t_{snlp}})^{N*P*t_{snlp}} * N * P).$$

eb	effective *branching* factor
ed	effective search *depth*
T_{node}	average time per node
N	total number of operators in a plan
P	total number of *preconditions* per operator
f_A	*fraction* of (p, S_{need}) pairs examined by algorithm A
v_A	average number of times a (p, S_{need}) pair is *visited* by A
t_A	average number of *threats* found by A at each node
r_{t_A}	average number of ways to *resolve* a threat by A
b_new	average number of *new* establishers for a precondition
b_old	average number of existing (or *old*) establishers for a precondition

Table 1: Parameters used in complexity analysis.

NONLIN's behaviour is similar to SNLP in that each pair (p, S_{need}) must be visited exactly once. Therefore, $f_{nonlin} = 1$ and $v_{nonlin} = 1$. Also similar to SNLP, NONLIN examines every causal link in the current plan in step 4. The difference between NONLIN and SNLP is that NONLIN resolves only negative threats. This means that in general NONLIN will have a smaller t value. The average time complexity for NONLIN is:

$$O(\max(\ b\_new + \ b\_old_{nonlin}, r_{t_{nonlin}})^{N*P*t_{nonlin}} * N * P)$$

In TWEAK, $f_{tweak} \leq 1$, and can be much smaller than one since TWEAK does not build explicit causal links for every precondition. If many preconditions already hold, then the number of chosen preconditions by step 3 in TWEAK could be much smaller than the total number of preconditions in the solution plan. Since TWEAK does not protect any past causal links, a precondition can be visited twice. Therefore, $v_{tweak} \geq 1$. t_{tweak}, on the other hand, should be much smaller than t_{snlp} and t_{nonlin}, since TWEAK only declobbers for the most recently constructed causal link, and only negative threats are considered. Thus the number of threats is much smaller. Finally, TWEAK uses MTC to check the correctness of a plan, resulting a complexity per node to be $O((N*P)^3)$. Overall, the complexity of TWEAK is:

$$O(\max(\ b\_new + \ b\_old_{tweak}, r_{t_{tweak}})^m * T_{tweak}$$

where $m = f_{tweak} * N * P * t_{tweak} * v_{tweak}$ and $T_{tweak} = (N * P)^3$.

In the next section, we discuss how these parameters change with certain domain features.

3.2 Systematicity

SNLP is systematic, which means that no redundant plans are generated in the search space. In contrast, neither TWEAK nor NONLIN are systematic. However, a planner that is systematic is not necessarily more efficient. The systematicity property reduces the branching factor by avoiding redundant plans. However, systematicity is achieved in SNLP by protecting against both the negative and positive threats, which increases the factor t, a multiplicative factor in the exponent. Thus, SNLP reduces the branching factor at a price of increasing the depth of search. Therefore, one can get a systematic, but less efficient planning system.

4 Domain Features and Search Performance

The analysis in the previous section can be used to predict the relative performance of the three planning algorithms in different types of domains. An important feature of a domain that determines the relative performance of any two algorithms is the *ratio* between the *number of positive threats* and *number of negative threats*. The ratio is an important factor in differentiating the algorithms because the major difference between any two algorithms is the way they handle positive and negative threats. Among the three algorithms, TWEAK only avoids some negative threats, SNLP protects against all positive and negative threats, and NONLIN protects against all negative threats but not the positive ones.

4.1 Predictions

The major difference between the algorithms manifest themselves in the execution of Step 1, the termination subroutine, and Step 4, threat detection. To see their effect on search efficiency, let t_+ denote the average number of positive threats, and let t_- be the average number of negative threats detected by Step 4 of SNLP. Let R denote the ratio of t_- to t_+: $R = \frac{t_-}{t_+}$. In this section we predict the performance of the three planning algorithms based on the value of R.

Case 1: $R \ll 1$

Since SNLP resolves all positive threats, it imposes more constraints on a plan. Thus, on the average an SNLP plan is more linearly ordered than either a TWEAK plan or a NONLIN plan. A more linearly ordered plan has a smaller number of existing establishing operators for a given precondition, and thus a smaller branching factor. Thus, the branching factor of SNLP is likely to be the smallest among the three, and that for TWEAK is the largest due to its conservative stand in resolving threats.

When t_+ is relatively large, the total number of threats t resolved by SNLP is large, which in turn increases SNLP's search depth. Also, for both NONLIN and SNLP, a causal link has to be built for every precondition in a plan, a behavior that fixes a lower bound on their search depths. With many positive threats in a plan, a

precondition is more likely to be achievable by an existing step. Therefore TWEAK will be able to skip many more preconditions compared to NONLIN and SNLP. Thus the search depth of TWEAK will be much less than both NONLIN and SNLP, and the search depth of NONLIN will be smaller than SNLP because it does not resolve positive threats.

As R decreases below one, the branching factor for TWEAK and NONLIN increase, while the search depth for SNLP increases. The time complexity for the former go up polynomially, while for the latter it goes up exponentially. Moreover, the depth of NONLIN is greater than the depth of TWEAK. Therefore, we predict that when $R \ll 1$ TWEAK will perform better than NONLIN, which in turn will perform better than SNLP.

Case 2: $R \approx 1$

As with the previous case, the additional constraints imposed by SNLP and NONLIN over TWEAK imply that SNLP will have a smaller branching factor then NONLIN, and NONLIN will have a smaller branching factor than TWEAK. However, the difference in the number of threats t resolved by TWEAK, SNLP, and NONLIN will be reduced since there are fewer positive threats and more negative threats. The reduced number of positive threats will reduce the depth for SNLP and NONLIN and the increased number of negative threats increases the chance that TWEAK will be forced to revisit the same precondition/step pair. As a result, the performance of the different planners could be very close and will depend on depth and branching factors for the problems being solved.

Case 3: $R \gg 1$

TWEAK is likely to have the largest branching factor because every time a negative threat occurs, all existing and new operators are considered as establishers again. This effect increases the factor *b_old* for TWEAK, resulting in the effective branching factor for TWEAK being greater than both SNLP and NONLIN. Also due to its resolution of positive threats, a SNLP plan is likely to be more linearized than a NONLIN plan, thus the branching factor of SNLP will be smaller than NONLIN.

Each negative threat creates a chance for TWEAK to revisit the same precondition/step pair. Since in the $R \gg 1$ case, there is a large number of negative threats, the number of times each precondition is visited, v_{tweak}, is likely to increase. Since TWEAK is expected to have a larger branching factor and depth greater than both SNLP and NONLIN, when $R \gg 1$ TWEAK is expected to perform the worst. SNLP will outperform NONLIN slightly due its smaller branching factor.

4.2 Empirical Results

In order to verify our predictions by comparing SNLP, NONLIN and TWEAK on problems with different ratios of negative and positive threats, we constructed an artificial domain where we could control the value of R. In this domain, each goal can be achieved by a subplan of two steps in a linear sequence. Each step either achieves a goal condition or a precondition of a later step. The preconditions of the first step always hold in the initial state. In addition, we also added extra operator effects to create threats in planning. The difficulty of the problems in this domain can be increased by increasing the number of goal conditions and the total number of threats.

(defstep :action A_{i1} :precond I_i :equals {}
:add $\{P_i; I_{i+1}$ *if* $i < n_+; I_0$ *if* $i = n-1$ *and* $n_+ > 0\}$
:delete $\{I_{i-1}$, *if* $0 < i < n_-; I_{n-1}$ *if* $i = 0$
and $n_- > 0\}$)

(defstep :action A_{i2} :precond P_i :equals {}
:add $\{G_i; P_{i+1}$ *if* $i < n_+; P_0$ *if* $i = n-1$ *and* $n_+ > 0\}$
:delete $\{P_{i-1}$, *if* $0 < i < n_-; P_{n-1}$ *if* $i = 0$
and $n_- > 0\}$)

We used this artificial domain to run a set of experiments to compare the performance of the different planners. In these experiments we simultaneously varied the number of positive and negative interactions, such that the total number of interactions remained the same, but the ratio R changes from zero to infinity; the number of negative interactions increased from 0 to 9 while the number of positive interactions decreased from 9 to 0. Below, we present the results of our empirical tests on different points of the spectrum of as defined by the ratio R.

In the experiments, each problem was run in SNLP [Barrett and Weld, 1992], a version of NONLIN and a version of TWEAK that were modified from SNLP. The problems were solved using a best-first search on the solution size in order to fairly compare the size of the problem spaces being searched by each system. All the problems were run on a SUN IPC in Lucid Common Lisp with a 120 CPU second time bound. For each value of ratio R, we ran the systems on 20 randomly generated problems. The points shown in the graphs below are an average of the 20 problems.

4.2.1 Branching Factor

The branching factor results are shown in Figure 4.2.1. Most of our predictions for branching factors are observable in the figure. For example, due to its conservative stand in resolving both positive and negative threats, SNLP imposes the most constraints onto a plan, and as a results it generally has the lowest branching factor. Also, as the number of negative threats increases, which constrains the possible plans, the branching factor decreases to one.

However, there are a few surprises shown in the figure. When $R \ll 1$, we had predicted that TWEAK would have a larger branching factor than SNLP and would be similar to NONLIN. This prediction cannot observed from the figure. In order to explain this effect we have broken the branching factor into the two parts described in the analysis, the establishment branching factor and the declobbering branching factor, which are combined to form the overall branching factor. These graphs are shown in Figures 4.2.1 and 4.2.1. As shown in the graphs, the smaller than expected branching factor for TWEAK is due to a smaller than expected establishment branching factor.

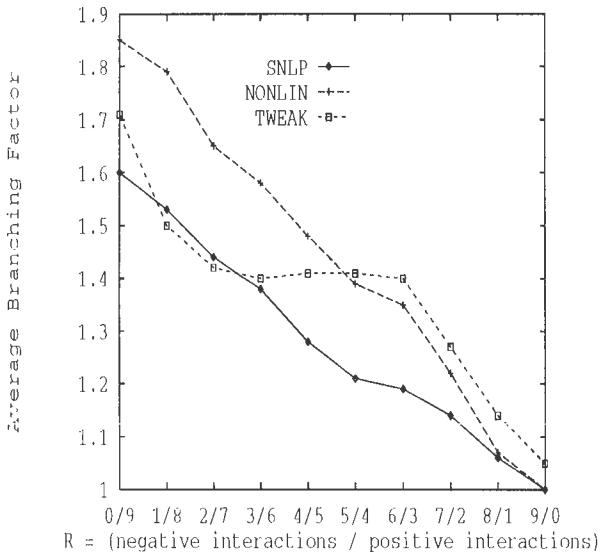


Figure 1: Comprison of the Average Branching Factor of each of the Algorithms

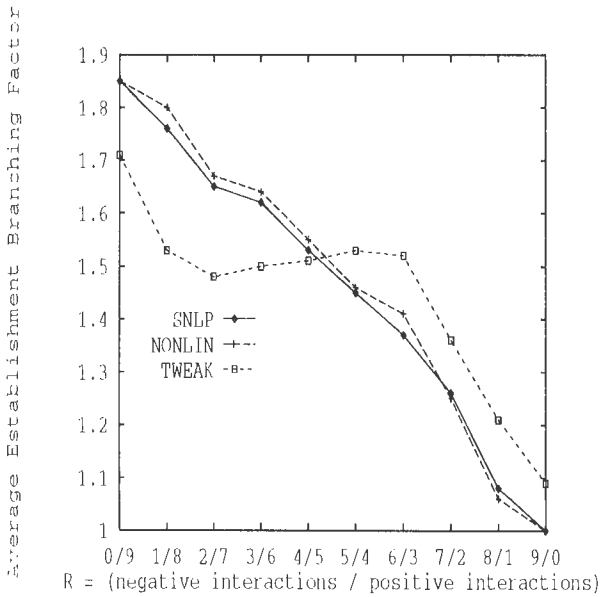


Figure 2: Comprison of the Average Establishment Branching Factor of each of the Algorithms

Careful analysis of the data shows that this discrepency with the predictions is due to the assumption that the branching factor is uniform across an entire problem-solving episode. In fact, where there are many positive interactions, TWEAK quickly narrows in on a plan and reduces the establishment branching factor. In contrast, because both SNLP and NONLIN build explicit causal links and resolve more threats they spend more time in the early plan formation stage when the branching factor is higher. Thus overall, SNLP and NONLIN expand a larger part of the search space that has a large branching factor, while TWEAK uses it ability to exploit positive threats to rapidly traverse that part of the search space.

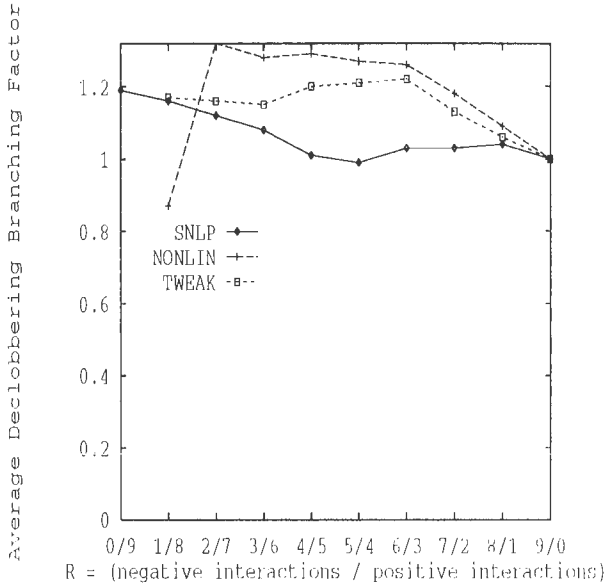


Figure 3: Comprison of the Average Declobbering Branching Factor of each of the Algorithms

4.2.2 Depth

The comparison of the search depths is shown in Figure 4 and they are as predicted. The only apparent discrepency is that the difference between SNLP and TWEAK should be larger when $R \ll 1$. However, the graph is a bit misleading in this case because it includes problems that could not be solved within the time bound by NONLIN and SNLP and so it underestimates their search depth.

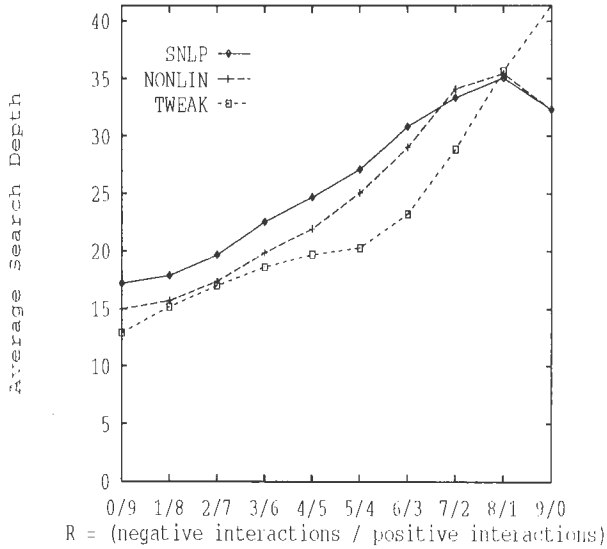


Figure 4: Comprison of the Average Depth of each of the Algorithms

The overall search depth is composed of a number of factors described in the analysis, which includes the fraction of the preconditions considered, the average number of times each precondition is visited,a and the average

number of threats detected by each algorithm. Figure 5 shows the fraction of preconditions considered. This number should be one for both SNLP and NONLIN but again the graphs are distorted by the fact that these two systems did not complete all of the problems within the time limit. In that case, there are a number of preconditions of operators that had not yet been considered. Note that for most of the problems, TWEAK only expanded roughly 60-80% of the preconditions and as the problems had fewer positive interactions, it was forced to expand more and more of the preconditions.

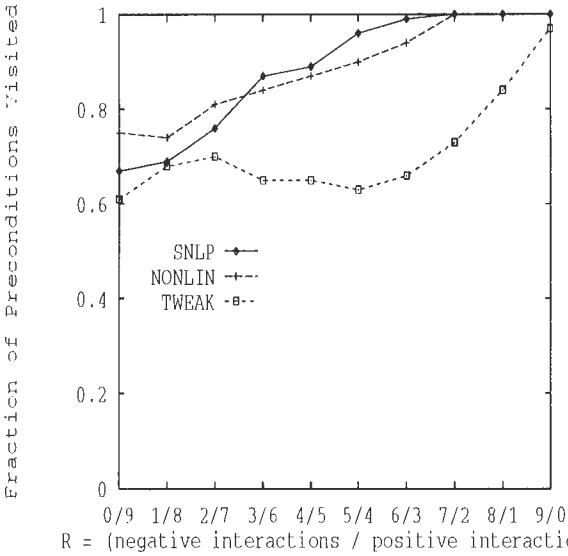


Figure 5: Comprison of the Average Fraction of Preconditions Considered by each of the Algorithms

Figure 6 shows the average of number of times each precondition is visited. As predicted, SNLP and NONLIN visit every precondition exactly once, while TWEAK visits some preconditions more than once. As the number of negative interactions increase, the value for TWEAK increases because it does not protect the conditions that have already been achieved.

Figure 7 shows the average number of threats detected by each of the systems. The fact that SNLP detects a much larger number of threats than both NONLIN and TWEAK comes as no surprise. However, the fact that the number of threats detected by NONLIN is less than the number detected by TWEAK when $R \ll 1$ was not predicted by the analysis. This appears to be due to the fact that the negative threats that NONLIN protects against impose additional ordering constraints on the plan and a more linearly ordered plan has fewer potential threats.

4.2.3 Average CPU Time

The average CPU time for solving problems in the artificial domain is shown in Figure 8. The result fits exactly with our predictions. One thing to note is that no system performs absolutely the best throughout the entire spectrum defined by R. Another is that although NONLIN did well as compared to SNLP when R is small, it is never significantly better than SNLP. In the case where

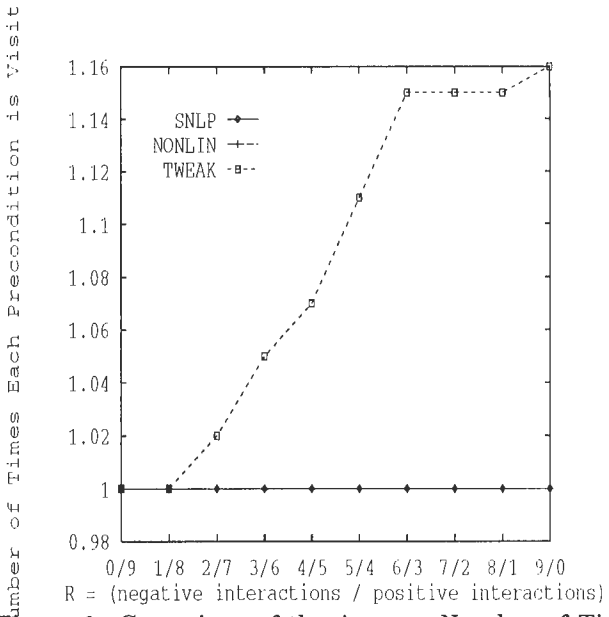


Figure 6: Comprison of the Average Number of Times each Precondition is Visited by each of the Algorithms

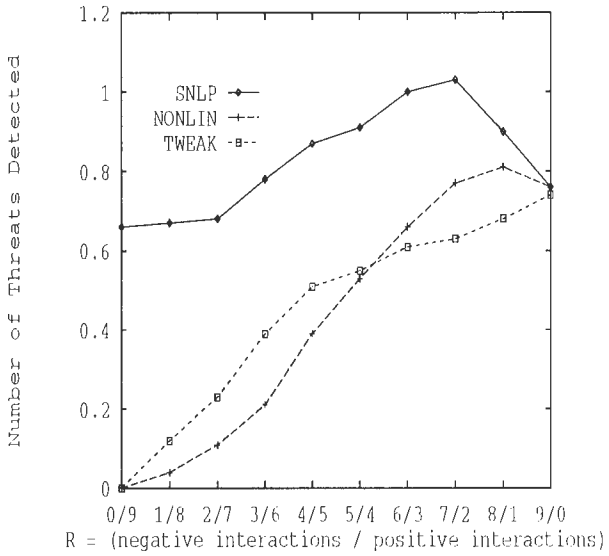


Figure 7: Comprison of the Average Number of Threats Detected by each of the Algorithms

it does outperfrom SNLP it is dramatically worse than TWEAK. This effect should lend credibility to the protection against positive threats as used in SNLP. Although protection of positive threats seemed clumsy when R is small, when the number of negative threats is relatively large the protection method used by SNLP imposes more constraints on a plan. The resulting plans in SNLP's search space are more linear due to the additional constraints. The computational advantage of dealing with a more linear plan compansates for the loss of efficiency due to the protection of positive threats.

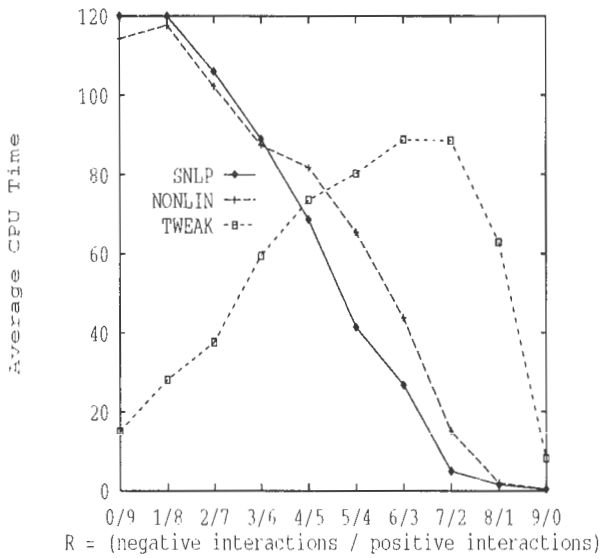


Figure 8: Comprison of the Average CPU Time of each of the Algorithms

5 Related Work and Conclusions

As we stated in the introduction, little work has been done on comparing different partial order planners. An exception is the work by Kambhampati[Kambhampati, 1993; Kambhampati, 1992], who (concurrently with our work) carried out a set of experiments to test the merits of different partial-order planners. In that work, a pair of partial-order planners MP and MP-I are proposed that build upon SNLP and NONLIN by making use of multiple contributors to achieve a precondition. Experiments in a set of closely related domains were conducted, and the resulting comparison of SNLP, NONLIN, TWEAK, MP, and MP-I show that MP-I outperforms all of the rest, and that NONLIN in one test performed much better than both SNLP and TWEAK(Figure 8, [Kambhampati, 1993]).

Contrasting Kambhampati's results to ours, we note that the former is based on a fixed domain. Our results clearly demonstrate that varying the ratio R of positive to negative threats experienced by a planner, almost any comparison result can be obtained; when $R \ll 1$ the comparison results should be dramatically different from that when $R \gg 1$. Thus, it is not surprising that one can find a domain, with a specific R value, where SNLP and/or TWEAK perform worse than NONLIN. From this perspective, the work by Kambhampati can be seen as orthogonal to ours; while we search for domain features by which to determine the relative performance of each system, Kambhampati looks for the best planner on a single point in the spectrum of features.

In summary, we have presented a detailed comparison of the goal protection strategies used in the SNLP, NONLIN, and TWEAK planning algorithms. The analysis provides a foundation for predicting the conditions under which different planning algorithms will perform well. As the results show, SNLP and NONLIN performs better than TWEAK when the ratio of negative threats to positive threats is large, and TWEAK performs significantly better than SNLP and NONLIN in the opposite case. The implications of these results for someone building a practical planning system is that the most appropriate goal protection strategy depends on the characteristics of the problem being solved. This paper provides an important step in building useful planners by identifying a feature of planning domains that has a major impact on the performance of different planning algorithms.

References

[Barrett and Weld, 1992] Anthony Barrett and Dan Weld. Partial order planning: Evaluating possible efficiency gains. Technical Report 92-05-01, University of Washington, Department of Computer Science and Engineering, 1992.

[Chapman 1987] David Chapman Planning for Conjunctive Goals. Artificial Intelligence, volume 32, pp. 333-377, 1987.

[Korf, 1987] Korf, R.E., "Planning as Search: A Quantitative Approach," *Artificial Intelligence* (33), 1987, 65-88.

[Oren et al., 1992] Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. submitted for publication, University of Washington, Department of Computer Science and Engineering, 1992.

[McAllester and Rosenblitt, 1991] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th AAAI*, Anaheim, CA, 1991.

[Minton et al. 1991] Steve Minton, John Bresina, and Mark Drummond. Commitment strategies in planning: A comparative analysis. In *Proceedings of the 12th IJCAI*, Sydney, Australia, 1991.

[Pednault, 1986] Edwin P.D. Pednault Toward a Mathematical Theory of Plan Synthesis. Ph.D. Thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, 1986.

[Kambhampati, 1992] Subbarao Kambhampati. Multi-Contributor Causal Structures for Planning: A Formalization and Evaluation. Arizona State University, technical report ASU-CS-TR-92-019, July 1992.

[Kambhampati, 1993] Subbarao Kambhampati. On the Utility of Systematicity: Understanding Tradeoffs between Redundancy and Commitment in Partial-ordering Planning. *Proceedings of 13th IJCAI*, Chambery, France, 1993, 1380-1387.

[Tate, 1977] Austin Tate Generating Project Networks. IJCAI77, pp. 888-893, 1977.

[Yang et al. 1991] Qiang Yang, Josh Tenenberg, and Steve Woods. Abstraction in nonlinear planning. University of Waterloo Technical Report CS 91-65, 1991.

Indicative and Action Planning for an Intelligent Agent

Gregory E. Kersten [1], **Ping Lu** [1], **Stan Szpakowicz** [2]

[1] School of Business
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
{gregory, lping}@business.carleton.ca

[2] Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada, K1N 6N5
szpak@csi.uottawa.ca

Abstract

We propose a generalization of reactive planning, motivated by economic planning and decision-theoretic principles. A two-level architecture consists of an action planner and an indicative planner. The indicative planner assesses the current situation and the history and outcomes of actions to determine goals and behaviour for the action planner. Action planning produces actions within the framework given by indicative planning. This planning architecture, expressed in the formalism of the Negoplan knowledge-based decision analysis and simulation system, is discussed and illustrated by an example of a robot carrying out a planetary mission.

KEYWORDS

Reactive planning, behavioural planning, simulation, restructurable modelling, knowledge-based systems.

1. Introduction

Classical planning is unsuitable for spatially distributed, unpredictable, dynamic domains. Attempts to overcome this limitation include interleaving the processes of plan formation and execution [Ambros-Ingerson and Steel, 1988], applying temporal reasoning with its logic to a formalization of temporal events [Allen, 1991] and dealing with new requirements for the representations that would help model the increasingly complex domains. Since none of these seem to offer an entirely satisfactory solution, reactive planning has been proposed [Agre and Chapman, 1987; Blythe and Reilly, 1993; Godefroid and Kabanza, 1991; Kaelbling, 1986; Schoppers, 1987].

We dedicate this article to the memory of our long-time friend and collaborator Zbig Koperczak.

This work has been supported by a strategic grant from the Natural Sciences and Engineering Research Council of Canada.

Reactive planning is based on the use of real-time mechanisms that generate and modify plans for working in a dynamic environment. After gathering new observations, a reactive planner can immediately generate a new plan to cope with the new situation [Dornand and Hommel, 1991]. Another direction, *replanning*, follows the classical planning paradigm, with the additional flavour of interleaving planning and execution in order to gain reactivity [Kambhampati, 1992]. A plan is constructed, but it is constantly revised. Reactivity is gained by monitoring execution.

The general idea of reactive planning is that an action is justified more by the situation and less by what it is expected to achieve. Actions are determined by behaviour that guides the agent towards the goal. The question we address here is whether behaviour can and should be planned.

Brooks [1986, 1991] proposed decomposing the overall problem into task-achieving units that realize distinct behaviour. His system is a layered architecture using situation-reaction rules for deriving actions directly from the sensed environment. Kaelbling [1986] proposed an interesting hybrid architecture based on an idea similar to Brooks's; it uses the Rex language to specify the control system. Agre and Chapman [1987] proposed a scheme whereby a highly reactive system can encode reactive behaviour by exploring plan representations.

The reactivity of PRS [Georgeff and Lansky, 1987] is driven by decomposing goals activated by a rule into primitive actions like a classical planner. PRS also uses "metareasoning" during execution to recognize problems that cause additional planning. The RAP architecture [Firby, 1992] consists of three layers: a planner produces sketchy plans for goals as they come in, the RAP execution system fills in the details of the sketchy plan at run time, and a control system actually carries out actions in the world. Hap, developed at Carnegie Mellon University, is a reactive system that uses pre-defined plans with reactive annotations (the success-test and context-conditions) to achieve goals [Loyall and Bates 1991; Blythe and Reilly 1993].

McDermott's RPL [1991; 1992a, 1992b] aims at using reactive plans in a simulated world. It contains "local variables, loops, multiple processes, interrupts, and several

other features" to meet the requirements of "being flexible to control a realistic robot", and "being transparent enough so that the planner can reason about the execution of plans and see ways to improve them". These systems all focus on implementation of behaviour-driven and situation-based actions.

Kersten *et al.* [1994] propose the negotiation metaphor in a simulation of an agent's behaviour in an unknown, changing milieu. The milieu is viewed as an opponent that reacts to the agent's actions or acts independently of the agent. In this paper we discuss the use of Negoplan [Kersten *et al.* 1991] as a tool for robot motion planning. A planner expressed in Negoplan allows us to develop reactive plans and to modify the agent's goals and behaviour according to the current, past, and predicted states of the environment. Negoplan provides a two-level planning architecture for responding to new situations during plan execution and modifying its goals and intentions accordingly.

2. Two-level planning architecture

A plan is a sequence of actions for reaching desired goals. It is not realistic to complete planning before an autonomous agent acts in a changing and unpredictable environment. We may first determine behaviour required for the agent to achieve its goals and then use this behaviour to guide its actions. Following Shu [1990] and Blythe and Reilly [1993] we propose a two-level planning architecture. Our architecture, however, has its roots in economic planning and decision theory.

In economic planning one talks about directive versus indicative planning. Directive planning is concerned with a set of specific directives or actions. A firm in a decentralized planned economy is given a set of indicator levels (value of production, total wage level, profit) and it is free to determine its activities within the framework given by the economic indicators. The assumption is that the firm will normally face unpredictable situations or situations too complex to be detailed *a priori*. Instead of combining specific actions, the indicative planner gives a set of indicators that are used to determine behaviour. Next, the action planner determines actions for a given behaviour and environment.

The two-level planning architecture is illustrated in Fig. 1. Behaviour that is an outcome of the indicative planner is an input of the action planner. *Indicative planning* specifies behaviour for an agent. Behaviour is a set of decision criteria, aspirations, goals, bounds and restrictions. *Action planning* produces a structure of actions within the framework given by the indicative planner. The indicative planner may also define new or temporary desired state (Current Desired State in Fig. 1) that the agent needs to achieve before the overall (ultimate) desired state can be achieved.

Note that the action planner need not know the expected results of a plan and the desired state. It only controls the agent's fast and intelligent reactions in a dynamic environment under the guidance of behaviour generated by the indicative planner. If the situation changes so that a change of the agent's behaviour is required, the indicative planner modifies the conditions in which the action planner operates.

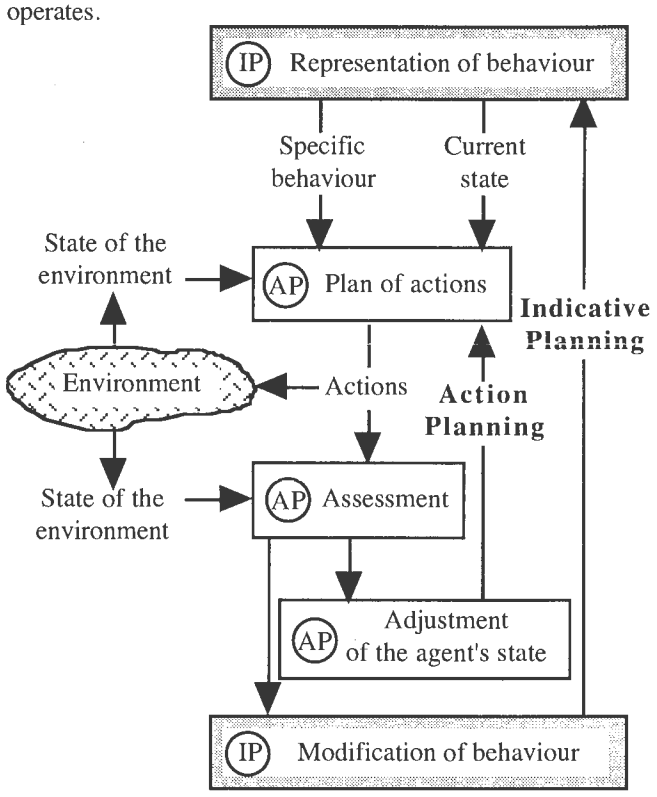


Figure 1. Indicative planning (IP) and action planning (AP) processes in Negoplan

Such specialization of agents resembles the model decision maker and model decision problem proposed in decision theory [French, 1986]. The model decision maker represents behaviour of the agent and the model decision problem represents the planning problem. It is also similar to strategic and tactical planning in a company. This is also a generalization of reactive planning which only considers generation of feasible actions and choice for one step ahead and for a given type of behaviour.

3. Negoplan—an outline

Negoplan is a method and a software tool that supports the solution of complex, dynamic, sequential decision making problems. It is based on the concept of restructurable modeling that allows one to modify the agent's behaviour in response to actions external to the agent [Kersten *et al* 1991; Matwin *et al.* 1989]. The Negoplan system has been used to model and support negotiations [Kersten and Michalowski 1989; Kersten *et al* 1990], and simulate sequential decision processes of intelligent agents [Kersten *et al.* 1993; 1994]. It is usually assumed that the supported or simulated agent makes decisions in a dynamic environment and distinguishes a separate entity with which it must cooperate or negotiate in order to succeed.

Negoplan's rule-based formalism represents the interacting entities. The agent has a hierarchy of goals or an internal hierarchical organization represented by a graph that we call *goal representation* (GR). A node of this graph is described by a *decomposition rule* that relates a goal to subgoals or an element of the agent's structure to substructures. Constraint propagation is used to determine a *problem solution*—a hierarchical decomposition of the decision problem; it is represented by a sub-graph of GR that we call *goal solution* (GS). Each node in this graph is represented by a predicate and given a logical value (true or false) that represents its status.

A set of *metafacts* is derived from GS. They define the agent's decision. A metafact is a predicate given a logical value and labeled by a symbol denoting one of the sides, participants in the decision process.

The process of decision making or simulation is modeled in Negoplan by means of *metarules*. *Response metarules* describe the participants' reactions to various elements of the situation represented by metafacts. A response metarule consists of *triggers*—a list of metafacts (describing elements of the current or past states) and embedded calls, and *conclusions*—a list of metafacts (describing actions and reactions) and embedded calls. An *embedded call* invokes a procedure external to the hierarchical decomposition of the problem; it is used to determine values of parameters or perform tests.

Response metarules change the current set of metafacts. New metafacts change the problem's solution chosen by the agent and define the state of the other agents and the environment. The agent must consider these changes and actions to make a new decision. The outcomes of the earlier decisions and exogenous actions of the participants often make the previously defined decision problem inadequate to the current situation. The supported agent can transform the problem representation GR to account for the new elements of the situation. This is modelled by *restructuring metarules* that add or replace rules in GR. A restructuring metarule may introduce new goals into the current problem representation.

Negoplan offers a rich structure for problem representation, sequencing, conditionals, adjustments and restructuring. We use the following constructs to model the robot's behaviour and actions in reactive planning:

- states, goals, information about the world, and attributes of behaviour and action outcomes represented with metafacts,
- internal structure of the robot GR represented with rules,
- behavioural mechanisms: recording behaviour in a GR and modifying behaviour by restructuring metarules,
- response metarules that convert behaviour into actions, analyze the situation of the environment and results of the actions,
- adjustment metarules that update the current state of the agent (currently represented with restructuring metarules),
- embedded calls to external procedures that simulate the environment and determine parameter values.

4. A robot and its exploratory mission

4.1. The mission

Consider a robot that moves through a surface represented by a grid with integer coordinates. Its initial position is at point (0,0) and it must move to point (8,8). At the beginning of planning the robot is unclear about the environment—positions of rough paths and randomly appearing obstacles in the grid. Movement requires energy and time: traveling one step on a smooth surface requires 0.5 energy unit and 0.5 time unit; traveling on a rough surface requires 1 energy unit and 1 time unit. If the robot has insufficient energy to arrive at the desired position, it will go to a near refuelling station to replenish energy. The goal of robot motion is to arrive at the desired position and also pick up some samples. The robot knows locations of energy supply stations and sample stations.

Reactive planning may require temporary or permanent changes of choice criteria, goals and desired states. We view these as part of an agent's behaviour. Six typical kinds of behaviour are used to determine actions of the robot:

- the shortest path criterion,
- the obstacle avoidance requirement,
- the save-time criterion,
- the save-energy criterion,
- the maximize samples taken criterion,
- the temporary goal "go to an energy station".

Behaviour of various kinds can be combined. For example, we may have a combined behaviour of moving towards a desired position without hitting anything.

4.2. Initial goal representation and the environment

The planning problem is represented as a graph GR that is constructed from node descriptions. GR rules take the form:

```
goal <- subgoal1 & ... & subgoaln.
```

The initial GR of the robot is as follows:

```
robot <-
   states & static_environ_info &
   attributes & behaviour & history.

states <-
   present_position(0, 0) &
   desired_position(8, 8).
```

```
static_environ_info <-
   energy_station([t(2,6), t(6,5), t(4,1)]) &
   sample_station([t(1,3), t(1,5), t(3,4),
                   t(5,3), t(7,2), t(7,5)]).

attributes <-
   energy_left(8.0) & time_left(8.0).

behaviour <-
   criterion1(gather_sample) &
   criterion2(short_path).

history <-
   history_record(0.0, 0.0, [t(0,0)]) &
   start.
```

The robot's behaviour is determined by two criteria, `criterion1(gather_sample)`, `criterion2(short_path)` and the final goal `desired_position(8, 8)`. The robot must consider both criteria: gather as many samples as possible and arrive at the desired position in the shortest time.

The environment is represented by a grid illustrated in Fig. 2. Samples are located at six points (dots). There are also three energy stations (rectangles). These are the static elements of the environment known to the robot before the mission. The dynamic and unpredictable elements are obstacles that may appear anywhere; the static but unknown element is the type of the surface (smooth or rough).

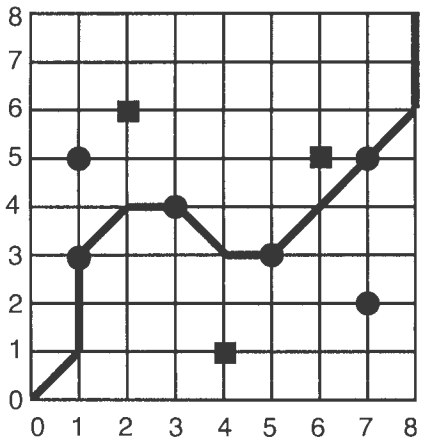


Figure 2. A path for gathering samples and arriving at the desired position

In Fig. 2 we show one of the shortest paths for the robot moving from point (0,0) to point (8,8) that allows it to pick the sufficient number of samples. This plan is developed for an ideal state of the environment. The robot does not pick samples located at points (1,5) and (7,2) because they are too far from the shortest path between (0,0) and (8,8).

Planning is a process of determining actions to achieve a goal (desired position) given current states, behaviour and environment information. Static environment information is recorded in the lists, parameters of predicates. For example:

```
energy_station(List_energy),

sample_station(List_sample).
```

The problem solver estimates and evaluates outcomes of behaviour and actions. Attributes of these outcomes include time and energy consumed by executing actions. The predicate `history(Eh, Th, Lish)` is used to record how much energy and time has been used for moving from the initial position to the current point `t(X, Y)`. The important information such as the starting position, the visited sample and energy stations is recorded in the list `Lish`. The predicate `start` indicates the start of the planning process. The initial behaviour of the robot is guided by the criteria of following the shortest path and gathering as many samples as possible.

4.3. Planning behaviour and modifying goals

The indicative planner defines the required behaviour and provides conditions required to determine actions. Behaviour is dynamically modified according to changes in the situation and the environment. The modification of the behaviour is done by restructuring metarules of the form:

```
LHS ==> modify ( new_rules ).
```

Elements of the left-hand side contain information about the current state of the world as well as the state of the search process. The right-hand side contains decomposition rules that are introduced into the current GR.

For example, the robot changes its criterion from `short_path` to `save_energy` if it does not have enough energy. This criterion causes it to avoid rough surfaces.

```
robot: criterion1(C1) ::= true &
robot: criterion2(short_path) ::= true &
robot: present_position(X1, Y1) ::= true &
robot:
  current_desired_position(Xd, Yd) ::= true &
robot: desired_position(Xdd, Ydd) ::= true &
robot: energy_left(E1) ::= true &
robot: time_left(T1) ::= true &
robot: energy_station(List_energy) ::= true &
robot: sample_station(List_sample) ::= true &
robot:
  history_record(Eh1, Th1, Listh1) ::= true &
{ predict(Eh1, Th1, Listh1, t(X1, Y1),
          t(Xd, Yd), Ef1, Tf1),
  E1 < Ef1, T1 < Tf1
}   % not enough energy and time left
==>
modify (
    states <-
       present_position(X1, Y1) &
       current_desired_position(Xd, Yd) &
       desired_position(Xdd, Ydd),
    static_environ_info <-
       energy_station(List_energy) &
       sample_station(List_sample),
```

```
    attributes <-
       energy_left(E1) & time_left(T1),
    behaviour <-
       criterion1(C1) &
       criterion2(save_energy),
    history <-
       history_record(Eh1, Th1, Listh1) &
       start_action_planning &
       action_index(1) ).
```

A prediction mechanism with the ability to learn from past experience is built into the embedded call `predict(Eh, Th, Listh, t(X, Y), t(Xd, Yd), Ef, Tf)`. According to the energy `Eh` and time `Th` used in moving from the original point `(0,0)` to the current point `(X, Y)`, the robot predicts the amount of energy `Ef` and time `Tf` needed to move from the current point `(X, Y)` to the desired point `(Xd1, Yd1)` using the following formulæ:

$$Ef = \alpha * Eh * \frac{\text{shortest\_distance\_to\_goal}}{\text{shortest\_distance\_in\_past}},$$

$$Tf = \beta * Th * \frac{\text{shortest\_distance\_to\_goal}}{\text{shortest\_distance\_in\_past}},$$

where α, β are normalization coefficients.

If the robot has insufficient energy to arrive at the goal, moving to a nearby energy station to refuel should be considered. In this case, the current goal is changed.

```
robot: criterion1(C1) ::= true &
robot: criterion2(C2) ::= true &
robot: present_position(X, Y) ::= true &
robot:
  current_desired_position(Xd, Yd) ::= true &
robot: desired_position(Xdd, Ydd) ::= true &
robot: energy_station(List_energy) ::= true &
robot: sample_station(List_sample) ::= true &
robot: energy_left(E1) ::= true &
robot: time_left(T1) ::= true &
robot:
  history_record(Eh1, Th1, Listh1) ::= true &
{ C1 \== get_energy,
  predict(Eh1, Th1, Listh1, t(X, Y),
          t(Xdd, Ydd), Ef1, Tf1),
  E1 < Ef1,
  T1 < Tf1, % not enough energy and time left
  closest_point(t(X, Y), List_energy,
                t(Xe, Ye), t(Xdd, Ydd))
}
==>
modify (
    states <-
       present_position(X, Y) &
       current_desired_position(Xe, Ye) &
       desired_position(Xdd, Ydd),
    static_environ_info <-
       energy_station(List_energy) &
       sample_station(List_sample),
    attributes <-
       energy_left(E1) & time_left(T1),
    behaviour <-
       criterion2(save_energy) &
       criterion1(get_energy),
    history <-
       history_record(Eh1, Th1,
                      [t(Xd, Yd) | Listh1]) &
       start_action_planning &
       action_index(1) ).
```

After replenishing energy the planning process is restarted from the current position

A different kind of metarules are used to end the mission. For example, the following metarule terminates a successful mission:

```
robot: present_position( X, Y ) ::= true &
robot: desired_position( X, Y ) ::= true
==>
terminate ' Reached the desired position.'.
```

4.4. Actions, states and assessments

The action planner obtains information from the indicative planner and constructs a plan by reactively responding to changes in the environment. The LHS of a response metarule represents the current behaviour, the state of the agent and the environment. The RHS represents an action undertaken by the agent.

The action planning process is divided into the following steps:

1. determine possible actions (without considering the environment) according to the current behaviour and situation;

2. collect information on the environment;

3. choose an action: move to a point determined by the situation and the environment, then change the agent's states: position, energy level, time level.

The robot assesses the situation at the currently visited point of the grid and selects a neighbouring point. In so doing it takes into account the current criteria and goals. The following metarule illustrates the robot's selection at the beginning of the mission:

```
robot: start ::= true &
robot: criterion1(gather_sample) ::= true &
robot: present_position(X, Y) ::= true &
robot: desired_position(Xd, Yd) ::= true &
robot: sample_station(List_sample) ::= true &
{ closest_point(t(X, Y), List_sample,
                t(Xs, Ys), t(Xd, Yd))
}
==>
robot:
 current_desired_position(Xs, Ys) ::= true &
robot: start_action_planning ::= true &
robot: action_index(1) ::= true.
```

The embedded call `closest_point( t(X, Y), Sample_list, t(Xs, Ys), t(Xd, Yd) )` gives a selection mechanism for choosing a sample station or an energy station. It is used to choose the point `(Xs, Ys)` from the

list `Sample_list` that is the closest to the current point `(X, Y)`. If more than one point is close to `(X, Y)` the closer to the desired point `(Xd, Yd)` is selected. For example, after the robot has arrived at the point (3,1), it determines the move either to point (5,1) or point (4,3) depending which is closer to the desired point (8,8).

An example of a response metarule that determines actions :

```
robot: action_index(1) ::= true &
robot: start_action_planning ::= true &
robot: present_position(X, Y) ::= true &
robot: current_desired_position(Xd1, Yd1) ::=
     true &
{ get_sorted_candidate_list(
      t(X, Y), t(Xd1, Yd1), List)
}
==>
robot: candidates(List) ::= true &
robot: action_index(1) ::= false &
robot: action_index(2) ::= true.
```

The embedded call `get_sorted_candidate_list( t(X, Y), t(Xd1, Yd1), List )` produces a list of points around point (X, Y), sorted according to the distance from a neighbouring point to the goal point. Different lists are determined by different situations. The point that is the closest to the goal point is put at the head of the list.

The robot checks the environment of the path to the first point in the candidate list to see whether it is possible to move to this point. If the move fails (there is an obstacle on the path), the robot will check the next point on the candidate list. The state of the environment is determined by metarules of the following form:

```
robot: start_action_planning ::= true &
robot: action_index(2) ::= true &
robot: present_position(X, Y) ::= true &
robot: candidates([t(Xa, Ya)|List1]) ::=
     true &
{ get_path_condition(t(X, Y), t(Xa, Ya),
                     Obstacle, Surface)
}
==>                                          ①
robot: proposed_position(Xa, Ya) ::= true &
robot: future_candidates(List1) ::= true &
robot:
   candidates([t(Xa, Ya)|List1]) ::= false &
environment: path_condition(
   t(X, Y), t(Xa, Ya), Obstacle, Surface) ::=
   true &
robot: action_index(2) ::= false &
robot: action_index(3) ::= true.
```

This metarule is used to determine randomly the presence of an obstacle and the surface condition on the path leading from point (X, Y) to point (Xa, Ya). The embedded call `get_path_condition( ... )` invokes a pseudo-random number generator that returns values indicating the state of the environment.

4.5. Adjustment mechanisms

The outcome of the assessment metarules is the evaluation of the current situation of the agent, the plan implementation and the environment. This evaluation may lead to a decision that the current behaviour remains appropriate or that it must be modified. If the behaviour need not be modified, the adjustment mechanism updates the current GR and the action planner continues. Response metarules for actions and assessment metarules are then applied. The adjustment mechanisms* change the parameter values of the predicates in GR and set controls for repeated searching and moving in action planning. For example, the following restructuring rule is used to continue action planning from response rule ① in order to consider the next point on the candidate list of intended moving direction.

```
robot: start_action_planning ::= true &
robot: action_index(5) ::= true &
robot: criterion1(C1) ::= true &
robot: criterion2(C2) ::= true &
robot: present_position(X, Y) ::= true &
robot:
  current_desired_position(Xd, Yd) ::= true &
robot: desired_position(Xdd, Ydd) ::= true &
robot: energy_left(E1) ::= true &
robot: time_left(T1) ::= true &
robot: energy_station(List_energy) ::= true &
robot: sample_station(List_sample) ::= true &
robot:
  history_record(Eh, Th, Listh) ::= true &
robot: candidates(List1) ::= true
==>
modify (
    states <-
       present_position(X, Y) &
       candidates(List1) &
       current_desired_position(Xd, Yd) &
       desired_position(Xdd, Ydd),
    static_environ_info <-
       energy_station(List_energy) &
       sample_station(List_sample),
    attributes <-
       energy_left(E1) & time_left(T1),
    behaviour <-
       criterion1(C1) &
       criterion2(C2),
    history <-
       history_record(Eh, Th, Listh) &
       start_action_planning &
       action_index(2) ).
```

5. An experiment

We ran several experiments with the knowledge bases whose small fragments are shown in Section 4. The experiment

* The adjustment mechanisms have not been implemented in the current version of Negoplan. We are working on an implementation because such mechanisms will be useful for different applications of Negoplan (for example, a patient simulator).

discussed in this section is illustrated in Fig. 3. The robot initially follows the plan given in Fig. 2 and moves to point (1,1) and then to (1,2). However, when it plans to move from (1,2) to (1,3), there is an obstacle on the path. Therefore it has to move through other paths—first move to (0,3) and then to (1,3) so that it can pick sample s1 located in (1,3). Although the path from (1,2) to (0,3) is rough, the robot still moves on it, because it operates under the criterion `short_path` .

After reaching point (1,3) the behavioural planner determines the plan for the second stage. It specifies the current goal for the robot: pick sample s3 at (3,4). This is the sample closest to (1,3) and it is also closer to the desired point (8,8) than the sample s2 at (1,5). Due to the environmental conditions the robot moves from (1,3) to (2,4) to (3,3) and finally to (3,4) where s3 is located (see Fig. 3).

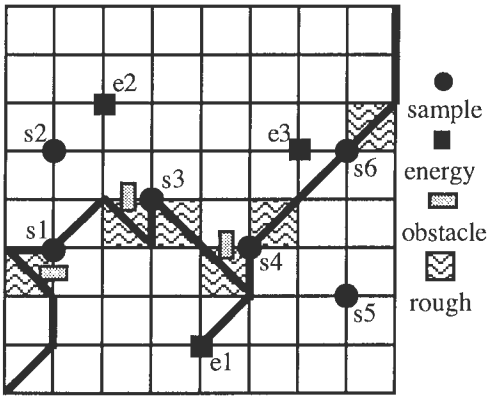


Figure 3. Results of an experiment

In the third stage of the planning process, the current goal is to reach point (5,3). However, when the robot arrives at (5,2), it has insufficient energy to complete the mission. The behavioural planner decides to change the current goal from picking a sample to moving to the energy station e1 at (4,1). Once the robot has replenished its energy, the goal of picking sample s4 becomes active. Next, the robot plans to pick sample s6 at (7,5). At point (7,5), it decides not to pick the remaining samples located at (1,5) and (7,2) because they are far from the destination (8,8). The robot changes its criterion and goes directly to (8,8).

6. Conclusions

Planning systems that solve complex tasks in unpredictable and changing environments have to provide solutions for several problems:

- to deal explicitly with the unpredictable situations,
- to modify the set of tasks and goals that are currently pursued,
- to treat planning time as a limited resource, i.e. quick responses,
- to reason about further planning,
- to be able to synthesize plans that implement any problem-solving behaviour necessary to solve complex tasks in its environment.

We used the Negoplan system to solve some of these problems. The system gives us a strong structure for replanning and real-time reacting and allows us to use the following mechanisms for reactive planning:

- selection mechanism for determining current goals,
- prediction mechanism for predicting the uncertain future by learning from the past experience,
- mechanisms for determining goal-oriented and problem-solving behaviour,
- decision making mechanisms to determine alternatives and make choice,
- mechanisms for the actions according to environment and behaviour,
- replanning mechanism to plan update and plan extension.

We proposed a two-level architecture for reactive planning and showed it could be expressed in Negoplan, a general-purpose decision analysis and simulation system. An action planner and an indicative planner are interlocked in our case study for robot motion planning.

The specialization of the two planners allows us to separate behaviour control from actions, to represent complex behaviour independently of the agent's states and plans of actions, and to formulate conditions in which the action planner operates. This simplifies the activities in the action planner and makes the planning process clearer. We show on a simple example the indicative planner generating behaviour for solving a problem and the action planner generating a sequence of actions. If a situation change requires plan modification, the indicative planner generates a new plan; otherwise the action planner continues to generate actions.

The same approach may apply to more complex situations in which the agent performs multiple tasks and needs to consider multiple facets of the environment as well as the actions of the other agents. The behavioural control mechanisms give the agent his goals and choice criteria. At any time, such goals and criteria may be in apparent conflict with a specific action that the agent must undertake, for example, when an obstacle is to be avoided. This conflict, however, does not impede the agent's ability to plan and act. The behavioural mechanisms, providing overall guidance, are of a strategic nature. The action planning mechanisms use them in plans and actions whose specifics need not be a direct transformation of behaviour. Moreover, the agent may be unable to follow the required behaviour. If there is a consistent discrepancy between the current behaviour and the actions generated by the action planner, the indicative planner may be forced to propose another behaviour.

References

[Agre and Chapman, 1987] P. E. Agre and D. Chapman. Pengi: An Implementation of a Theory of Activity. *Proc AAAI-87*, 268-272.

[Allen, 1991] J. F. Allen. Planning as Temporal Reasoning. *Proc Second International Conf on Principles of Knowledge Representation and Reasoning*, Cambridge, MA, April 1991, 3-14.

[Ambros-Ingerson and Steel, 1988] J. A. Ambros-Ingerson and S. Steel. Integrating Planning, Execution, and Monitoring. *Proc AAAI- 88*, 83-88.

[Blythe and Reilly, 1993] J. Blythe and W. S. Reilly. Integrating Reactive and Deliberative Planning for Agents. Technical Report CMU-CS-93-155, Department of Computer Science, Carnegie Mellon University, May 1993.

[Brooks, 1986] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Trans on Robotics and Automation*, 2(1): 14-23, 1986.

[Brooks, 1991] R. A. Brooks. Integrated Systems Based on Behaviors. *Proc AAAI-Spring Symposium on Integrated Intelligent Architectures*, Stanford University, March 1991. Available in SIGART Bulletin, 2(4), August 1991, 46-50.

[Dornand and Hommel, 1991] J. Dornand and G. Hommel. Reactive Planning—A Model of Knowledge-Based Real Time Planning. *Information Processing in Autonomous Mobile Robots, Proc International Workshop*, Munich, Germany, March 1991, Berlin: Springer-Verlag, 219-230.

[Firby, 1992] J. R. Firby. Building Symbolic Primitives with Continuous Control Routines, *Proc First International Conf on Artificial Intelligence Planning Systems*, 1992, 62-69.

[French, 1986] S. French. *Decision Theory. An Introduction to the Mathematics of Rationality*, New York: Wiley, 1986.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky. Reactive Reasoning and Planning. *Proc AAAI-87*, 677-682.

[Godefroid and Kabanza, 1991] P. Godefroid and F. Kabanza. An Efficient Reactive Planner for Synthesizing Reactive Plans. *Proc AAAI-91*, 640-645.

[Kaelbling, 1986] L. P. Kaelbling. An Architecture for Intelligent Reactive Systems. M. P. Georgeff and A. Lansky (eds.): *Reasoning about Actions and Plans*, Morgan Kaufmann, 1986, 395-410.

[Kambhampati, 1992] S. Kambhampati, A Validation Structure-Based Theory of Plan Modification and Reuse. *Artificial Intelligence*, 50(2-3): 193-258, 1992.

[Kersten and W. Michalowski 1989] G. E. Kersten and W. Michalowski. A Cooperative Expert System for Negotiation With a Hostage-Taker. *International Journal of Expert Systems*, 2: 357-376, 1989.

[Kersten *et al.* 1990] G. E. Kersten, L. Badcock, M. Iglewski and G.R. Mallory, "Structuring and Simulating Negotiations: An Approach and an Example", Theory and Decision, 28(3): 243-273, 1990.

[Kersten *et al.* 1991] G. E. Kersten, W. Michalowski, S. Szpakowicz and Z. Koperczak. Restructurable Representations of Negotiation. *Management Science*, 37(10): 1269-1290, 1991.

[Kersten *et al.* 1993] G. E. Kersten, S. Macdonald, S. Rubin and S. Szpakowicz. Knowledge-based Simulation for Medical Education. *Proc IASTED International Conference on Modelling and Simulation*, Pittsburgh 1993, 630-633.

[Kersten *et al.* 1994] G. E. Kersten, Z. Koperczak and S. Szpakowicz. Modeling Autonomous Agents in Changing Environments. Y. Ho and G. W. Zobrist (eds.): *Progress in Robotics and Intelligent Systems*, Norwood: Ablex (in print).

[Loyall and Bates, 1991] A. B. Loyall and J. Bates. Hap: A Reactive, Adaptive Architecture for Agents. Technical Report CMU-CS-147, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1991.

[Matwin *et al.* 1989] S. Matwin, S. Szpakowicz, Z. Koperczak, G. Kersten and W. Michalowski. Negoplan: An Expert System Shell for Negotiation Support. *IEEE Expert*, 4: 50-62, 1989.

[McDermott, 1991] D. McDermott. Planning Reactive Behavior: A Progress Report. *Proc Workshop on Innovative Approaches to Planning, Scheduling and Control*, 450-458.

[McDermott, 1992a] D. McDermott. Robot Planning. *AI Magazine* 13(2): 55-79, Summer 1992.

[McDermott, 1992b] D. McDermott. Transformation Planning of Reactive Behavior. *Research Report* No. 941, Department of Computer Science, Yale University, December 1992.

[Schoppers, 1987] M. J. Schoppers. Universal Plans for Reactive Robots in Unpredictable Environments. *Proc IJCAI-87*, 1039-1046.

[Shu, 1990] H. Shu. Planning with Reactivity. *Proc Artificial Intelligence in the Pacific Rim*, Nagoya, Japan, Nov. 1990, 681-686.

AIDA* – Asynchronous Parallel IDA*

Alexander Reinefeld and **Volker Schnecke**
PC² – Paderborn Center for Parallel Computing
D-33095 Paderborn, Germany
{ar|ossi}@uni-paderborn.de

Abstract

We present AIDA*, a generic adaptable scheme for highly parallel iterative-deepening search on large-scale asynchronous MIMD systems. AIDA* is based on a data partitioning scheme, where the different parts of the search space are processed asynchronously in parallel. Existing sequential solution algorithms can be linked to the AIDA* routines to build a fast, highly parallel search program.

Taking the 15-puzzle as an application domain, we achieved an average speedup of 807 on a 1024 processor system, corresponding to an efficiency of 79% on Korf's [1985] 25 largest problem instances. Specific problem instances yield more than 90% efficiency.

The total time taken by AIDA* to solve Korf's 100 random puzzles on a 1024-node system was 24.2 minutes. This is 5.7 times faster than the most efficient parallel algorithm on a 32 K CM-2 machine, SIDA* by Powley *et al.*

1 Introduction

Heuristic search is one of the most important techniques for problem solving in Artificial Intelligence and Operations Research. Since search algorithms usually exhibit exponential run-time, and sometimes also exponential space complexity, the design of efficient parallel searching methods is of obvious interest.

The backtracking approaches used in AI and OR benefit from a wealth of powerful heuristics that eliminate unnecessary states in the search space without affecting the final result. The most prominent methods include the universal *branch & bound* technique and *dynamic programming*, which examine only branches that are below/above a current upper/lower bound on the solution value. While these schemes are successfully applied in many problem domains, they do not work in domains with

- low solution density,
- high heuristic branching factor,
- poor initial upper/lower bounds on the optimal solution value.

Typical examples include single-agent games like the 15-puzzle [Korf, 1985], VLSI floorplan optimization [Wimer *et al.*, 1988], and some variants of the cutting stock problem [Morabito *et al.*, 1992]. For this kind of applications, there exists a simple and efficient backtracking method, called *Iterative-Deepening A* (IDA*)* [Korf, 1985], that performs a series of independent depth-first searches, each with the cost-bound increased by the minimal amount.

In this paper, we present AIDA*, a parallel implementation of iterative-deepening search on a massively parallel asynchronous MIMD system. AIDA* is based on a data partitioning scheme, where the different parts of the search space are processed asynchronously by the distributed processing elements. A simple, but effective task attraction scheme combined with a weak synchronization mechanism ensures high processor utilization and good scalability for up to more than a thousand processors.

Running on a 1024 processor transputer system, we achieved a speedup of 807 on twentyfive problem instances of the 15-puzzle, corresponding to an efficiency of 79%. Using Korf's [1985] random problem instances as a benchmark suite, AIDA* runs more than five times as fast as the fastest SIMD implementation, SIDA* by Powley *et al.* [1993], which was implemented on a CM-2 with 32 K processing elements. While such a comparison might seem unfair, because a single CM-2 processing element is about 100 times slower than the T805 transputers of our system, there are 32 times more processing elements in the CM-2. Hence one would expect our

transputer program to run three times faster. However, we achieved a time improvement by a factor of 5.7, due to faster work-load balancing and almost zero synchronization costs.

In the following, we first discuss the basic ideas of sequential IDA*, give a brief overview about previous parallel approaches, and present the AIDA* algorithm. Most of the paper is devoted to the discussion of our empirical performance results, including an analysis of the various overheads.

2 Iterative-Deepening Search

Iterative-Deepening A (IDA*)* [Korf, 1985] performs a series of independent depth-first searches, each with the cost-bound increased by the minimal amount. Following the lines of the well-known A* heuristic search algorithm [Nilsson, 1980, Pearl, 1985], the total cost $f(n)$ of a node n is made up of the cost already spent in reaching that node $g(n)$, plus a lower bound on the estimated cost of the path to a goal state $h(n)$. At the beginning, the cost bound is set to the heuristic estimate of the initial state, $h(root)$. Then, for each iteration, the bound is increased to the minimum value that exceeded the previous bound, as shown in the following pseudo code:

```
procedure IDA* (n);
bound := h(n);
while not solved do
    bound := DFS(n, bound);

function DFS (n, bound);
if f(n) > bound
    then return f(n);
if h(n) = 0
    then return solved;
return lowest value of DFS(n_i, bound)
        for all successors n_i of n
```

With an admissible (=non-overestimating) heuristic estimate function h, IDA* is guaranteed to find an optimal (shortest) solution path [Korf, 1985]. Moreover, IDA* obeys the same asymptotic branching factor as A* [Nilsson, 1980], if the number of newly expanded nodes grows exponentially with the search depth [Korf, 1985, Mahanti *et al.*, 1992]. This growth rate, the *heuristic branching factor*, depends on the average number of applicable operators per node and the discrimination power of the heuristic estimate h.

3 Applications

Typical application domains for IDA* search include VLSI floorplan optimization, some variants of the cutting stock problem and single-agent games like the 15-puzzle. These problems may be characterized by a high heuristic branching factor, a low solution density and poor information about bounds that can be used to prune the search tree.

We tested the performance of our parallel AIDA* algorithm on one hundred randomly generated problem instances [Korf, 1985] of the 15-puzzle. In its more general $n \times n$ extension, this puzzle is known to be *NP*-complete [Ratner and Warmuth, 1986]. While exact statistics on solving the smaller 8-puzzle are known [Reinefeld, 1993], the 15-puzzle spawns a search space of $16!/2 \approx 10^{13}$ states, which cannot be exhaustively examined. Using IDA*, an average of 10^8 node expansions are needed to obtain a first solution with the popular *Manhattan distance* (the sum of the minimum displacement of each tile from its goal position) as a heuristic estimate function.

4 Parallel Approaches

Previous approaches to parallel iterative-deepening search include parallel window searches, tree decomposition, search space mappings and special schemes for SIMD machines.

Powley and Korf [1991] presented a *Parallel Window Search*, where each processor examines the entire search space, but with a different cost-bound. Depending on the application, this method works only for a hand full of processors (e.g., 5–9 in [Powley and Korf, 1991, p. 475]) and the solution cannot be guaranteed to be optimal.

Kumar and Rao's [1987,1990] parallel IDA* variant is based on a task attraction scheme that shares subtrees (taken from a donator's search stack) among the processors on demand. For a selected problem set they achieved almost linear speedups on a variety of MIMD computers. These favorable results, however, apply only for MIMD systems with small communication diameters, like a 128 processor Intel Hypercube, a 30 processor Sequent Balance and a 120 processor BBN Butterfly. On a 128-node ring topology their algorithm achieved a maximum speedup of 63. From Kumar and Rao's analysis, it is evident that these results do not scale up to systems of, say, some thousand processors.

The algorithm of Evett *et al.* [1990] performs a mapping of the search space onto the processing elements of a SIMD machine. This allows to eliminate duplicate states at the cost of an increased communication overhead.

Two other approaches, SIDA* by Powley *et al.* [1993] and IDPS by Mahanti and Daniels [1993] also run on the CM-2. From these, SIDA* is probably the fastest parallel IDA* implementation, solving all 100 problem instances [Korf, 1985] of the 15-puzzle in 2.245 hours.

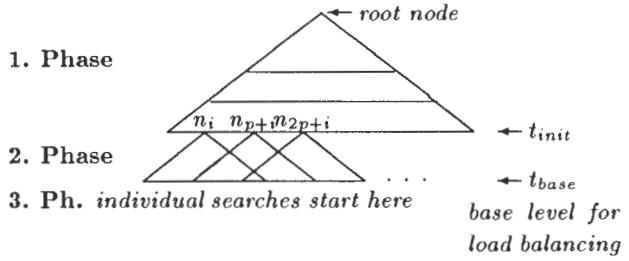


Figure 1: AIDA* Algorithm Architecture

5 AIDA*

In the following we describe *AIDA**, a generic adaptable scheme for highly parallel iterative-deepening search on asynchronous MIMD systems. AIDA* is based on a data partitioning scheme, where the different parts of the search space are processed asynchronously by the fastest available sequential routines running in parallel on the distributed processing elements. Existing sequential search code can be adapted to the parallel AIDA* system by linking the routines for initial tree partitioning, work-load balancing and communication. A simple, but efficient task attraction scheme combined with a 'weak' synchronization mechanism ensures a high processor utilization and good scalability up to some thousand processors.

AIDA* consists of three phases (cf., Fig. 1):

- a short *initial data partitioning phase*, where all processors redundantly expand the first few tree levels in an iterative-deepening manner until a sufficient amount of nodes is generated to keep each processor busy in the next phase,
- an additional *distributed node expansion phase*, where each processor expands its 'own' nodes of the first phase to generate a larger set of, say, some thousand fine grained work packets for the subsequent asynchronous search phase,
- an *asynchronous search phase*, where the processors generate and explore different subtrees in an iterative-deepening manner until one or all solutions are found.

None of these three phases requires a hard synchronization. Processors are allowed to proceed with the next phase as soon as they finished the previous one. Only in the third phase, some mechanism is needed to keep all processors working on about the same search iteration. However, this synchronization is a weak one (as opposed to hard barrier synchronization), allowing the processors to proceed with the next iteration after checking for work in their neighborhood only.

Similar to Newborn's [1988] *unsynchronized iteratively deepening parallel alpha-beta*, each processor carries out an iterative-deepening search on its selected subset of nodes. Our work-load balancing scheme ensures that all processors finish their iterations at about the same time.

5.1 Phase 1: Initial Data Partitioning

Before starting a distributed tree search, each processor must be supplied with a suitable amount of different nodes which can then be further expanded in parallel. This could be achieved in logarithmic time, $O(\log P)$, on P processors, using a binary divide-and-conquer approach. However, since communication on a MIMD-machine is usually an order of magnitude more time-consuming than the node expansion costs[1], AIDA* generates the first few tree levels redundantly on all processors. In the 15-puzzle, the processors perform an iterative-deepening search, saving all nodes of the last search frontier in a local node array, until there are at least $5 \cdot P$ entries. This gives a sufficient number of subtree roots (some 10,000 nodes) while not overflowing the memory resources of our transputer system.

At the end of this phase, duplicate nodes can be eliminated from the node array. In our experiments, however, we found that sorting the node array takes too much time. A total of 30% removed duplicate nodes (cf. [Powley *et al.*, 1993]) at the end of this phase gave only a 10% reduction of the total nodes, which did not pay for the increased overhead.

In practice, the first phase is short, taking less than three seconds (cf., Fig. 4) on the 1024-node system. There is neither communication or synchronization involved in this phase.

5.2 Phase 2: Generating Fine Grained Work Packets

In the second phase, processor P_i takes its nodes $n_i, n_{p+i}, n_{2p+i}, \ldots$ from the frontier node array t_{init} to get a wide-spread distribution of search frontier nodes. The nodes are expanded by applying two IDA* iterations, giving a new search frontier, t_{base}, as shown in Figure 1. At the end of the second phase, the local node arrays of the individual processors contain about 3000 frontier nodes each. These nodes make up the work packets used in dynamic load balancing in the third phase. As before, there is neither synchronization nor communication involved in this phase.

[1] This is especially true for the 15-puzzle with its cheap operator cost.

5.3 Phase 3: Asynchronous Search with Dynamic Load Balancing

The following iterations start on the frontier nodes t_{base}. All processors expand the nodes of their local array up to the current search-threshold. Since the size of the subtrees emanating from the t_{base} nodes is not known *a priori*, dynamic load balancing is required.

Our implementation of AIDA* employs a simple task attraction scheme. The $P = n^2$ processing elements are connected in a $n \times n$ torus topology (i.e. a mesh with wrap-around links in the rows and columns). Each processor is a member of two rings with n elements: the horizontal and the vertical ring.

A processor first expands its local frontier nodes of level t_{base}. When running out of work, it sends a `work_request` in clockwise order along the horizontal link of the torus. The first processor with unexpanded frontier nodes in its array sends a `work` packet back to the requester. If none of the processors on the horizontal ring has work to share, the request continues its path along the ring and eventually returns to the requester, indicating that the current iteration run out of work on this horizontal ring of the torus. The requester now sends a work request along the vertical ring using the same mechanism. If again no processor responds with a work packet, an `out_of_work` message is sent on both rings and this processor starts the next iteration. We call this a *weak synchronization* – as opposed to a hard barrier synchronization. It keeps all processors working at about the same iteration, while not requiring too much idle time [Newborn, 1988]. In practice, our weak synchronization works much like a majority consensus approach. When searching for a first solution, care must be taken that all processors working on shallower iterations finish their search before returning the optimal solution.

Note, that any work package is exclusively owned by a single processor. Whenever a package is transferred to another processor, it changes ownership. This is done with the expectation that all subtrees grow at about the same rate from one iteration to the next. Hence, the load balance will automatically improve during the search. In fact, the number of work packets decreases with increasing search time (cf., Fig. 7).

5.4 Implementation Details

While the above description gives a general outline of the AIDA* scheme, the actual implementation is more sophisticated:

- When a processor is done with its local nodes, it can start a new iteration when it receives an `out_of_work` message or detects a `work_request` with a higher cost threshold on the ring.
- To keep communication costs low, up to five nodes are bundled in a work packet. To avoid the donator from giving away all of its non processed nodes, only half of these are transferred.
- At the end of each iteration the nodes in t_{base} are re-ordered[2]: Medium size subtrees with $avg\_nodes/2 < x < 2 \times avg\_nodes$ are sorted to the end of the array, so that only work packets of average size will be transferred to other processors.
- In the hard problems (with many iterations), the size of work packets can differ by an order of magnitude. We therefore experimented with node splitting and node contraction strategies [Chakrabrti *et al.*, 1989] to adjust the work packets to an average size. Our preliminary results indicate that the additional overhead does not pay off.

6 Empirical Results

We implemented AIDA* on a 1024-node MIMD transputer system, using the 15-puzzle as a sample application. Figures 2 and 3 show the speedup results for two sets of 25 random problem instances with different difficulties. Speedup anomalies (cf. [Kumar and Rao, 1990]) were avoided by searching all nodes of the last (goal) iteration. We call this the *'all-solutions-case'* as opposed to the *'first-solution-case'*, where it suffices to find one optimal solution. Our fifty problem instances are the larger ones from Korf [1985], here sorted to the number of node generations in the 'all-solutions-case'. We also run AIDA* on Korf's fifty smaller problems. However, with an average parallel solution time of 8 seconds, a 1024-node MIMD system cannot be sufficiently utilized, so we did not include the data in this paper.

The speedup S of a parallel algorithm is measured as the ratio of the time taken by an equivalent and most efficient sequential implementation, $T(1)$, divided by the time taken by the parallel algorithm, $T(P)$:

$$S(P) = \frac{T(1)}{T(P)}.$$

Care was taken to use the most efficient sequential algorithm for comparison. Our IDA* is written in C and generates nodes at a rate of 35,000 nodes per second on a T805 transputer, corresponding to 350,000 nodes per second on a SUN Classic Workstation, or 660,000

[2] This is just a partial re-ordering, not a total sort. Nodes are sorted to the average size of all subtrees in the last iteration, *avg_nodes*.

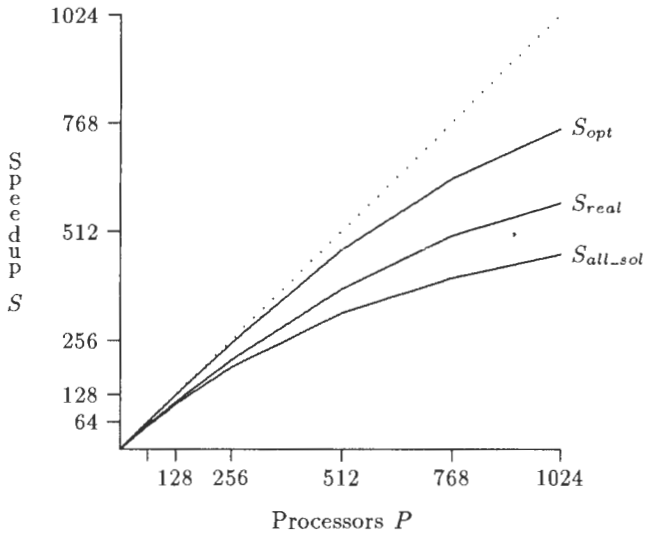


Figure 2: Speedup, prob. #51..75

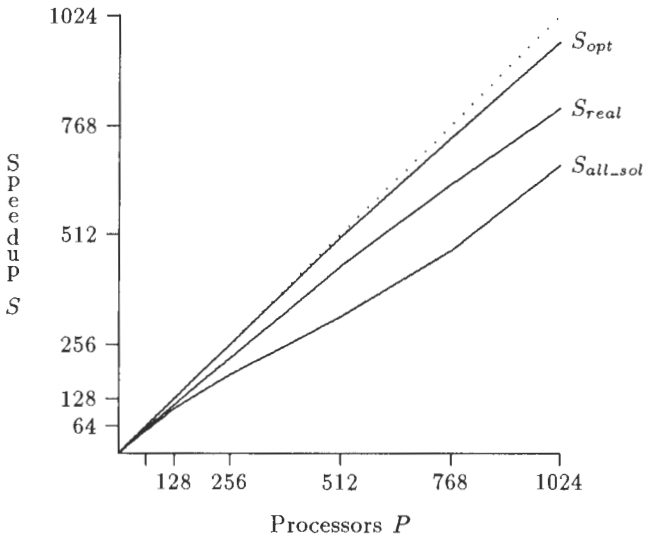


Figure 3: Speedup, prob. #76..100

nodes per second on a SUN SparcStation 10/40. Similar sequential IDA* run-times have been reported by Powley *et al.* [1993].

Figures 2 and 3 show the performance results on a torus topology. For each problem set, three graphs are shown:

S_{opt}: The topmost graph shows the maximum speedup that could be achieved with an optimal parallel algorithm (with zero overheads) after the first phase is done. This is a hypothetical measure to show how much time is taken by the initial data distribution phase.

S_{real}: The middle graph shows the speedup that would be obtained by a search for the *first* solution, one that stops right after one solution has been found. It includes the startup-time overhead, the communication overhead due to load balancing and the weak synchronization between iterations of the third phase.

S_{all_sol}: The bottom graph shows the actual speedup (measured in terms of elapsed time) of the 'all-solutions-case'. Compared to S_{real}, it also contains termination detection overhead and idle times due to processors which are done 'too soon' in the last iterations while others are still working on their last subtree.[3]

As is evident from Figures 2 and 3, good speedups are more difficult to achieve for the small problem instances #51..75 than for the hard ones #76..100. On the 1024-node system, the small problems take only an average of 16 seconds to solve, while the more difficult require three minutes. Hence, the negative effect of the initial work distribution, which is about constant for all problems, does not hamper the overall speedup in the hard problems too much.

7 Overheads in AIDA*

In this section, we analyze the various sources of overheads in more detail.

7.1 Initial Work Distribution

In the first phase, all processors perform a synchronous iterative-deepening search on the first few tree levels, storing all nodes of the last search frontier until there are at least $5 \cdot P$ nodes in each processor's local node array. This gives a sufficient number of work packets while not overflowing the memory resources.

For the larger systems, more nodes must be generated to give every processor a sufficient amount of 'own' nodes to work on. Hence, the CPU time spent in the first phase increases linearly with the system size, as shown in Figure 4. This additional node generation overhead does not reduce the overall efficiency of AIDA* in the large problems #76..100 too much. As shown in Figure 5, less than 1.5% of the total search time is spent in the first phase. Only the small problems #51..75 require up to 10% for the initial work-load distribution. This is just another manifestation of *Amdahl's Law*. The scalability of AIDA* can be improved by reducing the size of

[3] While in the 'first-solution-case' node expansion can be stopped after a first solution is found, all processors must finish searching their current subtree in the 'all-solutions-case'. Due to different work packet sizes, which vary most in the last iteration, some processors might get idle while others are still expanding their last tree. Most of the overheads in S_{all_sol} can be reduced by implementing a stack-splitting strategy as in [Kumar and Rao, 1990].

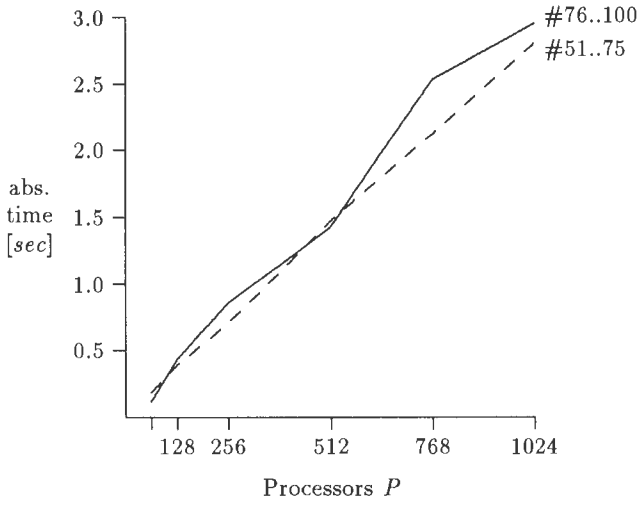


Figure 4: Absolute time of first phase

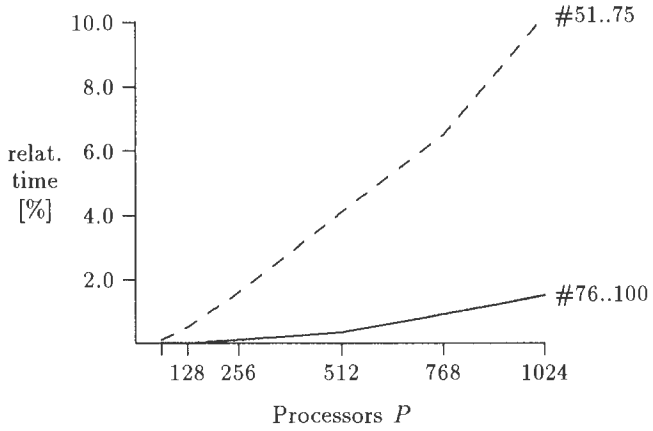


Figure 5: Time of first phase relative to total search time

the first phase or by expanding different subtrees on the parallel processors in a divide-and-conquer approach.

Note, that in the second phase, every processor starts node expansion on its own subtrees, thereby fully exploiting the parallel processing power.

7.2 Communication Overhead

Communication is another source of overhead hampering the performance of parallel algorithms, especially those running on massively parallel systems. Fortunately, AIDA* exhibits a very low communication overhead. Starting with 64 processors, one would expect the communication rate to increase by a factor of sixteen when increasing the system size to 1024 processors. However, as can be seen in Figure 6, the actual number of messages increases only by a factor of six. The curves seem to level off with growing system size, which can be explained by an increased likelihood that `work_requests` are answered in the immediate neighborhood of the idle

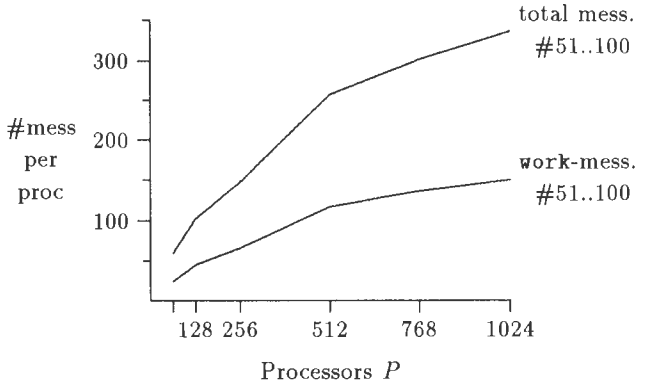


Figure 6: Messages per processor (last iteration only)

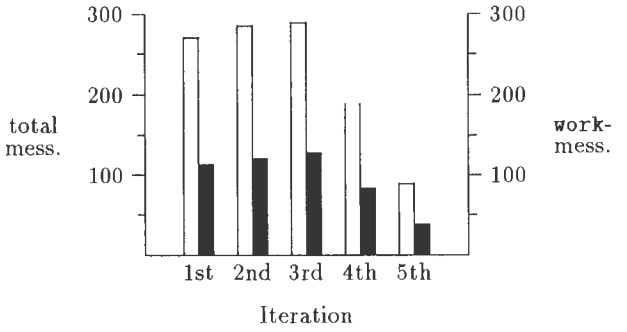


Figure 7: Messages per iteration (1024 procs.)

processor. As a consequence, the average distance between sender and receiver does not increase linearly with the system size.

At the end of an iteration, only a single `work_request` is sent around the ring to indicate that there is no further work available in the current iteration. All other processors on the ring are informed by an `out_of_work` message. They directly start the next iteration without asking for further work.

Moreover, as shown in Figure 7 the communication overhead seems to decrease with increasing search time. This is because the transferred nodes change ownership when being shipped to another processor, thereby constantly improving the global work-load balance. The number of messages that went through a single processor on a 1024-processor system decreases rapidly in the last two iterations. Due to this effect, one can assume an even lower communication overhead in other applications involving more iterations.

7.3 Termination Detection

With the weak synchronization scheme between the iterations, special provision must be taken to ensure that the returned solution is optimal in the 'first-solution-case'. When a goal node is found in iteration i, all processors

working on iterations $< i$ must complete their search to prove that no better solution exists.

In the 'all-solutions-case' (subject of this paper) the last iteration is searched to completion until all processors examined all their assigned subtrees. Due to the varying branching degree, the subtree sizes can hardly be estimated in advance. This results in different termination times for the parallel processors. As shown in Figures 2 and 3 (compare the two bottom graphs S_{real} and S_{all_sol}) the time spent in the finishing phase is appreciable. Note however, that this will usually not occur in the search for *one* solution, a case, that is more important in practice.

8 Conclusions and Future Research

In this paper, we presented a universal scheme for asynchronous parallel iterative-deepening search on massively parallel MIMD systems. Any sequential iterative-deepening algorithm can be linked to our generic AIDA* routines without much modification. Compared to a similar parallel MIMD scheme by Kumar and Rao [1990], our method is more general and obeys less communication overhead, especially on MIMD systems with a large diameter.

Moreover, AIDA* proved to be scalable for more than one thousand processors with a high efficiency.

For the 'first-solution-case', we solved Korf's [1985] hundred puzzle instances in 24 minutes, which is 5.7 times faster than SIDA* [Powley *et al.*, 1993] running on a 32 K CM-2. Comparing SIDA*'s theoretical speed of 32 K processors $\times$ 322 nodes/sec $= 10,551,296$ nodes/sec for the whole system with AIDA*'s theoretical speed of 1024 processors $\times$ 35,000 nodes/sec $= 35,840,000$ nodes/sec for our whole system, it is evident, that our MIMD approach makes better use of the processors. This is a remarkable result, considering that our faster machine solves the smallest 70 problems in less than 10 seconds each. Here, losses due to initialization are most pregnant. On the other hand, it is a more ambitious task to keep all processors of a SIMD machine working on relevant parts of the search space.

Our future work includes the solution of the 19-puzzle, where all heuristics that are known up to date, must be put together to obtain a solution within reasonable time limits. At the present time, we have solved 20 smaller[4] problems of 100 random 19-puzzle instances with an average runtime of 34 minutes on the 1024 processor system. VLSI floorplan optimization [Wimer *et al.*, 1988] is another practical application, which we intend to solve with AIDA*.

[4] avg. values: Manhattan distance h = 46, solution path length g = 67.5, 11.8 iterations, 56.2 billion nodes

Appendix: All-Solutions-Case Data

Prob.	Speedup				Time
	S(256)	S(512)	S(768)	S(1024)	T(1024)
51	163.8	278.9	363.3	419.4	11
52	195.2	351.8	492.0	303.9	18
53	209.7	393.7	389.2	453.4	11
54	196.6	360.3	502.1	620.2	9
55	211.8	402.9	430.2	419.5	10
56	196.9	355.1	503.8	597.2	10
57	184.8	341.0	474.8	541.0	11
58	211.0	330.4	446.9	531.9	11
59	213.8	338.3	460.5	563.8	12
60	210.6	403.3	577.2	500.1	13
61	190.5	354.2	323.3	349.1	24
62	221.2	393.8	562.2	707.8	11
63	215.3	359.4	492.4	604.4	13
64	213.3	411.4	457.1	560.8	14
65	208.8	401.4	573.4	728.9	13
66	211.1	317.2	432.1	518.8	18
67	216.2	413.8	473.6	579.1	16
68	219.9	399.0	570.5	730.6	17
69	216.5	415.8	595.3	769.2	14
70	216.2	416.9	599.4	534.9	24
71	211.2	352.2	494.9	616.5	20
72	212.5	427.1	625.3	653.5	21
73	221.4	404.5	576.6	736.5	21
74	222.4	406.5	589.6	754.0	24
75	222.0	382.8	541.3	682.8	23
76	220.3	428.8	624.4	811.9	21
77	218.0	423.5	625.0	801.0	32
78	218.0	413.7	599.7	775.1	34
79	222.0	444.2	536.3	666.0	37
80	221.3	397.1	559.1	717.3	34
81	220.1	431.3	637.1	677.9	42
82	208.3	396.8	580.3	733.0	40
83	220.1	430.5	636.6	662.2	55
84	221.1	434.6	565.7	719.4	46
85	221.4	433.5	562.4	709.7	54
86	219.2	429.7	633.2	828.7	62
87	220.7	432.7	647.2	846.0	66
88	223.1	439.8	602.4	777.2	88
89	220.5	435.7	646.9	852.4	79
90	225.5	435.2	642.4	845.9	80
91	225.7	396.7	576.5	734.8	125
92	226.2	448.7	671.2	890.6	165
93	224.1	445.5	664.6	882.6	144
94	223.6	446.0	665.9	869.9	149
95	225.5	450.7	658.5	870.7	175
96	224.8	448.1	652.7	858.9	207
97	226.3	449.8	672.3	897.4	214
98	235.8	467.6	698.1	924.9	520
99	230.4	452.2	675.0	896.4	579
100	230.4	459.8	687.3	915.3	1300

Acknowledgements

Thanks to Tony Marsland for many valuable comments and for visiting the PC^2 at the right time.

References

[Altmann *et al.*, 1988] E. Altmann, T. A. Marsland, T. Breitkreutz. *Accounting for Parallel Tree Search Overheads.* Procs. Int. Conf. Par. Proc. (1988), 198 – 201

[Chakrabrti *et al.*, 1989] P.P. Chakrabarti, S. Ghose, A. Acharya, S.C. de Sarkar. *Heuristic search in restricted memory.* Art. Intell. 41,2(1989/90), 197 – 221.

[Evett *et al.*, 1990] M. Evett, J. Hendler, A. Mahanti, D.S. Nau. *PRA*: A memory-limited heuristic search procedure for the Connection Machine.* 3rd IEEE Symp. Frontiers Mass. Par. Comp. (1990), 145 – 149.

[Korf, 1985] R.E. Korf. *Depth-first iterative-deepening: An optimal admissible tree search.* Art. Intell. 27(1985), 97 – 109.

[Kumar and Rao, 1990] V. Kumar, V.N. Rao. *Scalable parallel formulations of depth-first search.* In: Kumar, Gopalakrishnan, Kanal (eds.), Parallel Algorithms for Machine Intelligence and Vision, Springer-Verlag (1990), 1 – 41.

[Mahanti *et al.*, 1992] A. Mahanti, S. Ghosh, D.S. Nau, A.K. Pal and L. Kanal. *Performance of IDA* on trees and graphs.* 10th Nat. Conf. on Art. Int., AAAI-92, San Jose, CA, (1992), 539 – 544.

[Mahanti and Daniels, 1993] A. Mahanti, C.J. Daniels. *A SIMD approach to parallel heuristic search.* Art. Intell. 60(1993), 243 – 282.

[Morabito *et al.*, 1992] R.N. Morabito, M.N. Arenales, V.F. Arcaro. *An and-or-graph approach for two dimensional cutting problems.* European J. of OR 58(1992), 263 – 271.

[Newborn, 1988] M. Newborn. *Unsynchronized iteratively deepening parallel alpha-beta search.* IEEE Trans. Pattern Anal. Mach. Int., PAMI-10,9 (1988), 687 – 694.

[Nilsson, 1980] N.J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing, Palo Alto, CA, (1980).

[Pearl, 1985] J. Pearl. *Heuristics. Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, (1984).

[Powley and Korf, 1991] C. Powley, R.E. Korf. *Single-agent parallel window search.* IEEE Trans. Pattern Anal. Mach. Int., PAMI-13,5 (1991), 466 – 477.

[Powley *et al.*, 1993] C. Powley, C. Ferguson, R.E. Korf. *Depth-first heuristic search on a SIMD machine.* Art. Intell. 60(1993), 199 – 242.

[Rao *et al.*, 1991] V.N. Rao, V. Kumar, R.E. Korf. *Depth-first vs. best-first search.* 9th Nat. Conf. on Art. Int. AAAI-91, Anaheim, CA, (1991), 434 – 440.

[Rao and Kumar, 1987] V.N. Rao, V. Kumar. *Parallel depth-first search. Part I. Implementation.* Int. J. Par. Progr. 16,6(1987), 479 – 499.

[Ratner and Warmuth, 1986] D. Ratner, M. Warmuth. *Finding a shortest solution for the $N \times N$ extension of the 15-puzzle is intractable.* AAAI-86, 168 – 172.

[Reinefeld and Marsland, 1993] A. Reinefeld, T.A. Marsland. *Enhanced iterative-deepening search.* Univ. Paderborn, FB Mathematik-Informatik, Tech. Rep. 120 (March 1993), to appear IEEE-PAMI.

[Reinefeld, 1993] A. Reinefeld. *Complete solution of the Eight-Puzzle and the benefit of node-ordering in IDA*.* Procs. Int. Joint Conf. on AI, Chambéry, Savoi, France (Sept. 1993), 248 – 253.

[Russell, 1992] S. Russell. *Efficient memory-bounded search methods.* European AI-Conference, ECAI-92, Vienna, (1992), 1 – 5.

[Wimer *et al.*, 1988] S. Wimer, I. Koren, I. Cederbaum. *Optimal aspect ratios of building blocks in VLSI.* Procs. 25th ACM/IEEE Design Automation Conference, 1988, 66 – 72.