# CSCSI '88

## Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence

## Actes de la Septième Conférence Biennale de la Société Canadienne pour l'Étude de l'Intelligence par Ordinateur

edited by R. Goebel

Edmonton Convention Center
Edmonton, Alberta, Canada
June 6-10, 1988

# Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence

# Actes de la Septième Conférence Biennale de la Société Canadienne pour l'Étude de l'Intelligence par Ordinateur

edited by R. Goebel

**Edmonton Convention Center**
**Edmonton, Alberta, Canada**
**June 6-10, 1988**

in cooperation with

**University of Alberta**
**Canadian Man-Computer Communication Society**
**Canadian Image Processing and Pattern Recognition Society**

ISBN 0-88864-859-6



Edited by R. Goebel
Printed by University of Alberta Printing Services

Copies of these proceedings may be obtained as follows:

Within Canada, by prepaid order (Members: $35.00 CDN; Non-members: $40 CDN) plus $5 postage to:

*CIPS*
*243 College Street (5th floor)*
*Toronto, Ontario*
*CANADA M5T 2Y1*

Outside of Canada, enclose full payment ($35.00 US per proceedings, plus $2.00 US for the first copy and $.75 US for each additional copy for book rate shipment; for surface shipment out of the U.S. enclose full payment plus $3.00 US for the first copy and $2.00 US for each additional copy. California residents add 7% sales tax) and mail to:

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

# Message from the General Chairman

**Wayne A. Davis**
**Department of Computing Science, University of Alberta**

CSCSI '88, the seventh biennial conference on Artificial Intelligence in Canada, is establishing a new milestone and hopefully its most successful. This year a program has been put together comprising two full-day tutorials on such topical subjects as the development of knowledge-bases systems, mobile robotics and display of 3-D biomedical datasets. Fifteen paper sessions with more than fifty papers to be presented, follow the tutorial session. A trade show is also available for attendees to see state-of-the-art computer equipment.

Another milestone achieved this year is the concurrent conferences Vision Interface '88 and Graphics Interface '88. The decision to hold the three conferences together was made chiefly because of the overlap of subject matter, but it is also intended to stimulate meaningful interaction between the groups and to cement the cooperative bonds which have long existed between the societies. It is hoped that each person finds something of interest in the sessions and these proceedings which represent the definitive version of the presented papers.

I would like to thank each of the committee members, assistants, session chairmen, reviewers, authors, presenters, and visiting speakers for making this conference a success. As well, I would like to acknowledge the visible support for CSCSI '88 shown by the attendees.

# Organizing Committee, Conference '88

**Wayne A. Davis**
Conference '88 General Chairman
Dept. of Computing Science
University of Alberta
Edmonton, Alberta

**Darwyn R. Peachey**
Graphics Interface Program Chairman
Pixar
San Rafael, California, U.S.A.

**Tony Kasvand**
Vision Interface Program Co-Chairman
Computer Graphics Section, NRC
Ottawa, Ontario

**Adam Krzyzak**
Vision Interface Program Co-Chairman
Dept. of Computer Science
Concordia University
Montreal, Quebec

**Robert J. Woodham**
CSCSI Program Co-Chairman
Dept. of Computer Science
University of British Columbia
Vancouver, B.C.

**Nick Cercone**
CSCSI Program Co-Chairman
Centre for Systems Science
Simon Fraser University
Burnaby, B.C.

**Randy Goebel**
Registration Chairman
Dept. of Computing Science
University of Alberta
Edmonton, Alberta

**Terry Caelli**
Trade Show Chairman
Dept. of Psychology
University of Alberta
Edmonton, Alberta

**Mark Green**
Film Show Chairman
Dept. of Computing Science
University of Alberta
Edmonton, Alberta

**Jonathan Schaeffer**
Publicity Chairman
Dept. of Computing Science
University of Alberta
Edmonton, Alberta

**Ted Barnicoat**
Local Arrangements Chairman
Alberta Energy & Natural Resources
Edmonton, Alberta

**Rod Johnson**
Audio-Visual
Dept. of Computing Science
University of Alberta
Edmonton, Alberta

**Judith Abbott**
Registration Secretary
Dept. of Computing Science
University of Alberta
Edmonton, Alberta

**Lynn Gaetz**
Conference Treasurer
Dept. of Computing Science
University of Alberta
Edmonton, Alberta

# Message from the Program Co-Chairmen

**Nick Cercone**
**Centre for Systems Science, Simon Fraser University**

**Robert J. Woodham**
**Department of Computer Science, University of British Columbia**

In 1976, the first national conference sponsored by the CSCSI/SCEIO was held in Vancouver. Since then, conferences have been held every two years, the last one being in Montreal in 1986. These conferences have covered a wide spectrum of topics representative of contemporary research in artificial intelligence, cognitive psychology, and computational linguistics. In a significant departure, the CSCSI/SCEIO had also sponsored a special workshop in Halifax in 1985 on theoretical approaches to natural language understanding. The CSCSI/SCEIO is the oldest national organization for artificial intelligence research in the world, and it has been active.

This year's conference should be our best effort to date. The number and quality of submissions, with the success rate of acceptance of approximately 39%, is indicative. The organization of such an effort in a relatively short time requires the help of many people. The organizational efforts of Wayne Davis, Jonathan Schaeffer and Randy Goebel kept us on our toes and events on schedule. The program committee consists of Bill Bregar, Roger Browse, Terry Caelli, Veronica Dahl, Jim Delgrande, Renato De Mori, David Etherington, Randy Goebel, Bill Havens, Alan Jepson, Marlene Jones, James Little, Alan Mackworth, Gordon McCalla, John Mylopoulos, Peter Patel-Schneider, David Poole, Larry Rendell, Len Schubert and Steve Tanimoto. To all of these we offer our many thanks. To the invited speakers, to all of those who submitted papers and to all who participated in the reviewing process, we offer our gratitude. To our respective universities, Simon Fraser University and the University of British Columbia, we offer our sincere appreciation for the support they have extended to the conference. Finally, we wish to acknowledge the help of Carolyn Seely-Morrison (SFU) and Valerie McRae (UBC) for helping keep track of the day to day activities of the program committee.

Of the submitted papers, three were singled-out for award consideration. The paper entitled *The Complexity of Model-Preference Default Theories* by Selman and Kautz was selected for the best paper award. The papers *Time Revisited* by Miller and Schubert and *Curved Mondrians: a Generalized Approach to Shape from Shading* by Bischof and Ferraro received honorable mention.

# CSCSI Executive 1986-1988

**Dick Peacocke**, *President*
Bell-Northern Research

**Renato De Mori**, *Vice-President*
McGill University

**Bill Havens**, *Secretary*
Tektronix

**Randy Goebel**, *Treasurer*
University of Alberta

# CSCSI '88 Program Committee

## Program Co-Chairmen

**Nick Cercone**
Centre for Systems Science
Simon Fraser University

**Robert J. Woodham**
Computer Science
University of British Columbia

## Program Committee

**Bill Bregar,** Tektronix Research Laboratories, Tektronix
**Roger Browse,** Computing and Information Science, Queen's University
**Terry Caelli,** Psychology, University of Alberta
**Veronica Dahl,** Computing Science, Simon Fraser University
**Jim Delgrande,** Computing Science, Simon Fraser University
**Renato De Mori,** Computer Science, McGill University
**David Etherington,** AI Principles Research, AT&T Bell Laboratories
**Randy Goebel,** Computing Science, University of Alberta
**Bill Havens,** Tektronix Research Laboratories, Tektronix
**Alan Jepson,** Computer Science, University of Toronto
**Marlene Jones,** Advanced Technologies, Alberta Research Council
**James Little,** AI Laboratory, MIT
**Alan Mackworth,** Computer Science, University of British Columbia
**Gordon McCalla,** Computational Science, University of Saskatchewan
**John Mylopoulos,** Computer Science, University of Toronto
**Peter Patel-Schneider,** Schlumberger Palo Alto Research Lab
**David Poole,** Computer Science, University of Waterloo
**Larry Rendell,** Computer Science, University of Illinois at Urbana-Champaign
**Len Schubert,** Computing Science, University of Alberta
**Steve Tanimoto,** Computer Science, University of Washington

## Referees

Abramson, Harvey
Adams, Norman
Aleliunas, Romas
Angeles, J.
Armstrong, W.
Bacchus, Fahiem
Ballard, Bruce
van Beek, Peter
Bischof, Walter
Borgida, Alex
Borynec, Jim
Browse, Roger
Butler, Brian
Caelli, Terry
Campbell, M.
Cass, Todd
Cercone, N.
Chang, E.
Cheng, Mantis
Cohen, Robin
Dahl, Veronica
Dawes, Mike
Delgrande, Jim
Elcock, Ted
van Emden, Maarten
Etherington, David
Ferguson, Innes
Ferraro, Mario
Freiling, Mike
Gamble, Ken
Geffner, H.
Gillett, Walter
Ginsberg, Matt
Goebel, Randy
Goodwin, Scott
Greer, Jim
Greiner, Russ
Gurnsey, Rick
Hadley, Robert
Hamilton, Howard
Havens, Bill
Hayward, Vincent
Hirst, Graeme
Huang, Xueming
Hunter, Paul
Jacobsen, Chris
Jones, Marlene
Joseph, S.
Kautz, Henry
Lowe, David
Mackworth, Alan K.
Mahaderan, S.
Marley, A.
McAllester, D.
McCalla, Gord
McIlraith, Sheila
McLeish, Mary
Mendoza, M.
Mercer, R.E.
Merrett, T.
Mewhort, Doug
Neufeld, Eric
O'Hearn, Peter
Patel-Schneider, Peter
Pattabhiraman, T.
Pearl, Judea
Peters, Joe
Poole, David
Rehfuss, Steve
Reiter, Ray
Robinson, Edmund
Rudd, Walter
Saint-Dizier, Patrick
Saund, E.
Saxberg, Bror
Selman, Bart
Shoham, Y.
Snarr Carter, Vicky
Song, Fei
Spencer, Bruce
Stabler, Ed Jr.
Staley, Jeff
Szafron, Duane
Tubman, Jim
Turchan, Mark
Vegdahl, Steve
Vilain, Marc
You, Jia-Huai
Yukawa, Kei
Zlatin, Dan

# Invited Speakers

**Wolfgang Bibel**
Computer Science
University of British Columbia
***Finding Proofs, Programs, and Plans***

**Renato De Mori**
Computer Science
McGill University
***Neural Networks, Markov Models and Programming In Automatic Speech Recognition***

**David Etherington**
AI Principles Research
AT&T Bell Laboratories
***Non-Monotonic Reasoning: Is the Answer Harder than the Question?***

**Geoff Hinton**
Computer Science
University of Toronto
***Connectionist Symbol Processing***

**David Lowe**
Computer Science
University of British Columbia
***Recognizing Objects with Curved Surfaces and Moving Parts***

**Charles Morgan**
Philosophy
University of Victoria
***Bets, Logic and Monotonicity***

# Contents

**Applications**

# Finding Proofs, Programs, and Plans*

W. Bibel

University of British Columbia and
Canadian Institute for Advanced Research

**Abstract**

This paper is meant to set the stage for the oral presentation of material of a more technical nature that has been published elsewhere. In particular it articulates some of the troubles caused by current software production practice, while claiming that the route based on logical systems and their deductive capabilities would offer a more satisfactory approach. It also points out that such a route would require a concerted effort far beyond the usual investments of individuals. The paper is also meant to provide the reader with commented references to a number of technical results yielded in this context.

## 1 Introduction

Software production is a rapidly expanding multi–billion dollar business. The products coming out from this business, however, are far from satisfactory. Because of this people started talking of a "software crisis" around 1970. Since a crisis is something that is overcome after a while, the term is not used any more. The problems are worse today than they were in 1970, worse from the user's point of view that is. For business the situation is not bad at all, since unsatisfactory software raises the appetite for a new product and thus the selling figures.

There is little hope that this situation will change in the near future. The reason is that the production of fundamentally better software would require concerted efforts towards radical changes involving universities, industry, and governments. Matters must get really bad before they might have a chance to improve.

This pessimistic perspective should not prevent at least a few of us from taking little steps towards a different and better approach, far away from the mainstream activities in software engineering. I have devoted a substantial part of my work to such an approach, which since 1975 I have been calling *predicative programming*. It is based on the following principles.

Programming is to a large extent a reasoning process carried out on the basis of a body of knowledge of various kinds. It is an activity for which people typically are not at their very best. Hence machine support would help a lot. In order to provide suitable support, it is best to aim in principle, but not necessarily in practice, at a full automation of the whole process in order to get a better understanding of the mechanisms involved. This requires a full formalization of *all* parts involved. Only the formalisms of logic have the capacity to provide this formalization in a way that suits all these principles, in particular is capable to model the reasoning processes needed.

The problem with these principles is that they outline a research program of huge dimensions that will never be realized in an unconcerted way carried out by a few individuals. So this talk will not be able to present more than a few tiny pieces that eventually might fit into the pattern of the whole puzzle. These pieces are taken from my work pursued over more than a decade under the perspective of the principles just outlined. The paper itself is meant only to set the stage to the presentation of the more technical material and to provide the reader with the referential pointers to the publications elsewhere.

In detail, Section 2 reminds us of some of the problems with software while Section 3 summarizes the view of a concurrent way of predicative programming. The basic tools needed include deduction surveyed in Section 4. The next Section 5 then provides most of the pointers to predicative programming along with some comments and to the LOPS project realizing pieces of it. Section 6 points out that the same techniques are applicable to planning given the close relationship between planning and programming. Finally, Section 7 states in a somewhat more detailed way the rather pessimistic prospects for this kind of approach to programming.

## 2 The amenities of programming

The more advanced computer systems nowadays basically offer an option between a menu–driven and a command–oriented human interface. Let us consider the menu–driven option first. What the user can do there is choosing, at any given level, among a small set of options presented as icons or keywords in a menu. Any of these options would lead either to a different level with another

*Invited talk presented at CSCSI'88

menu or would activate a particular command.

An understanding of the meaning of the icons and keywords is expected from the user. This includes an understanding of the classification underlying the decision tree represented by the menus. It also includes an understanding of the commands offered for execution. Usually a help option supports this understanding if needed.

Menu–driven systems of this kind are considered to be very user–friendly, and indeed for limited and restricted applications such as text–processing they are quite useful. If we think of rather complex systems, however, their weaknesses become obvious. First of all, expecting the user's understanding, as just described, then means a substantial burden on his part. This burden can be alleviated by the help option to a very limited extent only, since the information to go into the help menu grows exponentially with the system's complexity. On the other side, running through a number of menus before being able to execute a certain command is a nuisance for the *experienced* user. Because of these reasons, the menu–technique is not preferred in more complex systems.

Command–oriented systems consist in the extreme of a single "menu" containing a listing of all commands available along with a description of their usage and meaning. This listing is provided to the user in the form of a "user's guide" or "user's manual". You have to be prepared to read a couple of hundreds of pages and keep them in your memory before you start doing useful work. We are here not talking of memorizing poems, mind you, but of attaching some meaning to a letter such as 'j' or a sequence like '^x^c', in fact to hundreds of them. Often the meaning depends on the system's actual status, thus complicating things even further.

This approach is fine for specialists spending much of their time with one or two systems. But we can forget it for normal people for which the computer is a *tool* that is taken into consideration in various circumstances and in irregular intervals to support their work, while often focussing on completely different subjects. What would be needed for such people?

Generally speaking, the system should behave in a way as to avoid distraction of the user's attention from his main work. That is, little effort should be needed to communicate a command to the system such as striking a key or at most a few of them. On the other hand, the user's awareness of the fixed associations of meanings to keys must not be anticipated by the system. Can we reconcile these seemingly contradictory requests?

Some progress towards such ends might still be possible with the current technology. In particular the help function could be made even more state sensitive, thus more focused. In addition, it could take into consideration statistical knowledge about the likelyhood of a certain command being sought for by the user at any given state. So calling help at any given moment would present, say, at most five options with brief descriptions among which in 95% of the cases the required command is included. The remaining 5% would require further paging down in the context–sensitively ordered list of commands. In particular, the phase of customization of a system needs being included into the domain of such a help function. In effect, such a system would more *actively* respond to the needs of the user.

Unfortunately, the solution just outlined is certainly not easy to achieve. It asks for the design of even more complex systems resulting in even more serious problems with respect to installation, adaptation, and maintenance (see below). Even if these could be managed, the result would still be a stupid system not noticing what's really going on.

We just touched upon a further crucial problem with the current state of affairs. Since present systems already are quite complex, never expect them to function properly or to suit even modest demands in all cases. Consequently, the next version will be released in due amount of time. You certainly want to keep up–to–date and therefore to replace the old version by the new one. Experience tells us that the tasks involved in this kind of work (fixing problems and incorporating new versions) keeps a technician busy full–time already for a relatively small installation. More importantly, the problems arising in daily use are becoming more and more complex. You better look for a technician that is really smart. Some of us may already have sensed a mood of despair in our labs.

So far our discussion was more or less concerned with the problems involved in doing the right thing at a given time in order to achieve an immediate behavior as desired. Programming to a great deal is concerned with foreseeing future needs of behavior without being able to anticipate the circumstances under which these needs arise. Typically at the time of programming we are only aware of a rough idea of the requirements of a task on a rather global level. It is therefore no surprise that the software eventually produced more often becomes a disappointment rather than a success. The wish for a change of the result is borne the day when the package is delivered.

Under this aspect, perhaps the most frustrating among all these problems is the fact that all software production is not additive at all. Relatively minor changes often require full rewriting of major parts of a system. Often,

such rewriting has to be carried out by a new team of programmers. Since familiarization with code is a demanding task, there is even more readiness to redo the whole thing from scratch. Paradoxically, in AI it has become even a virtue to throw away written software in the hope that the next version will be an improvement. The attitude reminds me of the past days where secretaries had to retype manuscripts again and again and again without ever achieving a correct version. And how much more difficult is programming in comparison with typing!

What is offered as a remedy for these problems in the literature of software engineering? You find countless pages filled on such topics as "application of development techniques", "life–cycle support", "verification, validation, and testing approaches", "high–level language programming", "data structures, abstract data types, hierarchical types, and operations", "integrated programming environments", and many others. I may be too short–sighted, but I simply cannot see the perspective of a solution to the fundamental problems discussed so far offered by any of these concepts. Indeed I also did not come across any article that would outline the scenario in which any of these would lead us in a more desirable future.

Since the software engineers should know, it seems that there is no much brighter future possible, or? There are a very few (compared to the masses of software engineers) researchers that have heralded a fundamentally different approach. Their background often is in Artificial Intelligence, or in Logic for that matter. Their keywords sometimes are "automatic programming" or "program synthesis" or "computational logic". Some of this work deserves a much closer inspection. Before we have a look at it let us envisage a scenario that would be more desirable, and do so even at the risk of wishful thinking.

# 3 Concurrent software production

Although the software engineers correctly speak of the life–*cycle* of software, thereby referring to various phases (requirements, specification, and so forth) they actually treat these phases in a *sequential* way, a fact that becomes evident even by this terminologically talking of *phases*. The appropriate approach would indeed be to think of the specialists for each of these phases sitting on a round table together, doing their work all in a concurrent way. Like with *concurrent manufacturing* we could speak of *concurrent software production*.

I have described the scenario of producing software in this way in quite some detail in [13]. The basic point simply is that it would be greatly desirable to have the various phases of the life–cycle intimitely interact with each other. For instance, it would be of great value to have an immediate feedback from the team implementing the software to those who set up the requirements, in order to be able to check whether the requirements are indeed met by the actual implementation. Similar arguments apply to other pairs of phases of the cycle, and they will not be repeated here. Anyway, no one would seriously disagree with such an idea. The questionable issue is whether this idea could ever be realized.

Realizing such a dream begs the solution to a number of very hard problems well–known in AI, as argued in some detail in [13], in fact in several preceding papers starting with [11]. The (semi–) automatic synthesis of an algorithm from a logical description is one of them, but by far not the only one. In fact, an even harder problem has to be solved in practice in this particular case since a complete and correct specification of that sort is hardly ever available. So the synthesis problem rather has to be solved even under the circumstances of incomplete, perhaps incorrect descriptions.

None of these problems is close to being solved. In fact, nearly no one seems actually be interested in their solution, at least none from software engineering, probably because there is little reward in attacking such very hard problems for which real progress is visible only on close inspection that no individual referee nor any appointment committee is able and willing to invest. In other words, the prospects for a realization look rather grim, an observation that will be discussed further in Section 7.

The software engineers seem to either have given up hope completely ever to be able to make software production happening in a desirable way, or they simply close their eyes in view of the tough problems they would be faced with, would they try to achieve such goals head–on. I am among those who have not given up hope completely (but see Section 7). Further I do at present see no other way towards such more desirable goals than solving some of these tough problems to at least some extent. For exactly that reason I am interested in achieving progress in the topics mentioned in the title of this paper which are discussed very briefly in the next three sections.

# 4 Proofs

There are various ways to formalize proofs. A number of them have been discussed in [6,1]. There we have also argued that first–order logic may be seen as a kind of core of any logical system worthwhile to be studied in de-

tail. Resolution [2] is the most popular proof method for first–order logic. The connection method [2,5] is an alternative. Each of these two methods has many particular realizations in terms of proof procedures.

Recently some progress has been achieved in evaluating the comparative power of both methods. [3] compares a number of specific procedures under this aspect. Later Haken [18] proved that resolution (on the ground level) necessarily is exponential. He established this using the pigeonhole formulas. In [7] it was now proved that, in contrast, the connection method allows short proofs for these same formulas. This shows that both methods are orthogonal to each other in a certain respect.

On the other hand resolution has a capacity that is not directly available in the connection method. Namely, a resolvent can be used many times in a proof like a lemma in mathematical proofs. Eder [15] has extended the connection method so that it incorporates the lemma feature like resolution. In fact, any resolution proof can now directly be simulated in this extended method so that it is at least as efficient as resolution. Indeed it is even provably more efficient because of the result concerning the pigeonhole formulas just mentioned.

In order to just give an idea of this extension, recall that the connection method is based on the characterisation of theorems by the existence of a spanning (and unifiable) set of connections. In Eder's generalization this set is organized by associating its connections with the nodes of a directed acyclic graph (*dag*) which defines the number of times each connection implicitly is taken into consideration for establishing the proof. This way a connection mentioned only once explicitly may play a multiple role like a lemma.

The purpose of these comparative studies is not so much a competitive one rather it apparently leads us to new insights into the nature of logical proofs and thus to more efficient proof mechanisms. While the comparisons focus on the simulation of proofs of the one method within the other, by the nature of these simulations they provide at the same time a comparative insight into the work necessary to *find* the proof.

The basic first–order proof techniques might provide only the core of future more complex proof systems as we already indicated above. So we emphasize once more that numerous further features [6,1] are expected to be integrated in order to make them powerful enough for applications. For instance, Wallen [19] has extended the connection method in order to deal with non–classical logics such as modal and intuitionistic logic.

Not only are there numerous possible features, but also many different ways to use these deductive mechanisms. For instance, though deductive in nature, they might well be used in order to achieve an *inductive* behaviour. Or, although they are designed with searching for correct and full proofs in mind, nothing prevents us to use the information provided by *partial* proofs of perhaps *incompletely* specified statements. Note in this context that the connection method provides us with a very natural notion of a partial proof as a set of connections that spans some of the paths but not necessarily all of them.

# 5 Programs

In [11,8] I have first explored the possibility of using proof techniques like those discussed in the previous section for the purpose of a more comfortable and reliable approach to programming. While this work appeared at roughly the same time when PROLOG entered the stage, it aimed from the very beginning towards a more general goal. In particular, it was not restricted to Horn clause logic. Also it did not compromise in terms of a purely predicative[1] style of programming.

The idea of predicative programming is to allow the user to communicate with the programming environment in a fashion that allows a problem description of the kind people use in non–computational circumstances. The thesis is that such a description, once it is sufficiently precise, can be transformed into some logical formalism. Supported by a knowledge base containing domain, strategic, and programming knowledge, such a description is transformed into a form more suitable for computation but still in logic. This part of the transformation involves strategically guided deductive processes. An interpreter (or compiler) would then enable this form of the problem to be executed like a PROLOG program, whereby the execution may be regarded as a proof. With the rising degree of automation, the idea of concurrent programming (cf. Section 3) would become feasible, since even partial descriptions could be handled in this way thus leading to immediate feedback during the phase of shaping the problem.

On the basis of [12] these ideas have been partially realized in the LOPS project [14] that is now further pursued by the AI group at the Technische Universität München within the ALPES project under the ambrella of ESPRIT. One could say so far that the project did not identify any fundamental problems indicating the infeasibility of the idea for whatever reason. On the other hand

[1] Note that this term of *predicative programming*, that recently has become fashionable, was coined first in [9,8].

it also demonstrated the enormous size of the task to be carried out, since we feel we have just gone a couple of steps on a long way towards the final goal. Some of the problems are the following ones.

Obviously there are all kinds of problems with the request for a more informal and flexible user interface that by and large are not directly related to the topic of this paper. Further there are numerous problems left for the transformational part that extracts a computationally feasible logic program out of the original description and the knowledge available to the system. It is this part where the LOPS project has contributed most, but where much more theoretical and practical work has to be invested. In particular, it is my strong believe that there is still a rich source of knowledge sitting in logical papers that could usefully be exploited for this purpose. Finally, there is still a great potential for improvement of compilers for the resulting logic program (see [1,10] for recent results). In this part there is an overlap with the work done in logic programming, and under certain aspects also with that in program transformation.

# 6 Plans

A program is some sort of a plan. So one would expect that planning is handled in a way similar to programming. It is not seen this way by most of those working on planning. In consequence, all the problems described for programming are inherent in planning systems in some way or another. In order to overcome these problems, a *predicative and concurrent planning* approach would have to be taken in much the same way as with programming.

As a first step towards such ends the relationship between proving and planning has been explored in [4], guided by the work known in program synthesis. The problem faced here concerns the fact that actions change the world as time progresses while logic seems to be of a nature not suitable for formalizing such changes. Our proposed solution to this problem considers the theorem prover as an actor that changes its state as time progresses, this way entering a new logical world after executing a rule that describes an action.

While the basic idea of this approach is pretty simple and attractive, again there are numerous problems waiting to be settled. Some of them are addressed in [17], partially in an inappropriate way from my point of view. One of the questions I would like to see an answer for in the first place, is the one about the relationship with modal logic. Apart from such specific questions concerning this particular setting of planning, all problems mentioned in the context of programming are of relevance here too, of course.

# 7 Prospects

There is a tremendous need for improvement of the current approach to programming (or planning) as we described in Section 2. In order to achieve such an improvement a more formal approach to programming would be needed. Thereby programming means all the activities involved in the entire life–cycle of software. Since formal approaches tend to be complicated, a uniform formalism would be of some help in this respect. Also, since introspection easily reveals that programming involves all sorts of reasoning, logic is the natural candidate for consideration as such a uniform formalism. We have briefly indicated in the last three sections, how logic could play a role in programming, mainly by referring to work published elsewhere.

So while there seems to be an attractive perspective, there is little reason for much optimism. First of all, logic is hard by all means. Even if you restrict your attention to one tiny subproblem involved, say for instance to term unification, then it already requires the full attention of a bright mind to come to grips with it. But there are tens if not hundreds of facets like unification involved in such a formal approach to programming. Therefore it seems hard to imagine one single mind that could ever bring together such a unified theory.

Hardly any of the facets we just addressed are ever taught at Universities. While any student of Computer Science is required to learn quite a bit of calculus which has only a minor relevance in our field they often can do without any knowledge in logic which is the field that brought forth the computer in the first place. So anyone starting a research career with the sort of problems discussed here in mind, is bound to reinvent the wheel. The frustration going along with such a discovery makes him change the subject to something less demanding in most cases.

In any case, striving for such a formal approach would require a lot of enthusiasm and concerted efforts. There is not the slightest indication for this sort of spirit in the field as a whole. On the contrary, key figures in software engineering seem not even to be aware of the nature of the problem. How else could it possibly be that one finds numerous unfriendly, certainly incorrect statements about the areas addressed in the present paper may be found in the literature such as in [16] to mention but one example.

So unfortunately it is more likely that this kind of

approach remains the hobby of a few outsiders that with their limited capacity will continue to contribute only tiny pieces of progress. Perhaps the only hope remaining might be the prospect that things become so bad as traditional systems become even more complex that people are really getting mad with them and start looking out for radical change. The principle of hope thus may be sustained also in this context of software engineering.

**Acknowledgements.** I want to express my thanks to D. Lowe for numerous helpful comments on this paper.

## References

[1] W. Bibel. Advanced topics in automated deduction. In R. Nossum, editor, *Fundamentals of Artificial Intelligence II*, Springer, Berlin, 1988.

[2] W. Bibel. *Automated Theorem Proving.* Vieweg Verlag, Braunschweig, second edition, 1987.

[3] W. Bibel. A comparative study of several proof procedures. *Artificial Intelligence*, 12:269–293, 1982.

[4] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[5] W. Bibel. Matings in matrices. *Comm. ACM*, 26:844–852, 1983.

[6] W. Bibel. Methods of automated reasoning. In W. Bibel and Ph. Jorrand, editors, *Fundamentals of Artificial Intelligence — An Advanced Course*, pages 173 – 222, Springer, LNCS *232*, Berlin, 1986.

[7] W. Bibel. On the comparative complexity of resolution and the connection method. *J.ACM*, submitted.

[8] W. Bibel. Prädikatives programmieren. In *GI — 2. Fachtagung über Automatentheorie und Formale Sprachen*, pages 274–283, Springer, Berlin, 1975.

[9] W. Bibel. *Predicative Programming.* Technical Report, Technische Universität München, 1975.

[10] W. Bibel. Predicative programming revisited. In W. Bibel and K. Jantke, editors, *MMSSSS'85 — Mathematical Methods for the Specification and Synthesis of Software Systems*, pages 24–40, Springer, Berlin, 1986.

[11] W. Bibel. *Programmieren in der Sprache der Prädikatenlogik.* (Rejected) thesis for "Habilitation" presented to the Faculty of Mathematics, Technische Universität München, January 1975.

[12] W. Bibel. Syntax–directed, semantics–supported program synthesis. *Artificial Intelligence*, 14:243–261, 1980.

[13] W. Bibel. Wissensbasierte softwareentwicklung. In W. Brauer and B. Radig, editors, *Wissensbasierte Systeme*, pages 17–41, Springer, Berlin, Fachberichte Informatik, 1985.

[14] W. Bibel and K. M. Hörnig. Lops — a system based on a strategical approach to program synthesis. In A. Biermann, G. Guiho, and Y. Kodratoff, editors, *Automatic program construction techniques*, pages 69–89, MacMillan, New York, 1984.

[15] Elmar Eder. Habilitationsarbeit. forthcoming.

[16] Jr. Frederick P. Brooks. No silver bullet — essence and accidents of software engineering. *IEEE Computer*, 20:10–19, April 1987.

[17] Bertram Fronhöfer. Linearity and plan generation. *New Generation Computing*, 5:213–225, 1987.

[18] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[19] L. Wallen. *Automated Deduction in Modal Logics.* PhD thesis, University of Edinburgh, 1987. PhD Thesis.

# Expressing Unrestricted Grammars by Extended DCG

Erik Knudsen

SYSLAB
Department of Information Processing and Computer Science University of Stockholm
S-106 91 STOCKHOLM SWEDEN

Abstract

A definition of extended definite clause grammar is presented and their relation to unrestricted grammars. A method for translating extended definite clause grammars describing unrestricted grammars into executable prolog programmes is presented. Three different parsing techniques are presented and for each a complete presentation of how to incorporate unrestricted grammars in the actual formalism is done.

Keywords: Logic Programming, Parsing, Unrestricted grammars, Definite Clause Grammars.

This work has been supported by the National Swedish Board for Technical Development (STU)

## 1 Introduction

Definite Clause Grammars (DCG) has gained a reputation of being a well suited formalism for defining arbitrary grammars that can easily be translated into executable prolog programmes [1,4,12]. Especially in natural language analysis this formalism is widespread and used in different systems which are able to analyze and understand sentences in natural language. By using the power of prologs unification, the backtracking mechanism and the ability of incorporating prolog programmes into a DCG, it is possible to build very powerful parsers.
Although it it possible to define very complex DCG they are in their pure form limited to describing only context free grammars. It is a well known fact that natural language can not be captured by merely a context free grammar. Different attempts not just within the logic programming society has been made in order to deal with this particular problem [2,3,5,11,13].
It is our belief that the grammar for a certain language is best defined purely from a linguistic point of view which means that the grammar should contain no artificial constructs or obscure restructuring of the grammar in order to enable the creation of a parser.
This does not mean that we do not accept the existence of embedded prolog goals or adding extra arguments to single constructs or parts of the grammar and thus make it possible to for instance build syntax trees.
Having this view in mind we strongly advocate that a defined grammar should as far as possible look the same even when expressed as a DCG. And this should also hold when dealing with unrestricted grammars.
In this paper we present one method of how any unrestricted grammar [6] can be expressed as an extended DCG (XDCG). Methods for translating the defined XDCG into executable prolog programmes are described in detail for three different kinds of parsing techniques namely top down parsing, left corner bottom up parsing and a parallel parsing system. We also give a brief introduction to the basic ideas behind those techniques [7,8,9,12].
In section 2 a brief definition of context free grammars is given as well as the method for translating the equivalent DCG into a prolog programme.
Section 3 defines unrestricted grammars and the equivalent form of the XDCG and how it can be translated into a prolog programme that is in fact a top down parsing system.
In section 4 different methods are given for realising two other kinds of parsing systems or rather how those existing systems can be extended in order to enable them to parse strings belonging to a language generated by an unrestricted grammar.
Finally we give some concluding remarks in section 5.

## 2 Context Free Grammars

A context free grammar $\Gamma$ can formally be defined as a quadruple

$\Gamma = (A, T, R, \Sigma)$ where A is the alphabet
$T \subseteq A$ is the set of terminals
$R \subseteq (A - T) \times A^*$ is the set of rules
$\Sigma \in A - T$ is the start symbol
$A - T$ is the set of non terminals

$L(\Gamma)$ i.e. the language generated by $\Gamma$ is defined as $\{\omega \in T^* : \Sigma \Rightarrow^* \omega\}$ where $\Rightarrow^*$ is a finite steps of derivations in $\Gamma$ from $\Sigma$ to $\omega$.
As an example let us choose:

$\Gamma = (A, T, R, S)$ where
$A = \{S, A, B, a, b\}$
$T = \{a, b\}$
$R = \{S \rightarrow \varepsilon,$
$S \rightarrow A\ S\ B,$
$A \rightarrow a,$
$B \rightarrow b\}$

then $L(\Gamma) = \{a^n b^n : n \geq 0\}$. The equivalent DCG could then be expressed as:

```
s --> ''.
s --> a & s & b.
a --> 'a'.
b --> 'b'.
```

where symbols within single quotes are regarded as terminals. Note that the empty string $\varepsilon$ is depicted as ''. The ampersand is the logical connective for conjunction.
This DCG is then translated into an executable prolog programme in a sequence of restructuring operations by the well known method that has been presented in previous works by e.g. [1,12].
The technique is roughly to add so called difference lists as arguments to each construct in the grammar. The parsing process is then carried out by passing a string sequentially from left to right over every construct in a chosen rule. A construct is understood as both being a terminal and a non terminal in the grammar. Another view is that every terminal or non terminal might be seen as a consumer of a substring of the original string and that a successful parse has been made when the complete string has been consumed.
The resulting prolog programme [10] is then:

```
s(S, S).
s(S0, S) <- a(S0, S1) & s(S1, S2) &
        b(S2, S).
a(a.S, S).
b(b.S, S).
```

The start symbol takes the string that is going to be parsed and if a successful parse is made, i.e. the string belongs to the language generated by the specified grammar, then the string should have been consumed completely. Normally strings are represented as a list of terminals and if a list is empty it consists only of the constant nil. So a parse of the string 'aabb' would be to prove the goal:

```
<- s(a.a.b.b.nil, nil).
true
```

If the string given belongs to the language the goal s/2 succeeds, if not it will fail.

```
<- s(a.b.a.b.nil, nil).
false
```

This technique can be extended to handle strings in every part of the right hand side of the rules. Furthermore extra variables or any prolog structure might be added as argument to each construct and thus e.g. build a syntax tree. As has also been said, arbitrary prolog goals can be inserted in any place of the right hand side of the rules in the DCG.
Since all this has been thoroughly described elsewhere [1,4,12] we will immediately tackle the problem of unrestricted grammars.

## 3 Unrestricted Grammars

An unrestricted grammar $\Psi$ can formally be defined as a quadruple:

$\Psi = (A, T, R, \Sigma)$ where A is the alphabet
$T \subseteq A$ is the set of terminals
$R \subseteq (A^*(A - T)A^*) \times A^*$ is the set of rules
$\Sigma \in A - T$ is the start symbol
$A - T$ is the set of non terminals

$L(\Gamma)$, i.e. the language generated by $\Gamma$, is defined as $\{\omega \in T^* : \Sigma \Rightarrow^* \omega\}$. We note that the only difference between context free grammars and unrestricted grammars is the definition of the rules.
As an example of an unrestricted grammar let us choose:

$\Psi = (A, T, R, S)$ where
$A = \{S, A, B, C, a, b, c\}$
$T = \{a, b, c\}$
$R = \{S \rightarrow \varepsilon,$
$S \rightarrow a\,S\,B\,C,$
$C\,B \rightarrow B\,C,$
$a\,B \rightarrow a\,b,$
$b\,B \rightarrow b\,b,$
$b\,C \rightarrow b\,c,$
$c\,C \rightarrow c\,c\}$

then $L(\Psi) = \{a^n b^n c^n : n \geq 0\}$ i.e. all strings with n numbers of a:s followed by n numbers of b:s and finally followed by n numbers of c:s.
Having in mind our earlier stated view of what should be expressed in the equivalent DCG describing the given grammar we get the following XDCG:

```
s --> ''.
s --> 'a' & s & b & c.
c & b --> b & c.
'a' & b --> 'a' & 'b'.
'b' & b --> 'b' & 'b'.
'b' & c --> 'b' & 'c'.
'c' & c --> 'c' & 'c'.
```

Note that this XDCG is almost identical to the given unrestricted grammar. No artificial constructs are added or other obscure restructuring is being made, all with the purpose of having a "clean" XDCG from a linguistic point of view.
Now the problem occurs of how this should be translated into an executable prolog programme since multiple conclusions are not allowed in a pure prolog programme. Hard as it might seem the solution is indeed simple and elegant. The idea is the following.
All the context free rules are translated according to the principle described earlier although terminals are treated somewhat differently. Terminals are treated according to the principle given in the following example:
We have three rules of the form:

```
a --> b & 'and' & c.
b --> 'b'.
c --> 'c'.
```

The generated programme becomes:

```
a(S0, S) <- b(S0, S1) & _and(S1, S2) &
        c(S2, S).
b(S0, S) <- _b(S0, S).
c(S0, S) <- _c(S0, S).
_and(a.n.d.S, S).
_b(b.S, S).
_c(c.S, S).
```

which means that for every terminal we create a terminal clause and the principle functor name is the constant generated from the terminal string. To ensure that there will be no conflict between grammar constructs and converted terminal symbols the underscore character is placed in front of every converted terminal. Note that in the prolog version we use this character is a constant.

For unrestricted grammar rules the following idea is applied:
The first construct of the left hand side of a rule is made the left hand side of the equivalent prolog clause. Add difference lists to every construct apart from the rest of the constructs of the left hand side of the rule. Create a list of those constructs and append it to the output argument of the head of the clause.
For every construct that now participates as a part of the created list, create an unrestricted terminal clause where the clause is a terminal with the actual construct as the token to be consumed when parsing.
Note that there is no difference between terminals and non terminals apart from the fact that terminals give rise to a set of terminal clauses. They resemble exactly the generated unrestricted terminal clauses.
So what we get is a set of clauses for every XDCG rule that is an unrestricted rule. The first member is an unrestricted non terminal clause (1) and the rest are unrestricted terminal clauses (2). If the same construct acts as an unrestricted terminal clause more then once, no more than one equivalent unrestricted terminal needs to be created. These generated unrestricted terminal clauses will only do a consumption when an unrestricted non terminal clause has been used when parsing. In other words, they convey the current context.
The generated prolog programme given the above XDCG is then:

```
        s(S, S).
        s(S0, S) <- a(S0, S1) & s(S1, S2) &
                b(S2, S3) & c(S3, S).
( 1 )   c(S0, b().S) <- b(S0, S1) & c(S1, S).
( 1 )   _a(S0, b().S) <- _a(S0, S1) & _b(S1, S).
( 1 )   _b(S0, b().S) <- _b(S0, S1) & _b(S1, S).
( 1 )   _b(S0, c().S) <- _b(S0, S1) & _c(S1, S).
( 1 )   _c(S0, c().S) <- _c(S0, S1) & _c(S1, S).
        _a(a.S, S).
        _b(b.S, S).
        _c(c.S, S).
( 2 )   b(b().S, S).
( 2 )   c(c().S, S).
```

An interpretation of an unrestricted non terminal clause is that a successful parse over the construct in the same clause has been made in the context of the appended constructs.
Unfortunately since the generated programme contains left recursive rules it is not possible to execute this with an ordinary prolog interpreter since it will surely loop. But it is easily seen that for example the goal s(a.a.b.b.c.c.nil, nil) is provable from a theoretical point of view.
In order to further prove our method consider the following unrestricted grammar expressed as a XDCG:

```
s --> ''.
s --> a & b & s & 'c'.
b & a --> a & b.
b & 'c' --> 'b' & 'c'.
b & 'b' --> 'b' & 'b'.
a & 'b' --> 'a' & 'b'.
a & 'a' --> 'a' & 'a'.
```

It should be clear that this XDCG expresses exactly the same language as the previous one, that is the set of strings such that $\{a^n b^n c^n : n \geq 0\}$. The equivalent prolog programme is then:

```
s(S, S).
s(S0, S) <- a(S0, S1) & b(S1, S2) &
        s(S2, S3) & _c(S3, S).
b(S0, a().S) <- a(S0, S1) & b(S1, S).
b(S0, b.S) <- _b(S0, S1) & _b(S1, S).
b(S0, c.S) <- _b(S0, S1) & _c(S1, S).
a(S0, b.S) <- _a(S0, S1) & _b(S1, S).
a(S0, a.S) <- _a(S0, S1) & _a(S1, S).
_a(a.S, S).
_c(c.S, S).
_b(b.S, S).
a(a().S, S).
```

Now we are able to parse sentences of the form $a^n b^n c^n$:

```
<- s(a.a.a.b.b.b.c.c.c.nil, nil).
true

<- s(a.b.c.a.b.c.nil, nil).
false
```

This functionality can easily be incorporated into existing systems built for translating ordinary DCG by adding a few clauses. Here although, we will only give the outlines for the process.
What is needed is to have a preprocessor that creates the two types of new XDCG rules from the given XDCG.
Assume a rule from the above given grammar e.g.
"b & a --> a & b". The preprocessor generates by using the backtracking facility the following two XDCG rules:

```
b & a --> a & b.
a  -->  a().nil.
```

The first one is the same as the given rule but the second one is a rule with the same construct acting both as non terminal and terminal. Note that it is converted into a term with arity zero and placed into a list as the only element.
Now these two XDCG rules are translated as usual in the main processor which adds difference lists. In that processor only the first construct of the left hand side of the rule gets distribution lists. After the main processor the rules now have become the prolog clauses:

```
b(S0, S) & a <- a(S0, S1) & b(S1, S).
a(a().S, S).
```

In the same preprocessor terminals may be converted into constants and the equivalent terminal clause be created. The rule

```
a --> 'a'.
```

will be replaced by the following two new rules:

```
a --> _a.
_a --> 'a'.
```

The final step is to have a postprocessor that appends all the untouched constructs in the head of the non terminal clause to the argument that outputs the remainder of a parsed string:

```
b(S0, a().S) <- a(S0, S1) & b(S1, S).
a(a().S, S).
```

If the existing translation processor is used in a context as below

```
repeat & read(X) & process(X, Y) &
X = eof
```

no major changes needs to be done:

```
repeat & read(X) & preprocess(X, Y) &
process(Y, Z) & postprocess(Z, U) &
X = eof
```

since we rely on the backtracking mechanism.
To conclude this part of the discussion in general any XDCG rule of the form:

$a_1(X_{11}, ..., X_{1m}) \& a_2(X_{21}, ..., X_{2m}) \& ... \&$
$a_n(X_{n1}, ..., X_{nm}) \dashrightarrow$
$b_1(Y_{11}, ..., Y_{1j}) \& ... \&$
$b_i(Y_{i1}, ..., Y_{ij}).$

is translated into:

$a_1(X_{11}, ..., X_{1m}, S_0, a_2(X_{21}, ..., X_{2m}). ...$
$.a_n(X_{n1}, ..., X_{nm}).S_i) \leftarrow$
$b_1(Y_{11}, ..., Y_{1j}, S_0, S_1) \& ... \&$
$b_i(Y_{i1}, ..., Y_{ij}, S_{i-1}, S_i).$
$a_2(X_{21}, ..., X_{2m}, a_2(X_{21}, ..., X_{2m}).S, S).$

.

$a_n(X_{n1}, ..., X_{nm}, a_n(X_{n1}, ..., X_{nm}).S, S).$

As has been said before, if terminals occur in a grammar rule they are first converted into atoms and then translated into a term with difference lists as arguments and terminal clauses are created for every terminal. This holds regardless of where the terminals occur. The technique for generating a prolog programme is still the same.
Formally, if any $b_k$ where $1 \le k \le i$ is a terminal it is converted into a constant $\_b_k$ and an additional rule

$\_b_k \dashrightarrow b_k.$

is generated if it does not already exist. The same holds for any $a_l$ where $1 < l \le n$.
Since terminals are treated in this way we allow synonyms to be included in the grammar.

$a_1 \dashrightarrow a_2.$

is after preprocessing converted into

$\_a_1 \dashrightarrow \_a_2.$
$\_a_2 \dashrightarrow a_2.$

For example this synonum rule

```
'a' --> 'd'.
```

generates the following two rules

```
_a --> _d.
_d --> 'd'.
```

and after appending difference lists we get

```
_a(S0, S) <- _d(S0, S).
_d(d.S, S).
```

Synonyms are rules which are not allowed in unrestricted grammars but since our formalism is able to handle those rules we do not want to prohibit this possibility.

## 4 Efficient parsing

Even if a top down parser is conceptually excellent it is by no means the most efficient way to parse strings in a language. On the contrary, it is in some situations too inefficient to be used in real life systems if the grammar is highly complicated or if it contains a large dictionary. Another drawback with pure top down parsing is that left recursive rules can not be dealt with, unless of cource grammar rules containing left recursion are rewritten. This is not a new insight since several works have been produced on how to build an efficient parser in prolog [7, 8,9]. We have come to like two different parsers which we will give a short presentation of and thereafter show how these parsing techniques can be extended in order to handle XDCG.

### 4.1 Bottom Up Parsing

The first one is called BUP and is a left corner bottom up parser which is described in detail in [7,8].
A BUP translator system restructures a given DCG into a new DCG that in its turn can be processed as usual with the earlier described translation principle for a top down parser.
For instance, given the same context free grammar as in section 2 the thus restructured DCG would look like:

```
        a(G) --> parse(s) & parse(b) & s(G).
        dictionary(s) --> ''.
        dictionary(a) --> 'a'.
        dictionary(b) --> 'b'.
( 1 )   s(s) --> ''.
( 1 )   a(b) --> ''
( 1 )   b(b) --> ''.
```

Note the extra so called terminal rules (1) that are added to the DCG. Those are created once for every non terminal in the grammar. Even if BUP is a bottom up parser it is in this version able to handle empty strings. A grammar containing empty strings or $\varepsilon$ productions will of course have an impact on the efficiency. A simple rewriting of the grammar so that it will only generate a language $L(\Gamma) = \{a^n b^n : n>0\}$ would give:

```
a(G) --> parse(b) & s(G).
a(G) --> parse(s) & parse(b) & s(G).
dictionary(a) --> 'a'.
dictionary(b) --> 'b'.
s(s) --> ''.
a(a) --> ''.
b(b) --> ''.
```

But we will use the grammar with the empty string allowed. The equivalent prolog programme would then look like:

```
a(G, S0, S) <- parse(s, S0, S1) &
        parse(b, S1, S2) & s(G, S2, S).
dictionary(s, S, S).
dictionary(a, a.S, S).
dictionary(b, b.S, S).
s(s, S, S).
a(a, S, S).
b(b, S, S).
```

The goal parse/3 is defined as:

```
parse(G, S0, S) <- dictionary(P, S0, S1) &
      X =.. (P.G.S1.S.nil) & X.
```

So to parse the string 'aabb' would be to prove the goal:

```
<- parse(s, a.a.b.b.nil, nil).
```

The principle for translating an XDCG into a prolog programme based upon the BUP idea is somewhat similar to the previous one described above. In general the following holds:
Given any unrestricted XDCG rule of the form:

$$a_1(X_{11}, ..., X_{1m}) \ \&\ a_2(X_{21}, ..., X_{2m}) \ \&\ ... \ \&$$
$$a_n(X_{n1}, ..., X_{nm}) \ \text{-->}$$
$$b_1(Y_{11}, ..., Y_{1j}) \ \&\ b_2(Y_{21}, ..., Y_{2j}) \ \&$$
$$... \ \&\ b_i(Y_{i1}, ..., Y_{ij}).$$

and $a_1$ is not a terminal it will after the preprocessing be translated into a new XDCG rule:

$$b_1(G, Y_{11}. ... .Y_{1j}.nil, A) \ \&$$
$$a_2(X_{21}, ..., X_{2m}) \ \&\ ... \ \&$$
$$a_n(X_{n1}, ..., X_{nm}) \ \text{-->}$$
$$parse(b_2, Y_{21}. ... .Y_{2j}.nil) \ \&\ ... \ \&$$
$$parse(b_i, Y_{i1}. ... .Y_{ij}.nil) \ \&$$
$$a_1(G, X_{11}. ... .X_{1m}.nil, A).$$

(1) $dictionary(a_2, X_{21}. ... .X_{2m}.nil) \ \text{-->}$
$$a_2(X_{21}, ..., X_{2m}).nil.$$

(1) $dictionary(a_n, X_{n1}. ... .X_{nm}.nil) \ \text{-->}$
$$a_n(X_{n1}, ..., X_{nm}).nil.$$

Here unrestricted dictionary entries (1) are created for every construct that participates in the left hand side of the rule except for the first construct. Terminals that occur in a rule that will not generate dictionary entries will be treated according to the same principle described as in section 3.
The goal parse/4 is defined as:

```
parse(G, A, S0, S) <-
      dictionary(P, A1, S0, S1) &
      X =.. (P.G.A1.A.S1.S.nil) &
      X.
```

If n=1 the following transformation is done:

$$b_1(G, Y_{11}. ... .Y_{1j}.nil, A) \ \text{-->}$$
$$parse(b_2, Y_{21}. ... .Y_{2j}.nil) \ \&\ ... \ \&$$
$$parse(b_i, Y_{i1}. ... .Y_{ij}.nil) \ \&$$
$$a_1(G, X_{11}. ... .X_{1m}.nil, A).$$

If n=1 and i=1 and $b_1$ is a terminal the following transformation is done:

$$dictionary(a_1, X_{11}. ... .X_{1m}.nil) \ \text{-->} \ b_1.$$

If i>1 and $b_1$ is a terminal the following two rules are generated:

$$terminal_k(G, *, A) \ \text{-->}$$
$$parse(b_2, Y_{21}. ... .Y_{2j}.nil) \ \&\ ... \ \&$$
$$parse(b_i, Y_{i1}. ... .Y_{ij}.nil) \ \&$$
$$a_1(G, X_{11}. ... .X_{1m}.nil, A).$$
$$dictionary(terminal_k, nil) \ \text{-->} \ b_1.$$

where $terminal_k$ is the k:th generated rule of this kind.
Finally we create terminal rules for every non terminal in the grammar according to the following

$$a_k(X_{1k}, ..., X_{kx}).$$

gives a terminal rule:

$$a_k(X_{1k}, ..., X_{kx}, Y, Y) \ \text{-->} \ ''.$$

When a synonym rule occur

$$a_1 \ \text{-->} \ a_2.$$

we get the following prolog clauses:

$$dictionary(G, A, a_2.S, S) \ \text{<-}$$
$$dictionary(G, A, a_1.S, S).$$
$$a_{1c}(S_0, S) \ \text{<-} \ a_{2c}(S_0, S).$$
$$a_{2c}(a_2.S, S).$$

where as usual $a_{ic}$ is the rewritten terminal symbol.
After the initial restructuring operations or the preprocessing phase all rules are treated as usual, eg appending distribution lists and the creation of prolog clauses. Finally we only have to consider those unrestricted rules or rather at this stage clauses with more than one conclusion. All constructs in the left hand side of the rule except the first one are appended to the last but one string argument in the final construct of the right hand side of the prolog clause.

$$b_1(G, Y_{11}. ... .Y_{1j}.nil, A, S_0, S_i) \ \text{<-}$$
$$parse(b_2, Y_{21}. ... .Y_{2j}.nil, S_0, S_1) \ \&\ ... \ \&$$
$$parse(b_i, Y_{i1}. ... .Y_{ij}.nil, S_{i-2}, S_{i-1}) \ \&$$
$$a_1(G, X_{11}. ... .X_{1m}.nil, A,$$
$$a_2(X_{21}, ..., X_{2m}). ... .$$
$$a_n(X_{n1}, ..., X_{nm}).S_{i-1}, S_i).$$

Translating the unrestricted grammar given in section 3 will result in the following prolog programme:

```
a(G, S0, S) <- parse(b, S0, S1) &
      parse(s, S1, S2) & _c(S2, S3) &
      s(G, S3, S).
a(G, S0, S) <- parse(b, S0, S1) &
      b(G, a().S1, S).
terminal1(G, S0, S) <- _c(S0, S1) &
      b(G, c.S1, S).
terminal2(G, S0, S) <- _b(S0, S1) &
      b(G, b.S1, S).
terminal3(G, S0, S) <- _b(S0, S1) &
      a(G, b.S1, S).
terminal4(G, S0, S) <- _a(S0, S1) &
      a(G, a.S1, S).
_a(a.S, S).
_b(b.S, S).
_c(c.S, S).
dictionary(s, S, S).
dictionary(a, a().S, S).
dictionary(b, b().S, S).
```

```
dictionary(terminal1, b.S, S).
dictionary(terminal2, b.S, S).
dictionary(terminal3, a.S, S).
dictionary(terminal4, a.S, S).
s(s, S, S).
b(b, S, S).
a(a, S, S).
```

The discrepancy between the general form and the examples we have given is due to the fact that we have not used any arguments to the non terminals in the defined XDCG. Integrating this mechanism into a DCG translating system will also require both a pre-translator and a post-translator. That is also a straightforward doing and will not be presented here.

In [7] different techniques for making the BUP parser more efficient are presented, e.g. introducing the link relation. All the techniques described are directly applicable in an XDCG without having to do any additional work.

### 4.2 Parallel parsing

The second parsing principle is the parallel parsing system (SAX) described in [9]. We will here only use the "primitive" version of the idea in order to describe how unrestricted grammars can be dealt with in a simple way. In the SAX version a given DCG is restructured by adding unique identifiers between both terminals and non terminals. Using our earlier presented context free grammar with the modification that we do not allow empty strings the equivalent DCG in the SAX environment would look like:

```
s --> a & id1 & b.
s --> a & id2 & s & id3 & b.
a --> 'a'.
b --> 'b'.
```

This DCG is now completely translated into a prolog prolog programme:

```
a(S, id1(S)).
b(id1(S0), S) <- s(S0, S).
a(S, id2(S)).
s(id2(S), id3(S)).
b(id3(S0), S) <- s(S0, S).
_a(S0, S) <- a(S0, S).
_b(S0, S) <- b(S0, S).
```

The start symbol in the grammar generates a prolog clause containing a test on the two markers that are introduced when the parsing takes place:

```
s(S0, S) <- S0 = begin & S = end.
```

When parsing the string 'aabb' the following goal is proved:

```
<- _a(begin, X1) & _a(X1, X2) &
      _b(X2, X3) & _b(X3, end).
```

Now to introduce the capability of parsing strings belonging to an unrestricted grammar we simply move all the left hand side constructs to the right and side. In general if given an unrestricted grammar rule of the form:

$$a_1(X_1) \,\&\, a_2(X_2) \,\&\, \ldots \,\&\, a_n(X_n) \text{ --> } b_1(Y_1) \,\&\, b_2(Y_2) \,\&\, \ldots \,\&\, b_m(Y_m).$$

where $1 \le k \le n$ and $1 \le l \le m$ and every $X_k$ and $Y_l$ equals $X_{k1}, \ldots, X_{kn}$ and $Y_{l1}, \ldots, Y_{lm}$ respectively, it is translated into:

$$b_1(Y_1, S0, id_1(S0, XY)).$$
$$b_2(Y_2, id_1(S0, XY), id_2(S0, XY)).$$
$$\vdots$$
$$b_{m-1}(Y_{m-1}, id_{m-2}(S0, XY), id_{m-1}(S0, XY)).$$
$$b_m(Y_m, id_{m-1}(S0, XY), Sn) \text{ <-}$$
$$\quad a_1(X_1, S0, S1) \,\&\, a_2(X_2, S1, S2) \,\&\, \ldots \,\&$$
$$\quad a_n(X_n, Sn\text{-}1, Sn).$$

where XY equals $X_1, \ldots, X_n, Y_1, \ldots, Y_m$.
If m=1 we get

$$b_1(Y_1, S0, Sn) \text{ <- } a_1(X_1, S0, S1) \,\&$$
$$\quad a_2(X_2, S1, S2) \,\&\, \ldots \,\&$$
$$\quad a_n(X_n, Sn\text{-}1, Sn).$$

Note that arguments to non terminals are passed via the id-structure. Further note that terminals are rewritten according to the same principle already described. By adding a list as argument to the end-marker we are able to get output from the parsing. If the start symbol in the grammar equals $s(X_1, \ldots, X_n)$ we get the following test predicate.

$$s(X_1, \ldots, X_n, S0, S) \text{ <- } S0 = begin \,\&$$
$$\quad S = end(X_1. \ldots .X_n.nil).$$

Given our previously introduced XDCG we change it to not contain empty productions to the following:

```
s --> a & b & 'c'.
s --> a & b & s & 'c'.
b & a --> a & b.
b & 'c' --> 'b' & 'c'.
b & 'b' --> 'b' & 'b'.
a & 'b' --> 'a' & 'b'.
a & 'a' --> 'a' & 'a'.
```

we get the following prolog programme:

```
_c(id2(S0), S) <- s(S0, S).
a(S, id1(S)).
b(id1(S), id2(S)).
_c(id5(S0), S) <- s(S0, S).
a(S, id3(S)).
b(id3(S), id4(S)).
s(id4(S), id5(S)).
b(id6(S0), S) <- b(S0, S1) & a(S1, S).
a(S, id6(S)).
_c(id7(S0), S) <- b(S0, S1) & _c(S1, S).
_b(S, id7(S)).
_b(id8(S0), S) <- b(S0, S1) & _b(S1, S).
_b(S, id8(S)).
_b(id9(S0), S) <- a(S0, S1) & _b(S1, S).
_a(S, id9(S)).
_a(id10(S0), S) <- a(S0, S1) &
      _a(S1, S).
_a(S, id10(S)).
```

The same idea can now be adapted into the extension of the SAX principle and expressed in a better suited form to be used in a parallel environment. This is however not described in this work.

## 5 Concluding Remarks

The notion of XDCG have been introduced and the principle for how this formalism can be translated into executable prolog programmes has been presented. We have also showed the outlines for how an existing system for translating DCG can be changed in order to deal with XDCG. A system for translating an arbitrary XDCG into either of the dicussed parsing principles has been implemented in prolog. Possible future research should be to investigate how both the BUP and SAX parsing techniques could be extended in order to handle the full class of unrestricted grammar rules which includes rules with heads beginning with a terminal.


## Acknowledgement

I would like to express my great gratitude to both G. Johnsson at IBM Svenska AB, Lidingo Laboratory and H. Lehmann at IBM Germany, Heidelberg Scientific Center who initially showed that it was possible to express a subset of unrestricted rules within the BUP formalism. Also to IBM Svenska AB, Lidingo Laboratory where parts of the ideas presented in this work has been implemented. Many thanks also to A. Bjornerstedt, I. P. Orci and B. Wangler for reading and commenting on this work.


## References


[1] Clocksin, Mellish, "Programming in Prolog", *Springer Verlag, 1981*

[2] A. Colmerauer, "Metamorphosis Grammars", *Natural Language Communication with Computers Ed. Bolc, Springer Berlin, May 1978*

[3] V. Dahl, H. Abramson, "On Gapping Grammars", *Proceedings of the Second International Logic Programming Conference, 1984*

[4] V. Dahl, "Hiding Complexity From the Casual Writer of Parsers", *Natural Language and Logic Programming, North Holland, 1985*

[5] H. Lehmann, N. Ott, M. Zeoppritz, "A Multilingual Interface to Databases", *IBM Germany, Heidelberg Scientific Center*

[6] H. Lewis, C. Papadimitriou, "Elements of the Theory of Computation", *Prentice Hall, 1981*

[7] Y Matsumoto, H. Tanaka, H.Hirikawa, H.Miyoshi, H.Yasukawa, "BUP: A Bottom-Up Parser Embedded in Prolog", *New Generation Computing, No 1, 1983, pp 145-158*

[8] Y Matsumoto, M. Kiyono, H. Tanaka, "Facilities of the BUP Parsing System", *Natural Language and Logic Programming, North Holland, 1985*

[9] Y. Matsumoto, "A Parallel Parsing System for Natural Language Analysis", *ICOT Research Center, Institute for New Generation Computer Technology*

[10] IBM, "VM/Programming in Logic", *Program Description and Operations Manual, Program Number 5785-ABH, 1985*

[11] F. Pereira, "Extraposition Grammars", *American Journal fo Computational Linguistics, Vol 7 No 4, 1981*

[12] F. Pereira, D. Warren, "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks", *Artificial Intelligence 13, 1980, pp 231-278*

[13] P. Sabatier, "Puzzle Grammars", *Natural Language and Logic Programming, North Holland, 1985*

# Using Default Logic to Derive Natural Language Presuppositions

Robert E. Mercer*
Department of Computer Science
Middlesex College
University of Western Ontario

## Abstract

Natural language presuppositions are an essential part of the meaning of a natural language utterance. Considered as inferences, presuppositions are derived from the uttered sentence, the background real world knowledge, and knowledge concerning conversational principles. Due to the conjectural and defeasible nature of these inferences, the derivation process cannot be a classical inference procedure. A method is discussed that uses default rules to capture the linguistic knowledge, one or more default theories to represent the utterance, and a default logic proof procedure to perform the inferencing.

## 1 Introduction

A hearer's interpretation of a natural language utterance should include the inferences that can be generated from three sources: the sentence uttered, knowledge about the world, and knowledge about language use. One well-studied inference is the *natural language presupposition*[1].

Being implied by a natural language sentence and the natural (or preferred) interpretation of its simple negation is the primary quality that qualifies an inference as a presupposition. This evaluation of inferences is called the *negation test*. Presuppositions are generated from lexical and syntactic contexts. Those contexts which pass the negation test can be termed *presuppositional environments*. Sentences (1)–(5) demonstrate some prototypical examples of presuppositions produced by the following presuppositional environments, respectively: noun phrases, possessives, factive verbs, certain aspectuals, and definitions of words. In each of these examples the truth of the affirmative a-sentence always implies the truth of the c-sentence, and the truth of the negative b-sentence normally implies the truth of the c-sentence.

(1) a. The present king of Buganda is bald.
b. The present king of Buganda is not bald.
c. There exists a present king of Buganda.

(2) a. Jack's children are bald.
b. Jack's children are not bald.
c. Jack has children.

(3) a. Mary is surprised that Fred left.
b. Mary is not surprised that Fred left.
c. Fred left.

(4) a. (At time $t$), John stopped beating the rug.
b. (At time $t$), John did not stop beating the rug.
c. (Prior to time $t$), John had been beating the rug.

(5) a. My cousin is a bachelor.
b. My cousin is not a bachelor.
c. My cousin is a male adult.

Presuppositions have the quality that there are linguistically natural means for indicating that a simple negation is not to be interpreted normally. Examples for each of the sentences (1)–(5) are given in (6). It is important to note that the method used to indicate that the negation is to be interpreted in an unnatural manner is that one of the normal (presuppositional) inferences is made inconsistent.

(6) a. The present king of Buganda is not bald; Buganda is a republic.
b. Jack's children are not bald; he doesn't have any.
c. Mary is not surprised that Fred left because he didn't leave.
d. John did not stop beating the rug because he hadn't started.

*This research was partially supported by NSERC grants A7642 (to R. Reiter) and A3039 (to P. C. Gilmore).

[1] I use this term here rather than the simpler *presupposition* to indicate that its linguistic usage is different than its orthodox philosophical use. Presuppositions were originally proposed to imply the existence of noun phrase referents and to explain the lack of a truth value for those sentences whose presuppositions were false. In its linguistic sense, the term embodies the class of inferences, generated from a number of linguistic situations, which pass a negation test. Throughout the remainder of this paper I will drop the modifiers and use the simpler *presupposition* to mean *natural language presupposition*.

e. My cousin is not a bachelor. He is only three years old.

In addition, knowledge about the world can override the normal interpretation of negative sentences. The following example provides an instance of a normal (presuppositional) inference being prevented by information contained in the non-linguistic context. Suppose that both Bill and Jim know that Bill's cousin is a three-year old. They want to go to a bachelor party tonight but Bill must babysit his cousin. In response to Jim's question, "What are we going to do with your cousin?", Bill utters (7), in a sense meaning that the cousin wouldn't be able to go to the party. In this case Jim would not make the normal (presuppositional) inference that Bill's cousin was an adult because the non-linguistic context, known to both Bill and Jim, contradicts this inference.

(7) My cousin is not a bachelor.

A persistent theme in the attempts by linguists to define presuppositions, is the *projection problem*: given the presuppositions of a simple sentence, which ones survive the embedding of this sentence in a more complex sentence. Verbs like 'dream' prevent the presuppositions of their subordinate clauses being projected as presuppositions of the main sentence. Sentence (8) does not presuppose the existence of a King of Buganda even though the subordinate clause in isolation does. 'Possibly' normally projects all underlying presuppositions. (9) normally implies that John has children. Other constructions, most notably 'or' and 'if ... then', sometimes project the presuppositions from both clauses, for example (10), and sometimes do not, for example the presupposition of the consequent is not projected in (11).

(8) Jack dreamt that the King of Buganda is bald.

(9) It is possible that John's children are at school.

(10) Mary stopped beating the rug or John stopped beating the egg.

(11) If John was beating the egg then he has stopped (beating the egg).

The purported solutions devised by a number of linguists have all been structural in nature: Given a tree-like structural description of the sentence and the presuppositions of the leaf nodes, which presuppositions get recursively inherited by the parent nodes. Some of the solutions require additional (non-structural) filtering. A different approach to the projection problem is to view presuppositions as inferences (in an appropriate logic) that can be derived from the sentence and the *linguistic* and *non-linguistic context*. This view places the study of presuppositions in the much broader context of knowledge representation and reasoning. The remainder of this paper discusses the technical aspects of a default logic approach, focussing mainly on the complications created by 'or' and 'if ... then'.

## 2 Representing Natural Language Negation

Classical representation problems are caused by negation. The problems occur because the standard method of negation in the representation language (I am assuming first order logic) does not correspond to the preferred interpretation of negation in natural language. These problems are exemplified in (12). The affirmative sentence (12a) is represented in (12b). The sentence (12c) is the negation of the affirmative sentence. Although the negation of (12b) is given in (12d), the 'usual meaning' of (12c) is more closely represented by (12e). On the other hand, (12f) cannot be represented by (12e).

(12) a. The present king of Buganda is bald.

b. $\exists x.\mathit{King\text{-}of\text{-}Buganda}(x) \wedge \forall y(\mathit{King\text{-}of\text{-}Buganda}(y) \supset x = y) \wedge \mathit{BALD}(x)$

c. The present king of Buganda is not bald.

d. $\neg\exists x.\mathit{King\text{-}of\text{-}Buganda}(x) \wedge \forall y(\mathit{King\text{-}of\text{-}Buganda}(y) \supset x = y) \wedge \mathit{BALD}(x)$

e. $\exists x.\mathit{King\text{-}of\text{-}Buganda}(x) \wedge \forall y(\mathit{King\text{-}of\text{-}Buganda}(y) \supset x = y) \wedge \neg\mathit{BALD}(x)$

f. The present king of Buganda is not bald because there is no king of Buganda.

There are two approaches to solving the representational problems caused by negation in natural language. The orthodox view is to say that negation is (syntactically or lexically) ambiguous between two or more representations. What seems to be an insurmountable problem for this view is to provide the means to decide which representation to use in different situations. The heterodoxy is to say that negation is *vague* that is, there exists only one representation which is true under more than one set of truth conditions. Proponents of the heterodox view include Kempson (1975), Wilson (1975), Atlas (1977), Gazdar (1979), and Mercer (1987). In this view sentence (12c) has the single representation given by (12d). The problem for this view is that while the representation allows for multiple interpretations, Grice's Principle of Cooperative Conversation (Grice (1975)) commits the speaker to using an utterance that should allow the hearer to generate the correct interpretation, that is, the one corresponding to the interpretation for (12e). Choosing the more general (12d) to represent (12c) immediately prohibits the use of first order logic to derive the preferred interpretation for the following reason. (12e) is not a logical consequence of (12d) in first order logic and if (12d) were supplemented with a set of axioms that allowed the derivation of (12e), an undesirable consequence would be that the presuppositions of (12c)

would be valid or they could be derived from the set of axioms (that is, they would be unconnected to the utterance). The other requirement, that the derived interpretation is only *preferred* and that one of the other possible interpretations is to be chosen if there is sufficiently clear indication to reject the preferred interpretation, also prohibits the use of first order logical techniques to derive the preferred interpretation. Since first order logic is monotonic the preferred interpretation would always be derivable. However, the other interpretations are inconsistent with the preferred interpretation.

Since the inference must be conjectural and the rules of inference must be defeasible, default logic has been used to capture the required inferencing abilities. This paper discusses this technique for negated presuppositional environments.

# 3 Logical Representation of Presuppositions Using Default Rules

I assume throughout this paper that the speaker's utterance has undergone the first phase of the interpretation process which generates a semantic representation (logical form) of the sentence uttered. This semantic representation will be a well-formed sentence in a first order S4 modal language containing a countably infinite set of predicate symbols, constant symbols, and variable symbols, plus the logical symbols $\wedge$, $\vee$, $\supset$, $\neg$, $\mathsf{K}_S$, and $\mathsf{P}_S$. The last two symbols, called modal operators, are to be interpreted as 'the speaker knows that' and 'for all the speaker knows, it is possible that', respectively. Although there is no general method known to generate this representation, some general rules can be followed. Any sentence with an explicit negation is translated into the widely scoped negation of its affirmative counterpart. Any compound sentence is mapped clause by clause into a logical form, each clause being treated as a sentence.

A *normal default rule* is a rule of inference denoted

$$\frac{\alpha(\vec{x}) : \beta(\vec{x})}{\beta(\vec{x})}$$

where $\alpha(\vec{x})$ and $\beta(\vec{x})$ are all first order formulae whose free variables are among those of $\vec{x} = x_1, \ldots, x_m$. Intuitively, a default rule can be interpreted as: For all individuals $x_1, \ldots, x_m$, if the prerequisite $\alpha(\vec{x})$ is believed[2] and if $\beta(\vec{x})$ is consistent with what is believed, then the consequent $\beta(\vec{x})$ may be conjectured. A *normal default theory* is a set of first order formulae together with a set of normal defaults. A *fixed point* of a normal default theory is the deductive closure of the set comprised of the first order formulae and some maximal set of consequents that are consistent with the fixed point.

For the purposes of this paper, I will change slightly the interpretation of the default rule to mean: if the speaker says '$\alpha(\vec{x})$' and $\beta(\vec{x})$ is consistent with the hearer's knowledge base, $KB_H$, then the hearer can conjecture $\beta(\vec{x})$. It is not absolutely clear what the verb *says* means or how it should be represented. For the purposes of this paper I only require those notions first presented in Grice (1975) under the title Principle of Cooperative Conversation and formalized in Gazdar (1979). Under Gazdar's interpretation of Grice's maxims the speaker is committed to the truth of $\mathbf{u}$, the sentence that he utters. Therefore the speaker knows $\mathbf{u}$. The conversational approach that I take views the contribution of a speaker's utterance $\mathbf{u}$ as the addition of $\mathsf{K}_S\mathbf{u}$ to $KB_H$ along with other conversational information which is detailed in section 5. The meaning of the utterance is then a function of the inferencing process on $KB_H \cup \{\mathsf{K}_S\mathbf{u}\}$.

The default rules require some extra information to guard against misuse of the default rules. This information is a conjunct in the prerequisite of the default rule. Except for this technical aspect this extra information plays no role. Since it creates long default rules, I have left it out of all the examples. For further details see Mercer (1987).

In the following sections I will show a method of deriving presuppositions which is based on default logic. This approach makes use of default rules to represent presuppositions (Mercer and Reiter (1982)). An example of one presuppositional environment should point out the salient features. The example will be given in some detail in order to describe the inferencing that leads to the *preferred interpretation*[3] (Wilson (1975)) of the certain aspectual 'stop'. A presupposition of a preferred interpretation of a simple sentence can be viewed as the consequent of a default rule and the preferred interpretations of vague linguistic forms are then inferences made using these assumptions.

## Example — Stop

In this example $e$ represents an event, and $t_1$ and $t_2$ are time parameters meant to represent times relevant to the event, $e$. Even though a proper representation for continuous actions has yet to be obtained, I assume here that the definition of 'stop' given in (13) is sufficient. Paraphrasing (13), an event stops *if and only if* there is a time, $t_1$, at which the event was being done and a later time, $t_2$, at which the event was not being done. By a simple negation of (13) the definition of 'not stop' given in (14) can be generated. What is important to note here is that this is the wide scoping of the logical negation operator, which gives the vague semantic definition of 'not stop'. In addition to the usual definition of 'not stop' given in (14), the default rule (15) also supplies part of the meaning of 'not

[2]The verb *believe* should be taken to mean first order derivable or conjectured.

[3]Kempson (1975) uses the term *natural interpretation*.

$$\mathbf{T_1} = \left\{ \begin{array}{c} \neg STOP(BEAT(John, r_1)) \\ \forall e. \neg STOP(e) \equiv \\ \quad \forall t_1 \forall t_2. (t_1 < t_2 \wedge DO(e, t_1)) \supset DO(e, t_2) \\ \frac{\neg STOP(e) : \exists t. DO(e, t)}{\exists t. DO(e, t)} \end{array} \right\}$$

Figure 1: A possible default theory for *John did not stop beating the rug.*

stop'. This default rule plays a crucial role in generating the preferred interpretation of 'not stop'.

(13) $STOP(e) \equiv \exists t_1 \exists t_2. t_1 < t_2 \wedge DO(e, t_1) \wedge \neg DO(e, t_2)$

(14) $\neg STOP(e) \equiv$
$\forall t_1 \forall t_2. (t_1 < t_2 \wedge DO(e, t_1)) \supset DO(e, t_2)$[4]

(15) $\frac{\neg STOP(e) : \exists t. DO(e, t)}{\exists t. DO(e, t)}$

Suppose that a speaker, $S$, utters (16). According to the rules of the communication act given in section 3, the hearer can interpret this utterance as (17). The resulting default theory, $\mathbf{T_1}$, shown in Figure 1 represents $KB_H \cup \{\mathsf{K}_S \mathbf{u}\}$ after it has undergone case analysis (see section 5). Both (18) and (19) can be derived from $\mathbf{T_1}$ using the default logic proof theory described in Reiter (1980). (18) represents the presupposition of (16). (19) represents the preferred interpretation, which can be paraphrased as there is some time at which the event $BEAT(John, r_1)$ was being done and it continues to be done at all future times.

(16) John did not stop beating the rug.

(17) $\mathsf{K}_S \neg STOP(BEAT(John, r_1))$[5]

(18) $\exists t. DO(BEAT(John, r_1), t)$

(19) $\exists t. DO(BEAT(John, r_1), t) \wedge$
$\forall t'. t < t' \supset DO(BEAT(John, r_1), t')$

On the other hand, the speaker can use the *'because'-clause* in (20) to indicate the extra qualification represented in (21) which is added to $\mathbf{T_1}$, to give $\mathbf{T_2}$, which is shown in Figure 2. Neither (18) nor (19) can be derived from the theory generated by this utterance, given in $\mathbf{T_2}$. Any derivation of (18) must include a successful invocation of the default rule (15). But in the default theory, $\mathbf{T_2}$, invocation of this rule is blocked by the sentence (21)[6].

[4] This representation is equivalent to the disjunctive notation
$\neg STOP(e) \equiv \forall t_1 \forall t_2. \neg(t_1 < t_2) \vee \neg DO(e, t_1) \vee DO(e, t_2)$
which I have used in previous discussions.

[5] $STOP(BEAT(John, r_1))$ should be interpreted as a succinct notation for the First Order representation:
$\exists e. EVENT(e) \wedge TYPE(e, BEAT) \wedge$
$SUBJ(e, John) \wedge OBJ(e, r_1) \wedge \neg STOP(e).$

[6] The 'because'-clause (21) together with the definition (14) can be used to derive
$\neg STOP(BEAT(John, r_1))$
which paraphrases as 'John did not stop beating the rug.' Although the ability to derive the main clause of the sentence may have significance, say for an analysis of relevance or causation, I am not interested in it here.

$$\mathbf{T_2} = \left\{ \begin{array}{c} \neg STOP(BEAT(John, r_1)) \wedge \\ \quad \forall t. \neg DO(BEAT(John, r_1), t)) \\ \forall e. \neg STOP(e) \equiv \\ \quad \forall t_1 \forall t_2. (t_1 < t_2 \wedge DO(e, t_1)) \supset DO(e, t_2) \\ \frac{\neg STOP(e) : \exists t. DO(e, t)}{\exists t. DO(e, t)} \end{array} \right\}$$

Figure 2: A possible default theory for *John did not stop beating the rug because he was never doing it.*

(20) John did not stop beating the rug because he was never doing it.

(21) $\mathsf{K}_S \forall t. \neg DO(BEAT(John, r_1), t)$

# 4 The Projection Problem

That presuppositions arise from lexical and syntactic environments is no longer a source of disagreement. However, the *projection problem* — how does a complex sentence inherit the presuppositions of its parts — has been a major source of disagreement. In particular, the projection class called the filters exhibits a classic problem: two sentences, (22) and (23), have the same form (two clauses joined by 'or') but have differing presuppositional properties. The first disjunct 'Mary stopped beating the rug' of the **S**-sentence in (22) presupposes[7] that 'Mary was beating the rug'. Likewise the second disjunct presupposes that 'John was beating the egg'. The sentence inherits all the presuppositions of its constituents.

(22) **S:** Mary stopped beating the rug or John stopped beating the egg.

**P:** Mary was beating the rug and John was beating the egg.

The first disjunct 'Your teacher is a bachelor' of the **S**-sentence in (23) presupposes that 'Your teacher is male and an adult'. The second disjunct 'Your teacher is a spinster' presupposes that 'Your teacher is female and an adult'. It is obvious that the sentence does not inherit all the presuppositions of its constituents.

(23) **S:** Your teacher is a bachelor or a spinster.

**P:** Your teacher is an adult.

[7] In the sense allowed by many linguists clauses can have (potential) presuppositions and presuppositional environments can be unnegated as well as negated. Although I argue against this usage in Mercer (1987), I use the verb 'presupposes' in the accepted linguistic sense so that I can explain the projection method.

**P*:** Your teacher is a male.

**P*:** Your teacher is a female.

The three standard fixes to the projection rule have been: a set of rules that take the presuppositions of the clauses and remove the undesirable presuppositions as the sentence meaning is being composed (Karttunen (1973, 1974), Karttunen and Peters (1975,1979), Soames (1979)); a set of rules, invoked after the sentence has been fully interpreted, that cancel the unwanted presuppositions from a complete set of potential presuppositions (Gazdar (1979)); or a set of rules that embody both of these methods (Soames (1982)). A non-projection rule method that interprets the sentence in a left-to-right sequential manner, can be found in (Gunji (1982)). The desired result in each case is to retain all and only the presuppositions of the complex sentence.

In Mercer (1987), I reject the projection rule paradigm and replace it with a theory that views presupposing as a form of inference. Since this paper is concerned only with the technical aspects of this theory, I refer to Mercer (1987) for a fuller discussion of the reasons for rejecting the projection rule paradigm. I now turn to the discussion of what takes the place of the projection rule in a theory which represents presuppositions as consequents of default rules.

# 5 Deriving Presuppositions in Complex Sentences

The concept discussed herein — using default logic to derive presuppositions — is strongly influenced by Gazdar's theory. Section 3 discusses how default rules in a default theory together with default logic proof theory captures Gazdar's idea of presuppositions being consistent with a context. Another influence is the use of clausal implicatures in connection with deriving presuppositions from complex sentences. In the default logic approach the clausal implicatures are used to control the division of the original theory into its first order cases. In Gazdar's theory the context is first incremented with the clausal implicatures which in appropriate situations make the clausal presuppositions inconsistent with the context (which for Gazdar's theory means that they fail to be presuppositions of the sentence). A major problem for Gazdar's approach — explaining why the implicatures are added to the context before the presuppositions — is not found in the derivational approach.

The clausal implicatures are derived from the natural language sentence according to Gazdar's formal treatment of Grice's conversational principles (Grice (1975)). The sentence uttered by a speaker commits the speaker not only to the truth of the sentence but also to the possibility of its clauses (its parts). So in the case of the speaker uttering '$A$ or $B$' or 'if $A$ then $B$', unless there is background knowledge or there are linguistic reasons to prevent it, the speaker is committed to $\mathsf{P}_S A$, $\mathsf{P}_S \neg A$, $\mathsf{P}_S B$, and $\mathsf{P}_S \neg B$. These implicatures will provide the means to restrict the division of the theory representing the utterance into its cases.

## 5.1 Choosing the Cases for the Case Analysis

Because default logic proof theory does not display any analogue to the law of the excluded middle (the antecedents of the default rules must be provable and there is no equivalent to the deduction theorem[8]) and because presuppositions do arise from the clauses of complex sentences, some form of analysis by cases is required. Since a statement is provable in a case analysis only if it is provable in *all* cases, the choice of cases is critical. As in the case of a first order theory, too few cases would allow inappropriate statements to be proved. In addition because of the non-monotonic nature of default logic, having too many cases could prevent appropriate statements being proved.

In general the choice of cases must reflect two principles. Since the case analysis is a proof theoretic analogue of the model theoretic law of the excluded middle, each case must *completely determine* the truth values of each of the disjuncts found in the statement to which case analysis is being applied. Also, since the case analysis is justified solely on linguistic grounds (see Mercer (1987) for further discussion), the cases must reflect this linguistic situation. To justify a case, the possibility of the statement that *distinguishes* the case must be provable from the original default theory. Since none of the modal statements take part in the proofs, they are left out of the cases. An example should clarify these ideas.

**Example**

Suppose the sentence '$A$ or $B$' is uttered. The default theory representing this utterance would be

$$\mathbf{T} = \{\mathsf{K}_S(A \vee B), \mathsf{P}_S A, \mathsf{P}_S \neg A, \mathsf{P}_S B, \mathsf{P}_S \neg B, \alpha_1, \ldots, \alpha_n, \delta_1, \ldots, \delta_n\}$$

where $\alpha_1, \ldots, \alpha_n$ represent the appropriate first order statements and $\delta_1, \ldots, \delta_n$ represent the appropriate default rules. Since $A \wedge \neg B$ and $\neg A \wedge B$ *completely determine* (that is, determine the truth values of *both*) $A$ and $B$, and since the statements $\mathsf{P}_S(A \wedge \neg B)$ and $\mathsf{P}_S(\neg A \wedge B)$ can be derived, $A \wedge \neg B$ and $\neg A \wedge B$ ***distinguish*** the two cases. Note that although $\mathsf{P}_S A, \mathsf{P}_S \neg A, \mathsf{P}_S B, \mathsf{P}_S \neg B$ are

[8] Besnard, Quiniou, and Quinton (1983) discuss properties of a transformed default theory in which all defaults of the form $\frac{\alpha : \beta}{\beta}$ are transformed into $\frac{: \alpha \supset \beta}{\alpha \supset \beta}$. The use of a transformed default theory rather than the case analysis approach has not been fully investigated.

all derivable, none of $A$, $\neg A$, $B$, $\neg B$ are candidates for distinguishing a case because, individually, none of them completely determine the truth values of both $A$ and $B$.

Hence the two cases of the original theory, $\mathbf{T}$, are

$$\mathbf{T_{Case1}} = \{A \wedge \neg B, \alpha_1, \ldots, \alpha_n, \delta_1, \ldots, \delta_n\}$$

$$\mathbf{T_{Case2}} = \{\neg A \wedge B, \alpha_1, \ldots, \alpha_n, \delta_1, \ldots, \delta_n\}$$

The simple negated sentence, an example of which is presented in section 3, is just a special instance of the case analysis procedure. In the simple negated sentence, $\neg X$ (which is represented as $\mathsf{K}_S \neg X$), the possibility of the only case (distinguished by $\neg X$) can be proved using the utterance and the theorem $\vdash \mathsf{K}_S \neg X \vdash \mathsf{P}_S \neg X$.

## 5.2 A Proof-Theoretic Definition of Presuppositions

**Definition 1** *A sentence $\alpha$ is a presupposition of an utterance* $\mathbf{u}$*, represented by the default theory* $\Delta_{\mathbf{u}}$[9]*, if and only if* $\Delta_{\mathbf{u}} \vdash_\Delta \alpha$[10] *and* $\alpha \in Th(CONSEQUENTS\{D\})$*, but* $\Delta_{\mathbf{u}} \not\vdash \alpha$ *and* $\Delta_{\mathbf{u}} \not\vdash_\Delta \neg\alpha$[11].

This definition can be loosely paraphrased as: if $\alpha$ is in the logical closure of the default consequents and is provable from the utterance, and all proofs require the invocation of a default rule and in the case of multiple extension default theories, $\alpha$ is in all extensions, then $\alpha$ is a presupposition of the utterance.

## 5.3 Example — *Or*: No cancellation

The discussion in section 4 indicates that sentence (24) has the conjunction of all the presuppositions that its two disjunctive clauses would have if uttered in isolation. The derivation procedure given below indicates the default proof theory approach to deriving presuppositions in complex sentences.

[9] For purposes of this definition, the only defaults in $\Delta_{\mathbf{u}}$ are the presupposition generating defaults. In reality the default theory would contain many other kinds of defaults. The definition would have to be changed so that the proof of $\alpha$, requires the invocation of a presupposition generating default, and that $\alpha \in Th(CONSEQUENTS\{D'\})$, where $D'$ is the set of presupposition generating defaults

[10] Since a case analysis is being used, $\Delta_{\mathbf{u}} \vdash_\Delta \alpha$ means that $\Delta_{\mathbf{u}_{Case_i}} \vdash_\Delta \alpha$ *for all* $i$.

[11] All of the examples presented in this paper deal with default theories having single extensions. In those theories which have multiple extensions, some way of stating that a presupposition is in all extensions is required. Since extensions of normal default theories are orthogonal, if $\Delta_{\mathbf{u}}$ has multiple extensions then there exists a sentence $\beta$ such that $\Delta_{\mathbf{u}} \vdash_\Delta \beta$ and $\Delta_{\mathbf{u}} \vdash_\Delta \neg\beta$. I will call this situation being split along the $\beta$-dimension. If the extensions do not split along the $\alpha$-dimension then either $\alpha$ is in all extensions or $\alpha$ is in no extension. So if $\Delta_{\mathbf{u}} \vdash_\Delta \alpha$ (which means that at least one extension contains $\alpha$) and $\Delta_{\mathbf{u}} \not\vdash_\Delta \neg\alpha$ (which means that no extension contains $\neg\alpha$, which means that the extensions do not split on the $\alpha$-dimension) then $\alpha$ is in all extensions.

$$\mathbf{T_3} = \left\{ \begin{array}{c} \mathsf{K}_S[STOP(BEAT(Mary, r_1)) \vee \\ \qquad STOP(BEAT(John, e_1))] \\ \forall e.STOP(e) \supset \\ \quad \exists t_1 \exists t_2 . t_1 < t_2 \wedge DO(e, t_1) \wedge \neg DO(e, t_2) \\ \quad \dfrac{\neg STOP(e) : \exists t.DO(e, t)}{\exists t.DO(e, t)} \\ \mathsf{P}_S STOP(BEAT(Mary, r_1)) \\ \mathsf{P}_S \neg STOP(BEAT(Mary, r_1)) \\ \mathsf{P}_S STOP(BEAT(John, e_1)) \\ \mathsf{P}_S \neg STOP(BEAT(John, e_1)) \end{array} \right\}$$

Figure 3: A possible $KB_H \cup \{\mathsf{K}_S \mathbf{u}\}$ for *Mary stopped beating the rug or John stopped beating the egg.*

(24) Mary stopped beating the rug or John stopped beating the egg.

In the same manner that was described in section 3 $\mathbf{T_3}$, which is displayed in Figure 3, is the $KB_H \cup \{\mathsf{K}_S \mathbf{u}\}$ produced as a result of (24) being uttered. The first three statements are exactly what is found in the theory described in section 3: the representation of the sentence, the first order definition of *STOP*, and the default rule for $\neg STOP$. In addition to these statements the enhanced theory now requires the next four statements which are the clausal implicatures derived from the disjunctive sentence. The two statements described in (25) are derivable from $\mathbf{T_3}$. Case analysis can be applied to the cases represented in the bodies of these two statements.

(25) $\mathsf{P}_S STOP(BEAT(Mary, r_1)) \wedge \neg STOP(BEAT(John, e_1))$
$\mathsf{P}_S \neg STOP(BEAT(Mary, r_1)) \wedge STOP(BEAT(John, e_1))$

I will now detail the derivation as it proceeds in the two cases. Rather than giving the two complete theories, $\mathbf{T_{3_{Case1}}}$ and $\mathbf{T_{3_{Case2}}}$, I will give only the first order statement that distinguishes each theory.

$\mathbf{T_{3_{Case1}}}$: $STOP(BEAT(Mary, r_1)) \wedge \neg STOP(BEAT(John, e_1))$.

The conjuncts (26) and (29) are derivable. Using (26), (27), universal instantiation, *modus ponens*, existential specification, derivation of a conjunct, and existential generalization, (28) can be derived.

(26) $STOP(BEAT(Mary, r_1))$

(27) $\forall e.STOP(e) \supset \exists t_1 \exists t_2 . t_1 < t_2 \wedge DO(e, t_1) \wedge \neg DO(e, t_2)$

(28) $\exists t.DO(BEAT(Mary, r_1), t)$

Using (29), the default rule (30), and default proof theory, (31) can be derived.

(29) $\neg STOP(BEAT(John, e_1))$

(30) $$\frac{\neg STOP(e) : \exists t.DO(e,t)}{\exists t.DO(e,t)}$$

(31) $\exists t.DO(BEAT(John, e_1), t)$

The conjunction of (28) and (31) gives (32). Note (32) is derivable using default proof theory but not using first order methods alone.

(32) $\exists t.DO(BEAT(Mary, r_1), t) \wedge \exists t.DO(BEAT(John, e_1), t)$

$\mathbf{T3_{Case2}}$: $\neg STOP(BEAT(Mary, r_1)) \wedge STOP(BEAT(John, e_1))$.

The derivation of (33) proceeds in a manner similar to the derivation of (32) except that the roles of $BEAT(John, e_1)$ and $BEAT(Mary, r_1)$ have been interchanged.

(33) $\exists t.DO(BEAT(Mary, r_1), t) \wedge \exists t.DO(BEAT(John, e_1), t)$

Because the statement in (32) and (33) is derivable in both cases, the presupposition generating default is required in its derivation, and it is in the logical closure of the default consequents, it is a presupposition of (24). This result should be predicted by any adequate theory of presuppositions.

## 5.4 Example — *Or*: Intrasentential cancellation

If the extra real world knowledge — that an egg can be beaten only by one person — is part of the knowledge base in which the derivation of the presupposition is done, then sentence (34) is an example of intrasentential cancellation of clausal presuppositions.[12] In terms of the theory presented here cancellation of clausal presuppositions is a failure to derive the conjunction of those inferences which would be derived if the disjuncts were used separately (in an appropriate context).

(34) Mary stopped beating the egg$_i$ or John stopped beating the egg$_i$.

The hearer's knowledge base $\mathbf{T_4}$, which is generated as a result of (34) being uttered, is displayed in Figure 4. The contents are the meaning postulates concerned with the concept *STOP*, the statement that eggs can be beaten only by one person, four first order modal statements (results of the clausal implicature rule), knowledge about Mary, John, and the egg, and the one equality axiom required in the following discussion. The two statements described in (35) are derivable from $\mathbf{T_4}$. Case analysis can be applied to the cases represented in the bodies of these two statements.

[12] I thank Alan Mackworth for this example.

(35) $P_S STOP(BEAT(Mary, e_1)) \wedge \neg STOP(BEAT(John, e_1))$
$P_S \neg STOP(BEAT(Mary, e_1)) \wedge STOP(BEAT(John, e_1))$

I will now detail how the derivation of the conjunction of the presuppositions of the two clauses is prevented. Rather than giving the two complete theories, $\mathbf{T_{4_{Case1}}}$ and $\mathbf{T_{4_{Case2}}}$, I will give only the first order statement that distinguishes each theory.

$\mathbf{T_{4_{Case1}}}$: $STOP(BEAT(Mary, e_1)) \wedge \neg STOP(BEAT(John, e_1))$.

The conjuncts (36) and (39) are derivable. Using (36), (37), universal instantiation, *modus ponens*, existential specification, derivation of a conjunct, and existential generalization, (38) can be derived.

(36) $STOP(BEAT(Mary, e_1))$

(37) $\forall e.STOP(e) \supset \exists t_1 \exists t_2.t_1 < t_2 \wedge DO(e, t_1) \wedge \neg DO(e, t_2)$

(38) $\exists t.DO(BEAT(Mary, e_1), t)$

(41) can be derived only by resorting to default proof theory. But the default rule (40), its antecedent (39), and default proof theory, cannot derive (41). Since (38) is derivable from first order principles, it must be in the fixed point of $\mathbf{T_{4_{Case1}}}$. The default rule (40), with $e$ instantiated to $BEAT(John, e_1)$, is blocked because its justification is not consistent with this fixed point. Intuitively, both Mary and John could not have beaten the same egg.

(39) $\neg STOP(BEAT(John, e_1))$

(40) $$\frac{\neg STOP(e) : \exists t.DO(e,t)}{\exists t.DO(e,t)}$$

(41) $\exists t.DO(BEAT(John, e_1), t)$

$\mathbf{T_{4_{Case2}}}$: $\neg STOP(BEAT(Mary, e_1)) \wedge STOP(BEAT(John, e_1))$.

The derivation of (42) proceeds in a manner similar to the derivation of (38) except that the roles of $BEAT(John, e_1)$ and $BEAT(Mary, e_1)$ have been interchanged.

(42) $\exists t.DO(BEAT(John, e_1), t)$

Since (38) is derivable from $\mathbf{T_{4_{Case1}}}$ and (41) from $\mathbf{T_{4_{Case2}}}$, (43) is derivable in both cases. Since this statement can be deduced from $\mathbf{T_4}$ using only first order proof theory, it is an entailment. Except for the minor difference in interpreting (43) as an entailment rather than as a presupposition of the utterance, this analysis coincides

$$\mathbf{T_4} = \left\{ \begin{array}{c} \mathsf{K}_S\left[STOP(BEAT(Mary, e_1)) \vee STOP(BEAT(John, e_1))\right] \\ \forall c.STOP(c) \supset \exists t_1 \exists t_2 . t_1 < t_2 \wedge DO(c, t_1) \wedge \neg DO(c, t_2) \\ \forall x \forall y \forall z_1 \forall z_2 \forall t_1 \forall t_2 . EGG(z_1) \wedge EGG(z_2) \wedge \\ DO(BEAT(x, z_1), t_1) \wedge DO(BEAT(y, z_2), t_2) \wedge x \neq y \supset z_1 \neq z_2 \\ \dfrac{\neg STOP(c) : \exists t . DO(e, t)}{\exists t . DO(e, t)} \\ \mathsf{P}_S STOP(BEAT(Mary, e_1)) \\ \mathsf{P}_S \neg STOP(BEAT(Mary, e_1)) \\ \mathsf{P}_S STOP(BEAT(John, e_1)) \\ \mathsf{P}_S \neg STOP(BEAT(John, e_1)) \\ Mary \neq John \\ EGG(e_1) \\ \forall x . x = x \end{array} \right\}$$

Figure 4: A possible $KB_H \cup \{\mathsf{K}_S \mathbf{u}\}$ for *Mary stopped beating the egg$_i$ or John stopped beating the egg$_i$.*

with what is predicted by any reasonable presuppositional theory. (Some investigations call these entailments trivial presuppositions (Soames (1982)). What is important is that the conjunction of (38) and (42) is not derivable.

(43) $\exists t . DO(BEAT(Mary, e_1), t) \vee \exists t . DO(BEAT(John, e_1), t)$

# 6 Conclusions

The semantic representations of natural language sentences can be *vague* (that is, it can be true under a variety of *truth conditions*). Because vagueness contravenes Grice's Principle of Cooperative Conversation this anomaly must be removed. The ambiguity caused by vagueness is resolved according to *pragmatic rules*. Because the pragmatic rules are defeasible and conjectural in nature, they are captured as *default rules*. The position is taken that presuppositions are inferences generated from these pragmatic rules. Presuppositions are then used to generate the *preferred interpretation* of the vague representation. This paper discusses the technical aspects of using default proof theory together with a case analysis to generate presuppositions for various natural language sentences.

# References


Atlas, J. D. (1977), "Negation, Ambiguity, and Presupposition", *Linguistics and Philosophy* **1**:321–336.

Besnard, Ph., R. Quiniou, and P. Quinton (1983), "A Theorem-Prover for a Decidable Subset of Default Logic", *Proceedings of AAAI-83*, pp 27–30.

Carnap, R. (1956), *Meaning and Necessity*, University Press.

Gazdar, G. J. M. (1979), *Pragmatics: Implicature, Presupposition, and Logical Form*, Academic Press.

Grice, H. P. (1975), "Logic and Conversation" in *Syntax and Semantics*, v.3, *Speech Acts*, P. Cole and J. L. Morgan (eds), Academic Press, pp 41–58.

Gunji, T. (1982), *Toward a Computational Theory of Pragmatics: Discourse, Presupposition, and Implicature*, Indiana University Linguistics Club.

Karttunen, L. (1973), "Presuppositions of Compound Sentences", *Linguistic Inquiry* **4**:169–193.

Karttunen, L. (1974), "Presupposition and Linguistic Context", *Theoretical Linguistics* **1**:181–194.

Karttunen, L. and S. Peters (1975), "Conventional Implicature in Montague Grammar", *Proceedings of the First Annual Meeting of the Berkeley Linguistics Society*, pp 266–278.

Karttunen, L. and S. Peters (1979), "Conventional Implicature" in *Syntax and Semantics*, v.11, *Presuppositions*, C.-K. Oh and D. A. Dineen (eds), Academic Press, pp 1–56.

Kempson, R. M. (1975), *Presupposition and the Delimitation of Semantics*, Cambridge University Press.

Mercer, R. E. (1987), *A Default Logic Approach to the Derivation of Natural Language Presuppositions*, Ph.D. Thesis, Dept. of Computer Science, University of British Columbia.

Mercer, R. E. and R. Reiter (1982), "The Representation of Presuppositions Using Defaults", *Proceedings of the Fourth Biennial Conference of the CSCSI/SCEIO*, pp 103–107.

Reiter, R. (1980), "A Logic for Default Reasoning", *Artificial Intelligence* **13**:81–132.

Soames, S. (1979), "A Projection Problem for Speaker Presuppositions", *Linguistic Inquiry* **10**:623–666.

Soames, S. (1982), "How Presuppositions are Inherited: A Solution to the Projection Problem", *Linguistic Inquiry* **13**:483–545.

Wilson, D. (1975), *Presuppositions and Non-Truth-Conditional Semantics*, Academic Press.

# An Evidence Oracle for Argument Understanding

Mark A Young
Department of Computer Science
University of Waterloo*
Waterloo, Canada

Robin Cohen
Department of Computer Science
University of Waterloo
Waterloo, Canada

## Abstract

When trying to understand a speaker's argument, it is necessary to determine what his claim is and what evidence he provides for it. It is necessary, therefore, to be able to recognise evidence relations in terms of the speaker's beliefs. This paper describes an implementation of an *Evidence Oracle*, which tests for evidence between statements and builds a model of the speaker based on the evidence relations found. This implementation constitutes a valuable achievement in the development of practical discourse analysis systems, proposing a basis for verifying certain relationships between utterances. Another contribution of the work is a stratified speaker model which allows for varying levels of acceptance of beliefs attributed to the speaker. Some extensions of this approach for plan inference are also discussed.

## 1 Introduction

This paper is a short description of the *Evidence Oracle* more fully described in [Young87]. The oracle takes two propositions, $Q$ and $P$, and answers the question "Does the speaker mean $P$ to be evidence for $Q$?" In coming to its answer the oracle considers its own model of the world and a model of the speaker. Based on the answer it derives, it may build on the speaker model.

### 1.1 The Argument Understanding System

The *EO* is based on the subsystem of the same name in Cohen's *Argument Understanding System (AUS)*. The *AUS* (described in [Cohen83, Cohen87a] and partially implemented in [Smedley86, Smedley87]) parses a Natural Language argument (monologue) into a tree representation. The root of the tree is the main claim of the speaker, and the children of each node are the statements made in evidence for that node. By identifying and using a limited number of coherent *transmission strategies*, Cohen was able to restrict the number of calls to the *EO* to be linear in the number of statements in the argument.

Figure 1 shows the flow of control and data in the *AUS*. The propositions of the argument are passed one at a time to the *Proposition Analyser (PA)*, which, based on the restrictions of the transmission strategy, selects those nodes in the argument representation which may be related to the current proposition. For each of the selected nodes, the question of evidence is asked of the *EO*. When an evidence relation is found, the current node is inserted at the appropriate position and the *AUS* goes on to the next proposition. The *Clue interpreter* finds and analyses clue words in the argument. These are special words and phrases used by the speaker to indicate the structure of the argument (e.g. connectives). These clues may restrict the nodes considered in the search of the *Proposition Analyser*, or the types of evidence relations considered for each node pairs (passing the clue information to the *Evidence Oracle*).

The transmission strategies the *AUS* accepts are:

1. claim first—each claim followed by the evidence for it.
2. claim last—each claim preceded by the evidence for it.
3. hybrid—each sub-argument is either claim first or claim last.

If the argument is coherent and the analysis is correct then the representation returned will be a tree. It is expected that this tree will be passed on to some sort of *Response Unit (RU)*, which will generate a reply to the argument. It is the *RU* which will deal with

*Now at Bell-Northern Research Ltd, Ottawa

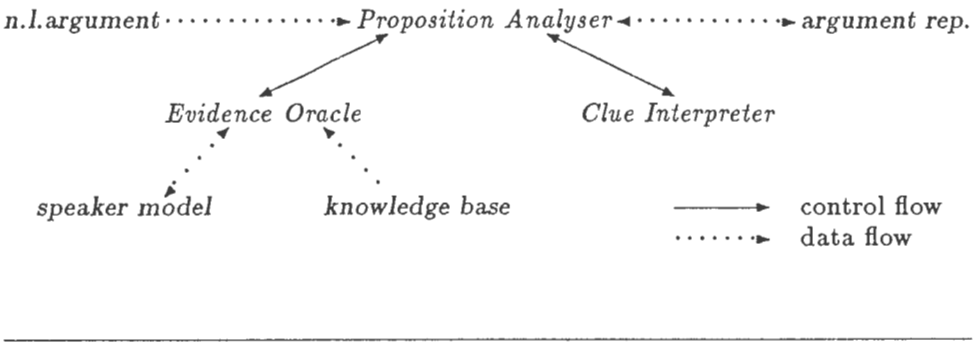



Figure 1: System Design

the believability of the overall argument; the *EO*, in contrast, is concerned with the believability of the individual evidence relations in the argument—the basis for deciding the yes/no answer for the *PA*.

# 2 The Evidence Oracle

## 2.1 Frames of Evidence

To recognise evidence relations the *EO* uses frames with slots for a conclusion and one or more premises. We say that $E$ is *evidence* to $C$ (*claim*) if $E$ and $C$ fit some frame of evidence with $E$ as a premise and $C$ as the conclusion. The frames used appear in tables 1 and 2. *Modus Ponens* and *Modus Tollens* represent the rules of logic. The Generalization is a variation on *Modus Ponens* in which the correspondence is not perfect; that is, it need not be true for all values of its free variables. It may be viewed as either a default rule (as in [Reiter80] or [Poole&al86]) or as a relation which is merely probable. The generalized rule corresponding to $A \rightarrow B$ is represented by $A \rightarrow\!\triangleright B$.

Refutation and Concession appear in counter-arguments, and serve to deny a rule proposed by another speaker. The "rule" is contradicted by giving a counter-example—the refutation. Before showing the counter-example, though, it is usual to lead the way with concessions—examples of the "rule" at work. These concessions are a form of contrastive evidence, as discussed in [Cohen83].

Table 2 shows frames of *partial* evidence. These forms of evidence allow for "giving examples". There is an implied rule at work here:

> If the examples for a generalization outweigh the examples against, then that generalization is (probably) true.

The above rule may not be particularly convincing (to say nothing of the generalization it is being used to support), but the usage is common.

The oracle should also be prepared to recognise incorrect rules of logic, such as "Asserting the Consequent", if there is evidence that the speaker is using them. For now we will restrict our attention to the above frames.

## 2.2 Missing Premises

The speaker often does not fill in all slots of the applicable frame. This phenomenon is called *Modus Brevis* in [Sadock77]. Thus, while the speaker could say

> Socrates is a man. All men are mortal. Therefore, Socrates is mortal.

he is more likely to say either

> All men are mortal. Therefore, Socrates is mortal.

or (even worse)

> Socrates is a man, and, therefore, mortal.

The listener is expected to fill in the empty slots from his own knowledge.

It is important that the missing premises could plausibly be held by the speaker. That the missing premises contradict shared beliefs or earlier statements of the speaker could indicate that the relation found is not the one intended.

[Cohen83] suggests that we determine whether a belief is plausible by consulting the following sets of knowledge:

1. Shared beliefs

| | Minor Premise | Major Premise | Conclusion |
|---|---|---|---|
| *Modus Ponens* | $A$ | $A \rightarrow B$ | $B$ |
| *Modus Tollens* | $\neg B$ | $A \rightarrow B$ | $\neg A$ |
| Generalization | $A$ | $A \triangleright B$ | $B$ |
| Refutation | $A$ | $\neg B$ | $\neg(A \rightarrow B)$ |
| Concession | $A$ | $B$ | $\neg(A \rightarrow B)$ |

Table 1: Frames of Evidence

| | Minor Premises | | Conclusion | | |
|---|---|---|---|---|---|
| Positive Example | $A$ | $B$ | $A \rightarrow B$ | or | $A \triangleright B$ |
| Contrapositive Example | $\neg B$ | $\neg A$ | $A \rightarrow B$ | or | $A \triangleright B$ |
| Counterexample | $A$ | $\neg B$ | | | $A \triangleright B$ |

Table 2: Frames of Partial Evidence

2. The hearer's beliefs
3. A stereotype of the speaker
4. A model of a hypothetical person—a "least detailed" speaker model.

To facilitate the implementation, we shall take a slightly different approach. The oracle will consult:

1. Shared beliefs
2. A model of the speaker, including a stereotype
3. The system's beliefs/knowledge.

We assume the speaker is reasonably competent at logic. Therefore, if he says that $P$ is evidence for $Q$ by virtue of the relation $P \wedge R \rightarrow Q$, then he believes not only that $P$, $Q$, and $P \wedge R \rightarrow Q$ are true, but also that $R$ is true. On this basis, any premises missing from an accepted evidence relation may be added to the speaker model.

## 2.3 Model of the World

The oracle maintains a model of the world to help it judge the plausibility of missing premises. Part of that world model is a model of the speaker. The world model is broken into several modules, depending on who holds the beliefs represented:

**facts** the beliefs common to the conversants[1].

**speaker** the speaker's beliefs, broken down into:

**explicit** those statements made by the speaker, and thus attributed to himself.

**missing** beliefs we have attributed to the speaker on the basis of evidence relations determined earlier in the argument.

**stereotype** default beliefs for the speaker. The system's beliefs will initially serve as a basis for the speaker stereotype.

**hearer** the private beliefs (or knowledge) of the system.

The ordering of modules given above reflects the order of search for missing beliefs. Roughly speaking, the further down the list one must search for a missing belief, the less plausible it is that the speaker has that belief. Predicates that do not appear in the list, however, may also be judged plausible. In particular, if a predicate is not explicitly contradicted by one of *facts, explicit,* or *missing,* then it can be considered plausible.

## 2.4 Determining Evidence Relations

To show how frames are used to determine evidence relations, consider the instantiation of the *Modus Ponens* frame given in table 3. The speaker gives the speech on the left (the statements are numbered for later reference), which is translated into the predicates on the right. (Our implementation of the evidence oracle, in Waterloo Unix Prolog (WUP), assumes that the input has already been pre-processed into the predicate notation)[2]. The oracle can then fit the predicates into the slots for *Modus Ponens* and recognise the relation intended.

Assuming a claim last transmission, and that the statements are made in the order given above, the *PA*

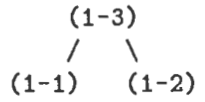
[1] These are referred to as shared beliefs. This may be seen as one-sided mutual belief, as in [Clark&Marshall81].

[2] The implementation was tested on a number of examples, including several much longer than the ones presented in this paper for illustration.

| | | |
|---|---|---|
| (1-1) *All men are mortal.* | Major Premise | $mortal(X) \leftarrow man(X)$ |
| (1-2) *Socrates is a man.* | Minor Premise | $man(socrates)$ |
| (1-3) *Therefore, Socrates is mortal.* | Conclusion | $mortal(socrates)$ |

Table 3: An instantiation of the *Modus Ponens* frame

first asks the *EO* whether (1-1) is evidence for (1-2). The oracle answers no, since there is no frame of evidence appropriate for the proposed relation. The *PA* defers handling (1-1) for the present. The next question asked is whether (1-2) is evidence for (1-3). The oracle discovers that (1-2) and (1-3) fit the *Modus Ponens* frame with (1-1) as the missing premise. Since (1-1) appears in *explicit* (it was put there on the first call to the oracle), it is deemed a plausible belief and the relation is accepted. The *PA* then completes processing the argument by asking whether (1-1) is evidence for (1-3). This succeeds in the same manner as (1-2) for (1-3) did, with the same result. The argument has been successfully parsed into the tree:




To illustrate adding missing premises to the speaker model, consider the following. Assume the speaker and system have a common belief that all greek men are mortal (represented by $mortal(X) \leftarrow greek(X) \wedge man(X)$). The speaker says "Socrates is greek, and therefore mortal." The oracle can recognise *greek(socrates)* as evidence for *mortal(socrates)* with two missing pieces: the rule about the mortality of greek men, and a belief that socrates is a man. As long as the speaker's beliefs do not indicate that Socrates is not a man we can accept that belief as plausible. The predicate *man(socrates)* would then be added to module *missing*.

## 2.5 Multiple Evidence Relations

Sometimes more than one frame of evidence may be appropriate for a given pair of predicates. Consider the following argument

| | |
|---|---|
| (2-1) Socrates is Greek, | *greek(socrates)* |
| (2-2) and, so, mortal. | *mortal(socrates)* |

with shared beliefs

$$mortal(X) \leftarrow greek(X) \wedge man(X)$$
$$mortal(X) \leftarrow greek(X) \wedge woman(X)$$

There are two frames of *Modus Ponens* that allow (2-1) as evidence for (2-2). One has Socrates as a man, the other as a woman[3]

If one of these beliefs is plausible and the other not, then there would be no problem to choose between them. When both are plausible, however, we must make a decision. The simplest solution is to take whichever is more convenient—the first one generated, for example. Another solution is to refrain from choosing—simply report a relation without updating *missing*. The former solution could lead to many errors, while the latter results in lost information. Keeping disjunctive knowledge in *missing* is another possibility, but would require a more complex management of the *missing* module. We prefer to make take some decision on the intended interpretation, in accordance with the general strategy of the *AUS* to incrementally reconstruct the representation for the input.

## 2.6 Belief Levels

Our solution is as follows: when faced with multiple evidence relations the oracle will select the most plausible. A predicate is not plausible if its negation appears in any of *facts*, *explicit*, or *missing*. The beliefs considered plausible (in order of decreasing plausibility) are

1. beliefs the speaker has attributed to himself
2. beliefs the oracle has attributed to the speaker
3. belief typical to the speaker's stereotype
4. beliefs not contradicted explicitly by 1 or 2.

Table 4 gives numerical values to the various acceptable combinations. Conjunctions receive the value of the least plausible conjunct. (Note that 0 indicates the most plausible, and 11 the least. Only predicates without a value on the above table are implausible.)

[3]Actually, this is a case where there are two possible major premises for the same frame of evidence. There are examples as well where two different frames may apply - e.g. Concession or Refutation. See [Young87] for more details.

| Truth value in *explicit/missing* | Truth value in *stereotype* | | |
|---|---|---|---|
| | True | Unknown | False |
| True/True | 0 | 1 | 2 |
| True/Unknown | 3 | 4 | 5 |
| Unknown/True | 6 | 7 | 8 |
| Unknown/Unknown | 9 | 10 | 11 |

Table 4: Relative Plausibility

A predicate is *true* in a module if it appears in that module, *false* if its negation appears there, and *unknown* otherwise. Based on the truth values found, the oracle assigns a numerical value representing plausibility. This internal representation allows comparisons to be made more easily.

If the system is aware that Socrates was a man and is using its own beliefs as a stereotype of the speaker, then it will judge it more plausible that the speaker also believes Socrates to be a man than that he believes him to be a woman (see figure 2).

## 2.7 Inconsistent Beliefs

As was said earlier, predicates which are explicitly contradicted by the speaker's beliefs are rejected as implausible. The restriction to *explicit* contradiction, as opposed to provable falseness, is motivated by recent work in modelling belief. Human beings are imperfect reasoning agents (some reasons for which are given in [Fagin&Halpern85]), and thus may be unaware of implicit contradictions in their arguments. [Levesque84] allows that a speaker may be aware that $A$ implies $C$ and that $B$ implies $\neg C$ and still believe both $A$ and $B$. A modification of [Levesque84] by [Fagin&Halpern85] restricts an agent from believing both $C$ and $\neg C$ explicitly[4] (though [Levesque84] allowed this).

## 2.8 Missing Major Premise

When one of the missing premises of an argument is the major premise, the judgement of plausibility is carried out somewhat differently than is described above. Clearly, any two premises can fit the *Modus Ponens* frame with an appropriate missing major premise. For this reason, no such major premise is generated unless it appears somewhere in the system's knowledge of the world.

On the other hand, we must allow that it is possible that the intended rule is one that the system has not previously encountered, yet which is still valid. Therefore, if no other frame of evidence is found to fit the given predicates, the oracle will generate a Generalization major premise and test that premise for plausibility.

The tests carried out to judge the plausibility of a self-generated major premise are based on philosophical definitions of evidence, particularly those of [Nathan80]. [Nathan80] provides four rules to test whether one proposition is evidence for another. The first version of the *EO* using a less sophisticated version of these tests, will merely make a count of examples and counterexamples to the proposed generalization.

For example, if the oracle is asked whether *shark(joey)* is evidence for *dangerous(joey)*, and it finds no other frames of evidence appropriate, it will test *shark(X)* $\rightarrow\!\triangleright$ *dangerous(X)* for plausibility. If it finds more dangerous sharks than non-dangerous ones, and it finds more non-dangerous non-sharks than non-dangerous sharks, it will judge the relation plausible. If either of the tests fails then the relation is rejected. In this way the oracle may generate new rules based on its observations.

# 3 Related Work

## 3.1 Plan Inference

The general problem of plan inference has been addressed recently by several researchers, including [Kautz87, Pollack86, Carberry87]. [Pollack86] states that it would be useful to have a model of plan inference that distinguishes the beliefs of the planner from those of the observer. The reason for this distinction is to allow the observer to recognise incorrect plans and generate appropriate responses. The *EO* makes such a distinction in the context of argument understanding, and can be applied to plan inference as well.

By identifying claim with plan, and evidence with subplan, the oracle can detect when one stated goal of the user is a subgoal of another. With the distinction between *explicit* and *missing* in the speaker model, the oracle can help isolate the cause of the

[4] Actually, explicit contradiction is allowed, but only between differing states of mind. See the article for more information.

hearer's beliefs: *man(socrates)*
shared beliefs: *mortal(X)* ← *greek(X)* ∧ *man(X)*
*mortal(X)* ← *greek(X)* ∧ *woman(X)*

```
considering greek(socrates) for mortal(socrates)

 greek(socrates) is evidence for mortal(socrates)
  with [man(socrates)] missing.
  man(socrates) has belief level 9
 The conjunction has belief level 9

 greek(socrates) is evidence for mortal(socrates)
  with [woman(socrates)] missing.
  woman(socrates) has belief level 10
 The conjunction has belief level 10

        success greek(socrates) for mortal(socrates)
        missing [man(socrates)]
```

Figure 2: Sample Session: Multiple Relations

user's error, especially when more than one explanation of the error exists. The oracle can also detect plans from questions regarding the conditions on a plan ([Young87, p 50]).

To elaborate on the potential use of the stratified speaker model of the *Evidence Oracle* for plan inference, consider the following example, from [Pollack86]:

> I want to prevent Tom from reading my mail file. How can I set the permissions on it to faculty read only?

with the translation into predicates as follows:

(1)*prevent (mmfile read tom)*
(2)*set-permission (mmfile read faculty)*

and a starting set of rules in the system of the following form:

*prevent (F P U)* ←
*set-permissions (F P G)*
*lnot (member (U G))*
*lnot(system-mgr (U))*

*set-permissions (F P G )* ←
*valid-permission (P G)*
*type ("SET PROTECTION (G : P) F")*

Using its own beliefs as a basis for the stereotype of the user, the system infers that she intends to do (1) by doing (2). Missing beliefs that Tom is not a member of the faculty and that he is not the system manager may be filled in. If one of these beliefs is wrong (in the system's view), then the plan is, in Pollack's terminology, ill-formed. The *Response Unit* could provide a reply by using the general rules above. Now, the division of beliefs into *explicit*, *missing* and *stereotype* may provide some more information. If the rule itself is in *missing*, then the source of the confusion may be an incorrect understanding of the conditions on the plan. If, however, all missing beliefs are also believed true by the system, then the plan may still be incorrect - e.g. valid-permission (faculty read) could be false, and the plan thus unexecutable. The *Response Unit* could provide a different kind of response, in this case.

[Carberry87] considers assumptions made by plan recognition systems that she feels ought not to be made. One of these is that the user does not have incorrect beliefs about the domain. The oracle does not make this assumption. Another is that the user's statements are correct and not misleading. This is related to the previous assumption, in that the *EO* allows for incorrect statements. If the user's statements lead to contradictions in the plan, the oracle will detect these. Otherwise the *Response Unit* must deal with whether the speaker's unsupported word should be accepted.

The oracle makes an attempt to deal with another assumption, that the user's queries always address aspects of the task within the system's knowledge of the domain. The oracle will try to recognise novel

methods, and to follow new premises laid out by the speaker. The oracle, however, does appeal to its own knowledge to judge plausibility. In this respect, it works best in limited domains where the relevant rules are likely to be already stored.

The final assumption is that no errors are introduced into the speaker model. The oracle cannot prevent incorrect inferences from being made, no more than people can; however, it does try to keep the speaker consistent, and keeps what the speaker said separate from what it infers he believes. It also saves its work, to make it easier to go back and fix the model when an error is detected.

### 3.2 Argument Understanding

This work on the evidence oracle can be merged with existing implementations of the *AUS* ([Smedley86, Smedley87]), providing a prototype working version of the overall model of [Cohen83].

In [Smedley86], the basic proposition analyser is implemented. The evidence oracle is replaced by an "ask-the-user" facility. [Smedley87] augments this basic analysis algorithm to include the acceptance of connective clues, to help restrict the processing and detect incoherent arguments (according to the algorithm of [Cohen87b]).

The current, stand-alone implementation of the oracle has some extra features. Some—such as a parser for an entire claim-first argument—are to facilitate demonstrations. Others—such as the ability to backtrack through incorrect decisions—were design decisions to simplify the work. These features can be bypassed and the oracle assimilated directly into the implementations of [Smedley86, Smedley87], replacing appeals to the user. (See [Young87] for the details). This integrated code will be thoroughly tested in an upcoming research project.

## 4 Conclusion

In this paper we have described an implementation of a module for deciding a question of evidence in the context of argument understanding. In particular, the module answers the question "Does the speaker intend his statement $P$ to be evidence for his statement $Q$?"

To decide the question, the oracle uses frames—prototypes of evidence relations. Each relation found must match one of these frames. If more than one relation is found, then considerations of user and system beliefs, with a ranking provided by "belief levels", provide an indication of the most likely intention of the speaker.

The implemented oracle thus makes possible a full implementation of an argument understanding system based on the model of [Cohen83], critical to the advancement of practical processing models of discourse.

As it answers the questions posed to it, the oracle takes note of what beliefs are required to support the relations found, and ensures that these are plausible. If the relation is accepted, this "keyhole recognition" is used to expand the speaker model. The system does not, however, give as much weight to these inferred beliefs as to explicit statements by the speaker. Moreover, for the stand-alone version of the oracle, if some contradiction is found in the speaker's argument, the oracle has the ability to revise the inferred beliefs to restore consistency.

The oracle might also be useful in recognising plans, particularly those that are only hinted at by the speaker. The two problems are similar in that some hierarchy applies, and that not all relevant components are mentioned. Therefore, the general proposals for belief recognition and updating of the speaker model have useful extensions for other natural language understanding tasks.

## Acknowledgements

Many thanks to Peter van Beek and David Poole for their comments on earlier versions of this paper, and to John Sellens for his technical assistance in producing the final copy. This research was partially supported by NSERC (Natural Sciences and Engineering Research Council of Canada).

## References

[Carberry87] Sandra Carberry, "Plan Recognition in User Modelling," to appear in *Computational Linguistics,* 1987.

[Clark&Marshall81] H H Clark and C R Marshall, "Definite reference and mutual knowledge," in A Joshi, B Webber and I Sag (eds) *Elements of Discourse Understanding,* Cambridge University Press, 1981.

[Cohen83] Robin Cohen, "A Computational Model for the Analysis of Arguments," University of Toronto Technical Report CSRG-151, October, 1983.

[Cohen87a] Robin Cohen, "Analyzing the Structure of Argumentative Discourse," *Computational*

*Linguistics,* Volume 13, Nos. 1–2, January-June 1987, pp 11–24.

[Cohen87b] Robin Cohen, "Interpreting Clues in Conjunction with Processing Restrictions," *Proceedings of AAAI-87,* 1987, pp 528–533.

[Fagin&Halpern85] Ronald Fagin and Joseph Y Halpern, "Belief, Awareness, and Limited Reasoning: Preliminary Report," *Proceedings of IJCAI-85,* 1985, pp 491–501.

[Kautz87] Henry Kautz, "A Formal Theory of Plan Recognition", Department of Computer Science, University of Rochester Technical Report TR 215, May, 1987.

[Levesque84] Hector J Levesque, "A Logic of Implicit and Explicit Belief," Fairchild Technical Report No. 653, and FLAIR Technical Report No. 32, August, 1984.

[Nathan80] N M L Nathan, *Evidence and Assurance,* Cambridge University Press, 1980.

[Pollack86] Martha Pollack, "A Model of Plan Inference that Distinguishes between the Beliefs of Actors and Observers," *Proceedings of ACL-86,* New York, NY, 1986.

[Poole&al86] David Poole, Randy Goebel and Romas Aleliunas, "Theorist: A logical reasoning system for defaults and diagnosis," University of Waterloo Research Report CS-86-06, February, 1986.

[Reiter80] R Reiter, "A logic for default reasoning," *Artificial Intelligence 13 (1&2),* 1980, pp. 81–132.

[Sadock77] J Sadock, "Modus Brevis: The Truncated Argument," in *Papers from the 13th Regional Meeting, Chicago Linguistic Society,* 1977.

[Smedley86] Trevor J Smedley, "An Implementation of a Computational Model for the Analysis of Arguments," University of Waterloo Research Report CS-86-26, July, 1986.

[Smedley87] Trevor J Smedley, "Integrating Connective Clue Processing into the Argument Analysis Algorithm Implementation," University of Waterloo Research Report CS-87-34, 1987.

[Young87] Mark A Young, "The Design and Implementation of an Evidence Oracle for the Understanding of Arguments," University of Waterloo Research Report CS-87-33, June, 1987.

# System X: A Portable Natural Language Interface

**Paul McFetridge, Gary Hall, Nick Cercone, and W.S. Luk**
Laboratory for Computer and Communications Research
School of Computing Science
Simon Fraser University
Burnaby, British Columbia, CANADA V5A 1S6

**Abstract**

**System X** is a natural language interface which currently translates English into the *de facto* standard relational database language SQL. The system consists of a set of independent modules, a Lexicon, a Parser, and a Semantic Interpreter, which create a canonical query representation. This canonical form represents the join path implicit in the query. A second set of modules translate the canonical form into a logical form. This logical form is currently translated into SQL. Subsequent versions will translate the logical form into different database languages. We describe the operation of the system and include discussion of the major advantages of System X with respect to current natural language interfaces to databases. These advantages include the degree of *portability* of System X, the (heuristic) solution to the *multiple access path problem* (MAPP) and *learn* the meanings of new words.

## 1. Introduction

Natural Language Understanding systems (NLUs) require world knowledge in order to understand input questions. Natural language database interface systems (NLIs) store world knowledge in a number of ways. All systems have semantic information stored in a lexicon. Some systems have domain specific semantic information encoded in rules for transforming queries from one internal (logical) form into another. An example of such a system is **TQA**, see (Johnson 1984). Some current systems maintain a knowledge structure separate from the database which is used to interpret queries. For example, **TEAM** (Martin et al., 1985) embodies a conceptual schema of general world knowledge and integrates into this structure knowledge specific to the application domain when customizing the system to a given database. A similar approach is taken by **Datalog**, see (Hafner and Godden 1985), which employs a semantic network of general concepts augmented by a set of database concept nodes to create a knowledge base.[1]

A significant obstacle in constructing NLI systems of any utility is the acquisition of this knowledge. Each application represents a certain domain. A NLI system must acquire knowledge of this domain and of the natual language (NL) expressions which refer to the entities and relationships it contains. The ease with which an NLI system acquires this knowledge is a measure of its *portability*.

There are two main approaches which make NLI systems portable between applications. The first approach is to modularize the system so that those modules containing domain specific information can be exchanged. This approach is emphasized by Datalog. Experts analyze the new domain and create the necessary modules. The second major approach is to provide interface software for a database administrator (DBA) or skilled user to provide the required domain specific information without undue difficulty. This is the approach taken by TEAM. In either case these systems require significant non-trivial human intervention resulting in loss of portability.

**System X**, a NLI to relational databases under development at Simon Fraser University has addressed portability in several ways Each major step in the translation from English to SQL is contained in a separate module. Each module is itself separated into submodules which contain domain independent or domain dependent knowledge, the latter being replaced for new applications. Although interface software is necessary to acquire knowledge, the philosophy of **System X** is that the amount of knowledge which humans must provide can be minimized by maximizing the amount of knowledge which the system can discover for itself by analyzing the database and by using information available from input queries.

The understanding of NL expressions referring to relationships among entities often requires extensive world knowledge. Such relationships among entities in the world are often represented in a relational database by *virtual* relations which must be derived by performing a sequence of "joins" on two or more of the base relations stored in the database. A NLI must be able to translate these NL expressions the corresponding sequences, or *join paths.* Current systems deal with this problem in one or both of two ways. Some systems, including TQA and Datalog, assume that the database is simple and only a few unambiguous relationships are represented in it. This assumption is highly unrealistic because databases which represent complex domains must represent many different relationships among many different sets of entities. Often there are different relationships among the same set of entities which must be represented. A second approach demands that a database expert provide the join path corresponding to every NL expression that refers to such a relationship and that can be expected to be encountered in a query. TEAM and TQA require that this daunting task be part of the customization process. System X contains **Pathfinder**, which uses a model of the application domain to automatically generate the vast majority of these join paths as they are required during semantic interpretation. The model is derived by System X directly from the database scheme with minimal assistance from someone familiar with the database.

Automatically disambiguating queries in languages which permit ambiguity with respect to the operations required to access desired data is known as the *query inference problem.* A subproblem of the query inference problem, known as the *Multiple Access Path Problem* (MAPP), is to decide which of the number of join paths that derive relations containing the attributes mentioned in an ambiguous query is the *correct* path. **System/U**, (Ullman 1982), and **Verdi**, (Wald and Sorenson, 1984) are two systems which have been built which solve the query inference problem for an ambiguous tuple calculus. The approach used by Pathfinder is similar to these two approaches but Pathfinder has several advantages. Unlike System/U, Pathfinder tries to distinguish the most likely path from a set of probable paths. Verdi also distinguishes most likely paths but uses a representation of the database scheme, known as an Entity-Relationship data model,

[1] For a more complete account of natural language interfaces, especially earlier systems, see Cercone and McCalla (1986). In that paper they discuss a number of additional natural language database systems including, for example, the systems of (Codd 1974, 1979), (Hendrix et al. 1978), (Kaplan 1984), (McCoy 1982), (Waltz 1978) and (Webber and Finin 1984).

which requires greater effort to create than does the representation used by Pathfinder.

## 2. System X: overall system design

System X consists, in part, of a set of modules which create a canonical query representation. This query representation is passed to a second set of modules which translate it into a (logical) form similar to that of TQA, (Petrick, 1984), and thence into SQL. The canonical form represents the join path implicit in the query, any predicates - such as "greaterthan" - which are to be applied against database values, and any operations - such as "average" - which are performed on values from the database, as well as quantifiers and their scope.

Modules which construct the canonical query representation include a Lexicon, a Parser and a Semantic Interpreter. Figure 1. is a simplified representation of the overall system. Each module is independent of the other modules, providing that replacements accept input and provide output of the appropriate sort.

The Lexicon consists of two dictionaries: syntactic and semantic. It is designed so that the addition of novel words is largely automatic and does not require human intervention. Serving as a front end to the syntactic dictionary is a morphological analyzer **Morphos**. The analyzer consists of two sets of rules: a set of morphological rules which define the set of inflectional endings and permit inferences about the grammatical category of strings, and a set of respelling rules which describe how strings are to be transformed after inflectional endings have been removed. This front end substantially reduces the amount of storage required for the syntactic dictionary since only the root form of a word need be stored.

Since the analyser does not require a root dictionary to check against, it is used to generate new lexical entries, querying the user when an inflectional ending is used by more than one grammatical category and automatically updating the lexicon when a new word is unambiguously analyzed.

The semantic dictionary is divided into two parts: a domain independent lexicon of predicates, operations, quantifiers, etc. which are transported from application to application, and an application dependent lexicon. The latter is largely generated automatically from the database schema. Hand customization is limited to a synonym dictionary.

Further reducing storage requirements is a preprocessor which examines queries for strings whose lexical representation - both syntactic and semantic - can be defined by their form alone. These include proper names, standard code names and numbers, etc. These strings are not represented in the lexicon, but are defined automatically when encountered in the input stream.

System X's Parser is a top-down breadth-first parser. Grammar rules are translated into LISP code which may be compiled. The compiled grammar is applied to the input query by an interpreter to produce a set of parse trees

It is possible for a grammar rule to invoke the semantic interpreter to choose among competing parses. This flexibility enables the parser to reject syntactically possible but semantically anomalous trees at the source of the anomaly, instead of maintaining competing parses and rejecting anomalous trees after parsing is completed. Rules of grammar may be assigned a rank relative to others which parse the same node, so that the order in which they are applied to a string is controlled. This ordering ensures that rules most likely to succeed are attempted first.

Associated with each rule of grammar is a set of semantic interpretation rules. When the semantic interpreter is presented with a parse tree, it retrieves from the tree the name of the grammar rule which produced the tree. This name provides access to the appropriate set of translation rules.

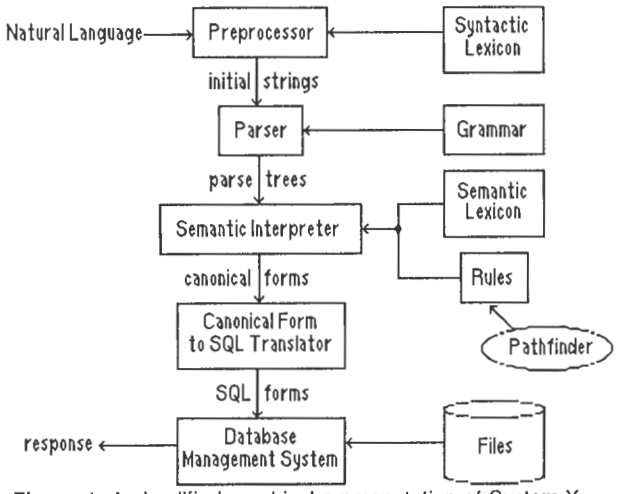



**Figure 1.** A simplified graphical representation of System X.

Semantic interpretation rules are divided into domain dependent and domain independent rules. The former are applied first to a parse tree to interpret any expressions peculiar to the application. They serve as filters to the latter which describe how any particular parse tree should be interpreted.

Underlying the semantic interpreter is the Pathfinder subsystem for generating join paths given a set of two or more column names. Since terms are treated as values in the database, the semantic representation of an expression of two or more terms or expressions is the join path between them. The task of the semantic interpreter is to pass the column names represented by the terms in an expression to this system and build up a representation using the join path which it returns. Treating semantic interpretation in this way results in a very flexible system which requires little customization.

The modules that translate the canonical form to logical form and thence to SQL, successfully convert canonical forms containing any combination of universal quantifiers, existential quantifiers and negation operators into complex SQL expressions that experienced SQL programmers would have great difficulty in composing. Routines in the translator perform some optimizations on the access path specified by the canonical form in order that the data may be more efficiently retrieved.

All the examples in the remainder of the paper are taken from an academic domain which is represented by the database scheme in Table 1.

| Relation | Attributes |
|---|---|
| STUDENT | student# name major minor sex status |
| FACULTY | faculty# name office sex status |
| DEPARTMENT | dept chairman faculty |
| COURSE | cname description dept |
| OFFERING | offer# cname semester units |
| CLASS | class# offer# sec faculty# text |
| ENROLL | class# student# final-grade |
| SCHEDULE | class# time.room |
| APPOINT | faculty# dept |

**Table 1** *The Academic Database* [2]

[2] Attributes in the key are underlined.

## 3. From English to parse trees

### 3.1 The Lexicon, Template and Morphos

Each of the **Lexicon's** two dictionaries, syntactic and semantic, is also divided into domain dependent and domain independent subfiles. System X does not use an *a priori* set of knowledge structures onto which the database structure must be mapped. Instead, the database structure serves as the basis of the semantics. The semantic representation of a term is its role in the database.Thus nouns and adjectives are defined as values in the database; verbs, when possible, are associated with relations. For example, the semantic representations of the constituents of *math course* are an adjective which is a value of either the *major* or the *dept* column and a noun which represents possible values of the *cname* column. The consequence of this is that large portions of the domain dependent semantic lexicon can be created without human interaction. Actual handcoding of the semantic lexicon is necessary for synonyms and some verbs. For example, verbs such as *pass* and *fail* refer to relationships among entities which cannot be discerned by analysis of the structure of the database

A major problem with any interface is lexicon size. In principle, the interface must be able to recognize the contents of the database; as a consequence, the lexicon may be at least as large as the database. Note that the proper names and identity numbers in the university database number in the tens of thousands. The lexicon is accessed through a pair of subsystems which function in part to reduce lexicon size. These subsystems are TEMPLATE and MORPHOS. The former provides a means to define terms which can be recognized by their physical shape. Such words include proper names, part numbers and report identity numbers. TEMPLATE provides a simple mechanism for describing the shape of these and associating this description with a means for creating the appropriate grammatical and semantic representations. TEMPLATE compiles the description into LISP as a pattern matcher which when successful creates grammatical and semantic representations for the input form.

TEMPLATE provides the flexibility an NLI system must have to easily handle the many small but difficult problems which particular applications present (Hafner and Godden, 1985). For example, the use of numbers to refer to hardware or university semesters cannot be handled by listing each number in the lexicon. However, since such expressions conform to a recognizable form, they may be assigned a (possibly ambiguous) definition by TEMPLATE.

MORPHOS is a rule-based morphological analyser which first scans a word for recognizable inflectional suffixes before searching the lexicon. Each morphological rule defines a possible suffix and states what grammatical information can be inferred from the presence of the suffix. After a suffix is removed, the remaining stem has a set of spelling rules applied and generates the correct spelling of the uninflected form. For example, after the inflectional suffix is removed from *cities* the resulting *citie* is respelled as *city*. Since MORPHOS generates lexical entries for regularly inflected forms and correctly generates the uninflected form, regularly inflected words need not be entered in the grammatical lexicon. In addition to thereby reducing lexicon size, MORPHOS is able to assist in generating the grammatical lexicon. If no lexicon is available or if the uninflected form of a word cannot be found in the lexicon, then if the analysis of MORPHOS is unambiguous, lexical entries for the input word and its uninflected form are generated.

### 3.2 Syntactic Analysis

When an input query is presented for syntactic analysis, the preporcessor first replaces each word in a query with its grammatical representation. The output of the preprocessor, an undifferentiated list of grammatical representations, is passed to the parsing system. The parsing system consists of three subsystems; a set of grammar rules, a compiler which translates these rules into LISP code and an interpreter which applies these rules to the input. The grammar is under continual revision; currently, it is well developed in most query types and in expansion of noun phrases and comparatives. It is unable to handle any but the uninteresting cases of conjunction and, though it can successfully parse sentence fragments, it cannot handle ellipsis as yet.

The grammar and interpreter are domain independent and may be easily transported to new applications. The only customization necessary in principle is the creation of a syntactic lexicon. However, if users of an application use idiosyncratic locutions to refer to entities in the domain, new rules are easily added.

The parser is permitted limited interaction with the semantic component. Among the actions which a grammar rule may take upon successfully parsing a node is to pass the resulting parse to the semantic interpreter. If the semantic interpreter cannot assign a meaning representation to the parse, it is rejected. If the interpreter returns a meaning representation, it is stored in the register TRANSLATION in the parse tree. The only information from the semantic interpreter which the parser uses is the fact that an interpretation is possible, it does not use the semantic structures in subsequent parsing as does a semantic grammar. Interaction of the syntactic and semantic components is used to resolve syntactically ambiguous structures. Since the parser operates breadth-first without backtracking, it must carry all possible parses throughout the parsing process and let the semantic interpreter select the appropriate parse. By providing a mechanism for interaction between the syntactic and semantic components, many syntactically ambiguous structures can be resolved at the source of the ambiguity and, thus, the number of possible parses is reduced.

## 4. Semantic Interpretation

The resulting parse tree from a query, which may have meaning representations attached to some of its subtrees, is passed to the semantic component. The organization of the semantic component mirrors that of its syntactic counterpart. It consists of a set of semantic rules, a compiler which translates these into LISP code and an interpreter which applies the rules to a parse tree. A semantic rule is associated with at least one syntactic rule and is designed to translate the parse tree which that syntactic rule creates. The rule-based implementation of the semantic component has important consequences for portability. Hafner and Godden(1985, p159) discuss examples of phrases which are within the range of Datalog's grammar and are paraphrases of queries which Datalog can understand but which it can not process. Their solution requires creating new semantic nodes and procedures for interpreting these phrases. The solution to this problem in System X requires no new nodes or procedures. Instead the semantic component is extended by adding new semantic rules. The addition of new rules is part of the general process of extending the linguistic coverage of the system.

Whereas many interfaces produce a single logical form, **System X** is designed to produce several representations each designed for a particular purpose; for example, query representation, representation of presuppositions, pronoun resolution, etc. At the moment, only the first representation has been implemented. The query representation language was inspired by TQA although it has it now bears little resemblance to its precursor.

When a parse tree is passed to the semantic interpreter, the meaning representations of each of its constituent nodes is retrieved. For each node, if it is a terminal node, its meaning representation is retrieved from the semantic lexicon; if it is nonterminal, the interpreter recursively descends the parse tree. For example, the parse tree assigned to the query *Has every math major taken math344?* is depicted in Figure 2. When this is presented to the semantic interpreter, the interpreter begins a left to right recursive descent down the tree. During the interpretation of the subject noun phrase, the interpreter arrives at the *N^* node. After it has retrieved the meaning representations of the constituents of this, it applies to them the set of semantic rules associated with the grammar rule which create the parse, in this case *1N^*. The relevant rules are those concerned with attaching an

adjective to a noun. Adjective attachment depends on the semantic content of the adjective; the semantic structure assigned to *math major* will differ from that assigned to *average grade,* which requires the operation AVERAGE.

Each semantic rule is a pattern matcher which specifies both the structure and the semantic content of the constituents. Most rules which create semantic structures pass the nodes in the appropriate order to the function CREATE-TREE. This function examines its arguments to determine what type of semantic structure to create. If no verb was passed as an argument, it creates a semantic structure which represents a database object. If a verb is passed, an *S* node is created with it as its verb. The phrase *math major* is passed without a verb, thus the semantic structure represents a database object, namely, the set of student numbers from the tuples in the *student* relation which have *math* as the value of the *major* column.

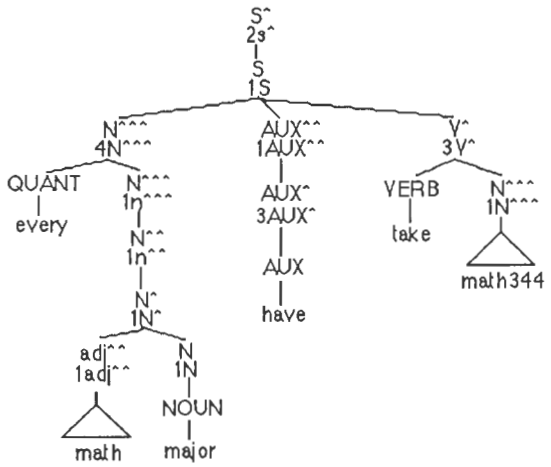



**Figure 2.** The parse tree for "Has every math major taken math344?".

During the interpretation of the query, the auxiliary phrase and the verb are examined for tense. During the customization process, if time is encoded in the database, the name of the column which represents time is placed in the register *TIME*. If this register is filled, the semantic interpreter will create a semantic structure for the period implicit in the query. The structure in Figure 4 represents the past tense of our example query.

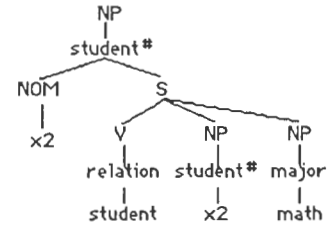



**Figure 3.** Semantic representation of *math major*

At the level of the *S* node, the semantic interpreter has retrieved semantic structures for the subject noun phrase, the verb, and the noun phrases and prepositional phrases in the verb phrase. It is possible that at this level some of the *NP* structures are ambiguous. For example, in the query *From whom did Smith take math301?* the words *whom* and *Smith* are defined as referring to either students or faculty members. In this case Pathfinder cannot be used to disambiguate these structures. In these cases, a small case grammar which describes the relationships in the application domain is used to disambiguate ambiguous structures.

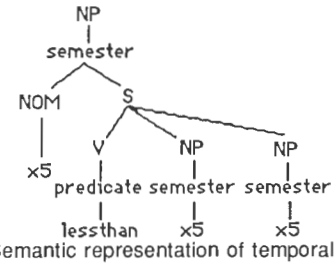



**Figure 4.** Semantic representation of temporal reference.

The semantic rules associated with *S^* node call functions responsible for retrieving quantified objects and representing their scope. By convention, all universal quantifiers are explicitly represented, but only those existential quantifiers which have within their scope a universal quantifier are represented.

All interpretation rules which build semantic structures relating different entities use **Pathfinder** to discover how these entities are related in the database. The path returned by **Pathfinder** is used to create the appropriate semantic structure.

## 5. Pathfinder

Relationships between different entities are represented by non-terminal nodes in the parse tree. The interpretation of these nodes requires establishing the relationship between attributes corresponding to the heads of the branches of the subtree rooted at that node. In other words, the access path to the database relation corresponding to the subtree must be found. Consider the *S* node in Figure 2. The interpretation of this node requires the derivation of the relationship between *math344* (which corresponds to the attribute CNAME in COURSE and OFFERING) and *majors* (which corresponds to the attribute STUDENT# in either STUDENT or ENROLL).

Since a given set of attributes may belong to different database relations derived by different access paths, interpreting this type of non-terminal node requires an ability to solve the MAPP. Pathfinder is used by the semantic translation rules responsible for interpreting this type of node to generate a "best guess" of the correct relation. The correct relation is assumed to be the one which contains the most cohesive relationship between the given set of attributes.

We begin our discussion of Pathfinder by examining the semantics of attribute relationships in relational databases. Next we make explicit certain assumptions about the design of the relational databases to which System X is to interface. Finally we describe the data structure and the algorithm Pathfinder uses to derive the database objects which correspond to the relationships referred to in NL queries.

### 5.1 Data Dependencies

A database scheme represents relationships between entities in the world because database designers represent dependencies that exist between entities by dependencies between the database attributes which denote those entities. It is these data dependencies that give the scheme its structure.

The basic type of data dependency from which most attribute relationships are constructed is the functional dependency (FD). We say $X \rightarrow Y$, read attribute *Y* is *functionally dependent* on an attribute *X*, (or *X functionally determines Y*), if for each value of *X* there is exactly one *Y*-value. In the academic world represented by our example database a student may only have one major and a course may be offered by only one department. Thus the database design contains the functional dependencies STUDENT# $\rightarrow$ MAJOR, and CNAME $\rightarrow$ DEPT.

Functional dependency is a transitive relation. The attribute CLASS# which denotes classes functionally determines OFFER# which denotes offerings. Since OFFER# functionally determines CNAME which denotes courses, CLASS# also (transitively) functionally determines CNAME.

Database theory identifies one kind of data dependency, other than the functional dependency, which corresponds to a "real world" dependency. This data dependency is known as the multivalued dependency (MVD). Roughly speaking, there is a *multivalued dependency* in relation $r$ of attributes $Y$ on attributes $X$ if there is a set of zero or more values of $Y$ associated with a given value of $X$, and the set of $Y$-values is not determined in any way by attributes in $r$ other than those in $X$ or $Y$. A functional dependency is a special case of a multivalued dependency.

For example, suppose our academic database scheme contained a base relation composed of the union of the schemes for ENROLL and SCHEDULE instead of those two schemes. This new scheme would be

ENROLL/SCHEDULE = student# class# time room final-grade.

In a relation on ENROLL/SCHEDULE there would be a MVD of {TIME ROOM} on CLASS# and a MVD of {STUDENT# FINAL-GRADE} on CLASS#. These MVDs reflect the fact that there is a given set of times and rooms to which all students enrolled in a class are scheduled, and that there is a given set of students expected to attend all schedulings of each class. It is this dependency between schedulings and enrollments that is represented in the database by the MVDs mentioned above.

A relation containing MVDs will contain redundancies. In ENROLL/SCHEDULE each class' schedule will be repeated for every student in the class. Adherence to fourth normal form (4NF), a commonly accepted standard for database design, requires that such relations be decomposed in order to eliminate the redundancies. The two relations, ENROLL and SCHEDULE, are the result of the decomposition of ENROLL/SCHEDULE into 4NF.

### 5.2 Cohesiveness of Attribute Relationships

The relationship among a given set of attributes is more or less cohesive depending on the type of joins which are required to associate the members of the set into a single relation. According to relational database theory joins may be characterized as either *lossless* or *lossy* (Ullman, 1982). Informally, a lossless join is one in which there is a MVD of each the attributes in the resulting relation on the attribute set on which the join takes place. A lossy join is a join which is not lossless. A relation whose derivation includes only lossless joins is more cohesive than a relation whose derivation includes a lossy join. For example, consider two relations: the first formed by the lossless joins of the OFFERING relation to CLASS and COURSE on the attributes OFFER# and CNAME respectively, and the second formed by the lossy join of the CLASS relation to the APPOINT relation on the attribute FACULTY#. Both derived relations contain the attribute set {CLASS# DEPT}. The two occurrences of the set represent two different relationships that exist between classes and departments. The relationship represented in the relation created by the lossless joins is the relationship that exists between a class and the department which offers the class. The relationship represented in the relation created by the lossy join is the relationship that exists between a class and the department which employs the faculty member who teaches the class. Clearly, this latter relationship is weaker than the former. Even though the second relation was created by fewer joins than the first, the fact that it was derived via a lossy join led to a less cohesive object. The relationship between attributes that are brought together by a lossy join is always weaker than a relationship between attributes brought together by any number of lossless join.

We measure the cohesiveness of attribute relationships within relations derived via lossless joins alone by weighing the number and type of dependencies that make up those relationships. We assign weights to each dependency type in inverse proportion to the strength of the dependency. MVDs are the weakest; therefore they are assigned the greatest weight, which is 2 units. FDs are stronger; we assign to them a weight of 1 unit, except in the special case of the FD of an attribute on itself. In this case the weight assigned is 0.1. (For a justification of this particular weighing scheme, see (Hall et al, 1988)).

The weight of a particular relationship among a given set of attributes is the sum of the weights of the dependencies that make up that relationship. The more numerous and the weaker the dependencies that make up an attribute relationship, the less is its cohesiveness. Thus the cohesiveness of an attribute relationship is inversely related to its weight.

### 5.3 Assumptions

Before describing the method Pathfinder uses to calculate access paths, we outline our assumptions about the design of the target database in which the paths are sought. Firstly, we assume that the database is in 4NF.[3] The only dependencies that exist in 4NF databases are FDs between keys and attributes within a single relation.

We also assume that if two attributes have the same name and one is part of a key of a relation then both denote the same entity. We call this the Unique Key Name Assumption (UKNA). The UKNA ensures that transitive FDs are preserved. That is, $X \rightarrow Z$ when $X \rightarrow Y$ and $Y \rightarrow Z$ only if $Y$ denotes the same entity in both the latter expressions.

Finally, we assume that an entity denoted by a composite primary key $K_k$ of a relation $R_k$ is not referred to in another relation $R_l$. This restriction can be enforced by adding, when required, a single attribute surrogate key to $R_k$ and using the surrogate in $R_l$ to refer to the entity denoted by $K_k$. We call this the Surrogate Key Name Assumption (SKNA). As a result the SKNA different relationships between the same entities may be represented by the same attribute set in different relations without concern that System X will mistakenly conclude that an occurrence which is a key of one relation is functionally dependent on the key of another relation which contains a second occurrence.

### 5.4 Representing Attribute Relationships

In order to derive access paths, Pathfinder uses a representation of the database scheme which we call the *join graph*.

The nodes in a join graph represent the base relations ($R_1$, ..., $R_n$) in a database. Each node is labelled with the name of its corresponding relation and its key. There is a directed edge ($R_i$, $R_j$) from node $R_i$ to node $R_j$ iff there is an attribute $A$ in $R_i$ which denotes the entity denoted by the key $K_j$ of $R_j$. For example, in our academic database the key of the relation STUDENT, STUDENT#, denotes the entity type "student". The relation ENROLL contains an attribute, (also named STUDENT#), which represents the same entity type. Thus there is an directed edge in the join graph from the node ENROLL to the node STUDENT.

In most cases, as it was in the example above, the two attributes $A$ and $K_j$ will have the same name. However it may be that a database designer gives different names to attributes which denote the same entity. For example, in the academic database scheme the attribute in the relation STUDENT which denotes the department constituting a student's major is named "MAJOR" and those same departments are denoted in the relation DEPARTMENT by the (key) attribute named "DEPT". Note that in the join graph there is an edge from node STUDENT to node DEPARTMENT.

---

[3] Pathfinder may be easily extended to handle 3NF databases. However we have not yet implemented the extension.

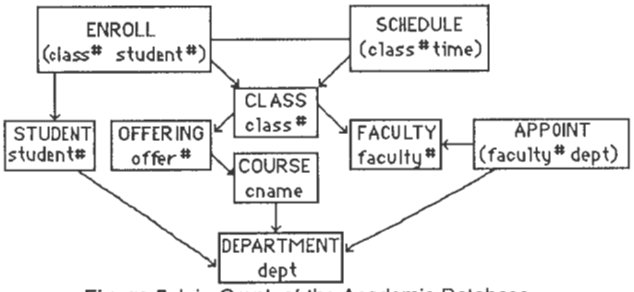



**Figure 5** Join Graph of the Academic Database

There is an undirected edge $\{R_v, R_w\}$ between $R_v$ and $R_w$ if a dependency of the type that yields MVDs exists between the entities represented by $R_v$ and $R_w$ . The relation pair (ENROLL SCHEDULE) meets this requirement and therefore there is an undirected edge in the join graph between the nodes labelled with the names of these two relations.

When a set of attributes is submitted to Pathfinder, the system creates nodes to represent the attributes and adds those nodes to the join graph. After these *target attribute* nodes are added, we call the resulting graph the *augmented join graph*. There is a directed edge from (relation) node $R_i$ to (attribute) node $A_j$ in the augmented join graph if attribute $A_j$ is contained in relation $R_i$ . Thus, in our example, if {FACULTY# CHAIRMAN} was a target attribute set, two attribute nodes, FACULTY# and CHAIRMAN would be added to the graph along with edges (FACULTY,FACULTY#), (APPOINT,FACULTY#), (CLASS,FACULTY#), and (DEPARTMENT,CHAIRMAN). Figure 6 shows our example join graph augmented with nodes representing the attributes FACULTY# and CHAIRMAN.

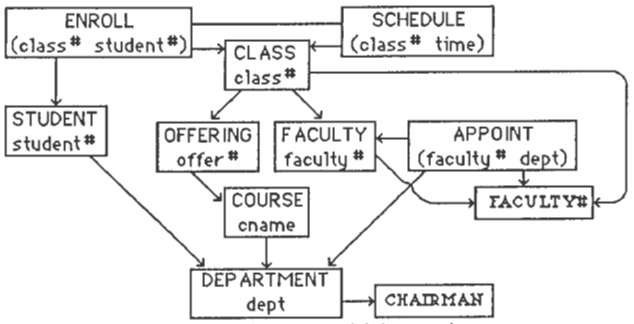



**Figure 6.** Augmented join graph.

Join graph edges represent lossless joins between the relations corresponding to the nodes adjacent to the edge. In the case of a directed edge the join takes place on the key of the relation represented by the node at the head of the edge. In the case of an undirected edge the join takes place on the intersection of the keys of the two relations represented by the adjacent nodes. Every lossless join between two base relations in the database is represented by exactly one edge in the join graph. At the time that the join graph is created, a table called *join-edges* is also created. This table contains the join specifications for each of the lossless joins corresponding to the edges in the join graph.

Lossy joins are represented in augmented join graphs by a pair of directed edges which have a common node at the head of each edge. Such an edge pair represents the join of the relations represented by the nodes at the tail of each of the two edges on the attribute represented by the common node. (If the common node represents a relation, the join takes place on the key of the relation.)

The above mapping between augmented join graph edges and database joins results in the following mapping between subgraphs of the augmented join graph and database relations. Trees in an augmented join graph represent relations derived by 0 or more lossless joins. Subgraphs of the augmented join graph which are not trees represent relations whose derivation includes at least one lossy join.

In addition to representing joins between relations, the edges of an augmented join graph represent dependencies between attributes of the database. A directed edge represents the FD of the attribute represented by the node at the head of the edge on the key of the relation represented by the node at the tail of the edge. (If the node at the head represents a relation, the determined attribute in the corresponding FD is the key of that relation.) An undirected edge represents the MVD between the sets of attributes which comprise the keys of the relations represented by the nodes adjacent to the edge. Weights equal to the weights of the corresponding dependencies are assigned to the edges of an augmented join graph. Thus the cohesiveness of the relation corresponding to a subtree of an augmented join graph can be measured by summing the weights of the edges of the subtree.

### 5.5 Finding the Most Cohesive Relation

Recall that Pathfinder is used to derive access paths for relations corresponding to non-terminal nodes of parse trees. The correct relation is assumed to be the one containing the most cohesive relationship between the attributes represented by the heads of the branches of the subtree rooted at the node being interpreted. Pathfinder restricts its search to relations derived exclusively by lossless joins; relations derived by lossy joins are so weak that they are not even considered. For example, recall the relationship between CLASS# and DEPT contained in the relation formed by the lossy join of the relation APPOINT to the relation TEACH. This attribute relationship represents the real world relationship between a department and a class which occurs when the class is taught by a faculty member in the department. The simplest English phrase referring to this relationship must make reference to the faculty member or members in question. For example, "classes of math professors" refers to those classes taught by professors appointed to the math department. Removing the reference to the faculty member yields a phrase such as "math classes". Phrases mentioning only classes and departments refer to the relationship that exists between classes and the departments which offer those classes. That is, such phrases correspond to the relationship between the attributes DEPT and CLASS# which is contained in the lossless join of the relations CLASS, OFFERING, and COURSE. Figure 7 is the parse tree for the phrase "classes of math professors".

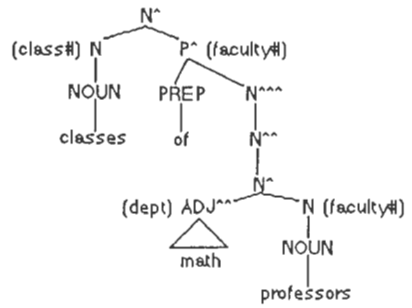



**Figure 7** Parse of a phrase referring to a lossy join.

The interpreter will resolve the relationship between FACULTY# and DEPT at the N^ node in the right branch of the tree. This is the relationship denoted by the occurrence of {FACULTY# DEPT} in the base relation APPOINT. The attribute FACULTY# will then be identified as the head of the P^ node. Finally the interpreter will

resolve the relationship between CLASS# and FACULTY# at the N^ node at the root of the tree. This is the relationship denoted by the occurrence of {FACULTY# CLASS#} in the relation CLASS. Note that, although the original phrase refers to a complex relationship in the database which involves a lossy join, neither of the composite relationships that the interpreter will resolve involve such a join.

Since the relation corresponding to a non-terminal parse tree node must be derivable without a single lossy join, Pathfinder restricts its search to subtrees of the augmented join graph. Finding the most cohesive attribute relationship containing a given set of target attributes is accomplished by finding the tree in the augmented join graph which contains the nodes corresponding to the target attributes and has an edge weight less than any other tree containing those nodes.

This is a version of the Minimum Cost Steiner Tree problem for graphs, which may be defined as follows. Let $G = (V, E)$ be a connected graph with vertices $V$ and edges $E$. Let $TV$ be a set of target nodes where $V \supseteq TV$. A *Steiner tree* is a graph $SG = (SV, SE)$ such that $SG$ is a subtree of $G$ and $SV \supseteq TV$. Let COST:$E \rightarrow N$ be a cost function from the edges $E$ to the positive integers $N$. The *minimum cost Steiner tree* (MCST) minimizes the sum of COST($e$) for edges $e$ in $SE$. A recent survey of the Steiner problem in graphs may be found in (Duin and Volgenant, 1987).

Although the corresponding decision problem was proved NP-complete in (Karp, 1972), there is an efficient linear time algorithm for solving the MDST problem in graphs when $|TV| = k$, for any small, fixed $k$ (Levin, 1971). This algorithm suits our problem since we can realistically assume that the number of attributes at the head of the branches rooted at a non-terminal node in a parse tree will be less than six. We call our version of the algorithm STEINER. The algorithm STEINER returns all MCSTs that contain its input attribute set.

A MCST returned by STEINER is a complete specification of the access path to the relation containing the relationship between the target attributes that the MCST represents. Each edge in the MCST adjacent to two relation nodes represents a join between the corresponding base relations on the attributes specified in the *join-edges* table. The remaining edges adjacent to the leaves represent the projection of the relation created by the joins onto the target attributes.

Figure 6 is the join graph for our example database augmented with nodes to represent the target attribute set {CHAIRMAN FACULTY#}. This is the set that would be submitted to Pathfinder when an interpretation is being sought for the phrase "Prof. Jones' chairman". Of the two Steiner trees for this target set depicted in Figure 8, clearly the minimum tree is the one rooted at the APPOINT node. A glance at Figure 6 will confirm that this is the minimum cost Steiner tree connecting these two attributes in the augmented join graph. This tree represents the relation formed by the join of the relations APPOINT and DEPARTMENT on DEPT, the key for DEPARTMENT. The relationship it represents is the relationship between a faculty member and the chairman of the department to which he is appointed. The other tree in Figure 8 represents the relationship between a faculty member and the chairman of the department for which the faculty member teaches courses. The minimum tree corresponds to our intuition about the relationship that is actually being referred to in the phrase "Prof. Jones' chairman".

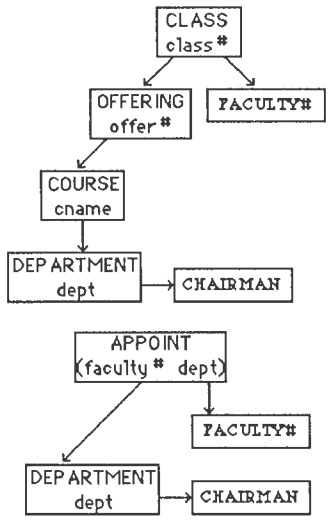



**Figure 8** Subtrees containing {FACULTY# CHAIRMAN}

### 5.6 Portability Considerations

System X uses the Pathfinder subsystem to derive the access paths which correspond to NL constructions in its input queries. The semantic representation that Pathfinder uses is the join graph. Almost all the information required to create a join graph is available in the database scheme. The system must be informed of attributes which are named differently but denote the same entity. In addition, the system must be informed of the existence of MVDs. The effort required for this intervention is significantly less than the effort required for a database expert to derive all the join paths that may be required and insert them into a table or some other data structure. Thus, the Pathfinder subsystem reduces the amount of time and effort required to bring up **System X** in a new application domain and makes it easier to transport between domains.

## 6. From Canonical Form to SQL

Since the canonical form represents the query in terms of database entities the translation from this form to SQL is relatively straightforward.

The canonical form is initially translated into a *logical form*. The logical form is a set domain relational calculus. Like the canonical form its inspiration comes from the TQA logical form to which it retains many similarities. Since it is a relational calculus it can be translated into any relational query language of equal or greater expressive power, thus enabling System X to be transported to a wide range of relational database management systems.

Figure 9 contains the logical form and SQL for the query of Figure 2.

**Logical Form:**

```
(INT*
   (NOT*
      (SETX X2
         (AND (RELATION STUDENT (STUDENT# MAJOR)
                                (X2 'MATH)
                                (= =))
         (NOT*
            (SETX X5
               (SETX X7
                  (SETX X6
                     (AND (RELATION ENROLL
                                    (STUDENT# CLASS#)
                                    (X2 X6)
                                    (= =))
                          (RELATION CLASS
                                    (OFFER# CLASS#)
                                    (X7 X5)
                                    (= =))
                          (RELATION OFFERING
                                    (SEMESTER CNAME
                                     OFFER#)
                                    (X5 'MATH344 X7)
                                    (= = =))
                          (LESSTHAN X5 '873)))))))))))
```

**SQL:**

```
CTR
SELECT UNIQUE A.STUDENT#
       FROM STUDENT A
       WHERE A.MAJOR = 'MATH'
       AND NOT EXISTS
          (SELECT UNIQUE C.SEMESTER
             FROM CLASS B, OFFERING C, ENROLL D
             WHERE C.CNAME = 'MATH344'
             AND B.OFFER# = C.OFFER#
             AND A.STUDENT# = D.STUDENT#
             AND B.CLASS# = D.CLASS#
             AND C.SEMESTER < 873)
```

**Figure 9.** Example Logical Form.and SQL

The relational calculus that constitutes the logical form has been extended slightly to allow some flexibility in the response of System X to different types of queries. A verification clause, defined as

<verification> ::= (INT* [NOT*] <retrieval>),

specifies a query that expects a "yes" or "no" answer, depending on whether the <retrieval> does or does not succeed in retrieving records from the database. The logical form in our example indicates that if the <retrieval> (SETX X2 ...) succeeds in finding a record the response should be "no", otherwise it should be "yes". Since this facility does not form part of SQL a similar extension has been made by System X to that language. In the SQL version of the query the keyword CTR (that is, "counter") performs this same function of indicating how the result of the retrieval is to be interpreted.

Since SQL does not contain a universal quantifier, universal quantification must be expressed by means of double negation in SQL queries. The transformation of the universal quantifier into double negation occurs during the translation of canonical form into logical form. The logical form in Figure 9 contains the result of such a transformation.

## 7. Concluding Remarks

**System X** makes some interesting, although occasionally small, improvements in the state-of-the-art for natural language interfaces to databases. Our achievements centre around four aspects indigenous to natural language interfaces specifically and natural language understanding in general.

Since templates permit definition by recognition rather than storage, we claim lexicon storage reduction as an accomplishment. Reducing the amount of information that must be stored in a lexicon makes the system more likely to be able to be transported to computers that have limitations on storage and to applications which have large storage requirements.

MORPHOS also contributes to storage reduction as it obviates the need to store all forms of regularly inflected words. In addition, it provides a unique customization tool operating in the background to create a grammatical lexicon, querying the user only when necessary.

**System X** is flexible, providing different ways to solve many different syntactic and semantic problems. For example, the use of numbers as common nouns is solved using TEMPLATE but we could of used syntactic and semantic rules instead.

**Pathfinder** automatically generates most of the access paths required by **System X** using a representation of the database world which is constructed with a minimum of human intervention. Thus, **Pathfinder** reduces both customization and storage requirements.

## Acknowledgements

We would like to thank the Natural Sciences and Engineering Research Council of Canada for the grants which have made this research possible. In addition, we would like to thank Dr. Gordon McCalla who contributed significantly to the ideas we presented in this paper.

## References

[1] Ballard, Bruce W., Lusth, John C. and Tinkham, Nancy L. LDC-1: A Transportable, Knowledge-Based Natural Language Processor for Office Environments. *ACM Transactions on Office Information Systems* 3(2):1-25, 1985.

[2] Cercone, N. and McCalla, G. Accessing Knowledge through Natural Language. Invited Chapter for M. Yovits 25th Anniversary Issue *Advances in Computers* series, Academic Press, pages 1-99, 1986.

[3] Codd, E.F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 13(6):377-387, 1970.

[4] Codd, E.F. Seven Steps to RENDEZVOUS with the Casual User. *Data Base Management.* North-Holland Publishing Co., Amsterdam, 1974, pages 179-200.

[5] Codd, E.F. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems* 4(4):397-434, 1979.

[6] Duin, C.W. and A. Volgenant. Some Generalizations of the Steiner Problem in Graphs. *Networks* 17(3):353-364, 1987.

[7] Hafner, Carole D. and Godden, Kurt. Portablility of Syntax and Semantics in Datalog. *ACM Transactions on Office Information Systems* 3(2):141-164, 1985.

[8] Hall, Gary. *Querying Cyclic Databases in Natural Language.* Master's thesis, , September, 1986.

[9] Hall, Gary, WoShun Luk and Nick Cercone. Disambiguating Queries Using Dependency Graphs. *Technical Report* 87-7, LCCR, SFU, Burnaby, B.C., 1987.

[10] Hall, Gary, McFetridge, P., Cercone, N., Luk, W.S. Automatic Access Path Generation in System X. Submitted to 5th Annual Conference on Data Engineering.

[11] Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D. and Slocum, J. Developing a Natural Language Interface to Complex Data. *ACM-TODS* 3(2):105-147, 1978.

[12] Johnson, D.E. Design of a Portable Natural Language Interface Grammar. Technical Report 10767, IBM Thomas J. Watson Research Laboratory, Yorkton Heights, N.Y. , 1984.

[13] Kao, M., Cercone, N. and Luk, W.S. Turning Null Responses into Quality Responses. *IEEE Transactions on Software Engineering.* 1987, to appear.

[14] Kaplan, S.J. Designing a Portable Natural Language Database Query System. *ACM-TODS* 9(1):1-19, 1984.
[15] Karp, R.M. Reducibility among combinatorial problems. *Complexity of Computer Computations*. Plenum Press, New York, 1972, pages 85-103.
[16] Levin, A. Ju. Algorithm for the Shortest Connection of a Group of Graph Vertices. Soviet Math. *Dokl.* 12(5):1477-1481, 1971.
[17] Martin, P., Appelt, D., Grosz, B. and Periera, F. TEAM: An Experimental Transportable Natural Language Interface. *Database Engineering* 8(3):10-22, 1985.
[18] McCoy, K.F. The ENHANCE System: Augmenting a Knowledge Base for Natural Language Generation (1). Technical Report MS-CIS-82-52, Dept. of Cmpt. & Info Science, Univ. of Penn., Philadelphia, PA., 1982.
[19] Petrick. S.R. Semantic Interpretation in the Request System. Technical Report RC 4457, IBM Thomas J. Watson Research Laboratory, Yorkton Heights, N.Y., 1973.
[20] Petrick. S.R. Natural Language Database Query Systems. Technical Report RC 10508, IBM Thomas J. Watson Research Laboratory, Yorkton Heights, N.Y., 1984.
[21] Ullman, Jeffrey D. *Principles of Database Systems*, 2nd ed. Computer Science Press, Rockville, Maryland, 1982.
[22] Ullman, Jeffrey D. The U. R. Strikes Back. *In Proceedings of the ACM Symposium on Principles of Database Systems*, pages 10-22. Los Angeles, CA., 1982.
[23] Ullman, Jeffrey D. Implementation of Logical Query Languages for Databases. *ACM-TODS* 10(3):289-321, 1985.
[24] Wald, J.A. and P.G. Sorenson. Resolving the Query Inference Problem Using Steiner Trees. *ACM-TODS* 9(3):348-368, 1984.
[25] Waltz, D. An English Language Question Answering System for a Large Relational Database. *CACM* 21(7):526-539, 1978.
[26] Webber, B.L. and Finin, T. In Response: Next Steps. *Natural Language Interaction in Artificial Intelligence Applications for Business*. Ablax Pub. Co., Norwood, N.J., 1984.

# Time Revisited

*Stephanie A. Miller*

*Lenhart K. Schubert*

Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

*ABSTRACT*

Temporal reasoning is essential for many AI applications. To date, most research has concentrated on temporal inference in isolation without considering the role it can play in a more general reasoning environment. This paper takes an efficient temporal reasoner and extends its inferential capabilities to handle both strict and nonstrict relations. The resulting temporal specialist is incorporated into a system intended for low level reasoning in natural language understanding. The specialist assists the resolution-based theorem prover in function evaluation, literal evaluation, and generalized resolving and factoring. The combined system can do some proofs in just a few steps that would normally require many. An example from the fully operational hybrid system is included.

Keywords: temporal reasoning, special inference methods, theory resolution, knowledge representation

## 1. Introduction

Given certain explicit relationships among a set of events or episodes, we would like to be able to infer additional relationships implicit in the ones given. For example, if the events are part of a narrative, they will frequently form sequences in which adjacent events are known to follow one another; for such a sequence, we would like to be able to infer, without much effort, that events earlier in the sequence precede later ones, regardless of the number of intervening events. In addition, if we are given information about the durations or absolute times of some of the events, we would like to be able to infer quantitative consequences of this information, such as minimum and maximum elapsed time between given events. These sorts of inferences are essential in several areas of AI including story understanding, causal reasoning, and planning [All84].

Since temporal orderings are transitive and durations cumulative, such inferences in a typical general theorem prover can be computationally expensive. To compensate for this, researchers have tried to develop special representations and efficient methods for temporal inference. However, much of this work concentrates on temporal inference in isolation, rather than on using such a mechanism in a more general environment.

In this paper, we start with an efficient temporal specialist (based on Taugher and Schubert's model [SPT87, Tau83]) that does temporal inference in isolation. This specialist is unusual in the way it exploits chains of events such as are commonly found in narratives (or plans), so as to achieve constant-time determination of time order for many pairs of events. It does so without requiring all events to lie on chains, and without computing transitive closure. To further increase the effectiveness of the specialist, its capabilities are extended here to make it more complete and flexible; the major enhancement being the ability to handle both strict and non-strict time ordering. Non-strict ordering is often appropriate for the end of one event in relation to the beginning of the next in a narrative (where there may or may not be a delay between them), while strict ordering is often appropriate for the beginning of an event in relation to its end (when the event is of a type that cannot transpire instantaneously). This, as well as consistency and expressibility considerations, required considerable expansion of the set of temporal predicates and argument patterns handled by the specialist.

We then incorporate the resulting specialist into a general inference system with a resolution-based core (based on de Haan's theorem prover [HaS86]), where it assists the theorem prover with function evaluation, literal evaluation, and generalized resolution and factoring. The temporal specialist bypasses the normal proof procedure for the operations it handles, and can cut out numerous proof steps that would otherwise be required. Temporal literal evaluation uses the specialist's timegraph representation to simplify assertions, and resolvents generated by the theorem prover. Function evaluation simplifies a term by evaluating it (for example, *(start-of e1)* can be simplified to a constant, *e1start*). Generalized resolving and factoring make use of Stickel's partial theory resolution [Sti83] to quickly determine incompatibility or subordination of one literal by another. This allows resolution and factoring to be done where they usually cannot. For example, we can factor *[~[x during a] or [x during b]]* to *[a during b]*[1], even

[1] If we have *[a during b]* represented in the timegraph.

though the literals have different signs. Similarly, if we know *[a before-1*[2] *b]*, we can resolve *[x before a]* with *[x after b]* to the null clause, although the predicates are not identical, and the signs are the same.

The hybrid system consisting of the resolution-based theorem prover and the temporal specialist has been implemented in Lucid Common Lisp and runs on a Sun 3/75. An example is included from the operation of the system, which is called **ECoNet** (since it does the low level inferencing necessary for ECoSystem [HaS86]). This system is to the best of our knowledge the most powerful combination of a general deductive mechanism with a temporal specialist built to date.

## 2. The Temporal Specialist

Most current research in temporal inference uses two basic types of representations - *time intervals* and *time points*. *Time intervals* represent a given time period of finite length. There are numerous simple relations (from [All83, ViK86]) that can be defined between two intervals (*equal, before* (including *meets*), *after* (including *met-by*), *during* (including *starts* and *ends*), *contains* (including *started-by* and *ended-by*), *overlaps*, and *overlapped-by*).

The other major representation used is *time points*. These are abstract "instants" of time and are assumed to be non-decomposable. Pairs of such time points can be used to represent intervals. Only a few relations can exist between time points (<, <=, >, >=, =), and logical combinations of these can be used to define interval relations.

There is a trade-off between expressibility and efficiency in the use of the two representations. For instance, if we restrict the relationships that may be specificed for time intervals to *disjunctions* of those listed above (as Allen did), and similarly restrict relationships that may be specified for two time points to disjunctions of equalities and inequalities, then the *interval* representation is more expressive and flexible, while the *time point* one is more efficient. In particular, closure on Allen's interval based approach [All83] is shown to be NP-hard by Kautz and Vilain [ViK86], while closure may be done in $O(n^3)$ time in the time point representation, where $n$ is the number of points. If arbitrary conjunctive/disjunctive constraints on sets of time points are allowed, then of course the time point representation is as expressive as the interval representation, but efficiency is then lost. In general, the function of a specialist should be to handle as large a class of relationships as possible without sacrificing efficiency. Any inferences that fall outside the capabilities of the specialist can still be handled by the general deductive mechanism, albeit much less efficiently.

### 2.1. Taugher and Schubert's Representation

Taugher and Schubert's representation [SPT87, Tau83] is based on time points. Besides being able to determine relations between time points quickly (often in constant time), it can represent and reason with durations and absolute times. The representation uses a partial order graph, called a timegraph, whose nodes represent time points. Directed links between points indicate the given relation between the two points (< or <= depending on interpretation). The timegraph is partitioned into *chains*, which are defined as sets of points that are all linearly ordered with respect to each other, with possible transitive arcs[3]. Links between points in the same chain are *in-chain* links; between points in different chains, *cross-chain* links. Each point has a *pseudo-time* (a number) associated with it, which is arbitrary except that it respects the ordering relationship between it and other points on the same chain. Chain and psuedo-time information are calculated when the point is first entered into the timegraph, and stored directly with the point. Determining the relationship between any two points in the same chain may be done in constant time simply by comparing their pseudo-times, rather than following the in-chain links.

To determine the relationship between points in different chains, a search is required, but only the cross-chain links need to be examined explicitly. A *metagraph* keeps track of the cross-chain links effectively by maintaining a *metanode* for each chain, and using the cross-chain links for links between metanodes. As in-chain checking can be done in constant time, a graph search is dependent on the number of cross-chain links rather than the total number of time points. Creation of all supporting graph structures (including the metagraph) requires $O(n+e)$ space and $O(n+e)$ time, where $n$ is the number of time points, and $e$ is the number of relations between them. Note that a timegraph amounts to a set of atomic inequalities. Disjunctions such as *[[a <= b] or [c <= d]]* or inequalities such as *[a ~= b]* cannot be represented. This restriction, along with the exploitation of time chains, is what enables the specialist to operate efficiently.

Figure 1 shows an example time graph and metagraph. Following the cross-chain links, we can get that triangle1 is before square3, and square1 is before circle3, but no information about triangle2 and square3.

Furthermore, an absolute time (date) minimum and maximum are stored with each time point. These are six-tuples of the form *(year month day hour minute second)*, where each element may be numeric or symbolic (e.g. *(1987 04 a 12 b c)* represents some time at or after 12 a.m. and before 1 p.m. of some day in April, 1987). Absolute time maxima propagate back to points before the given point (in the chain or on other chains), and minima propagate forward. This ensures that each point has the best absolute time information possible. Details may be found in [Tau83]. Absolute time comparisons can

[2] *Before-1* means strictly before.

[3] Schubert et al. clarify this point in [SPT87], as Taugher's thesis was not clear on it. Transitive arcs are in-chain links which do not make up the "backbone" of the chain; that is, there are intervening nodes and links. Figure 1 shows a transitive arc between circle2 and circle4.

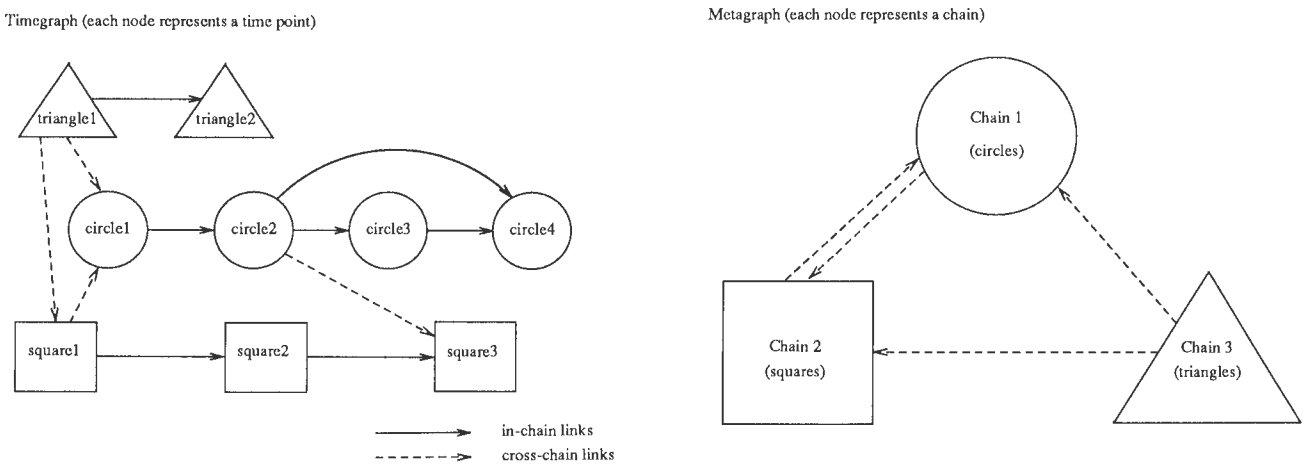



Figure 1. Example Timegraph and Metagraph

sometimes be used to get a relation in constant time between two points on different chains, avoiding a metagraph search. Duration minima and maxima (in seconds) are stored on the links between points. These may affect the absolute times around them, which are then propagated.

Insertion of a relation requires a constant amount of processing in most cases, except for propagation of absolute times. In the worst case, propagation may require going to every point in the graph, although this is highly unlikely. Occasionally a chain may have to be renumbered, which involves all the points in a single chain.

Intervals/events/episodes[4] are represented by their start and end time points. The intervals are kept in a separate table, which contains these end points. Interval relations are defined as combinations of time point relations between the start and end points of the intervals.

### 2.2. Enhancements to the Taugher and Schubert Model

Although this time model is efficient and does most of the temporal reasoning we need, it requires several enhancements. First, confusion surrounding whether a point (*a*) could be inserted between two others (*b* and *c*) and remain on the same chain[5], was resolved by allowing this to take place only when *c* has the smallest pseudo-time of any in-chain descendant of *b* (i.e is closest). Note that in this case the link from *b* to *c* will be a transitive arc (bypassing one intermediate node, namely *a*) of the resulting chain. Otherwise *a* must be placed on a new chain.

Second, the original representation can represent either strict relations (<, >, =), or non-strict relations (<=, >=, =), depending on whether links between points are interpreted as < or <=, but not both. Our goal here was to extend the model to handle both, without losing the efficiency of the original. For cross-chain relations, a flag indicating strictness on the link itself is enough, since cross-chain links are explicitly examined during a search. For points within the same chain however, examining each link for strictness would mean that we could no longer determine in-chain relationships in constant time. Attempts to handle this by adding another pseudo-time failed because they either could not represent all possible combinations, or were not constant time.

The successful method eventually found requires two extra numbers - a *maximum-pseudo* and a *minimum-pseudo*. The *minimum-pseudo* of a given point is the pseudo-time of the nearest predecessor on the chain that the point cannot be equal to; the *maximum-pseudo*, of the nearest successor on the chain that it cannot be equal to[6]. To determine the relation between two points, we first compare their pseudo-times to find out what order the points are in. Then we compare the pseudo-time of the first point with the range given by the minimum and maximum of the second point (or vice versa). If it is properly within that range (i.e. greater than the minimum and less than the maximum), the relation given by the pseudo-times alone is non-strict; otherwise it is strict. Propagation of minima and maxima maintains strictness of relations throughout the chain. This propagation is similar to propagation of absolute times (minima propagate forward, maxima back), although the comparison at each propagation step may be shorter (only one number to compare instead of a possible six).

To see how this works, consider the following example. Suppose $a <= b <= c <= d <= e$, and then we assert $a < d$:

[4] Interval, event, and episode are used interchangeably in this paper.

[5] A CMPUT 551 (Artificial Intelligence I) class at the University of Alberta discovered this in 1984 while reimplementing the algorithm as part of a project.

[6] This requires the addition of two additional pseudo-times to represents points before the beginning of the chain ($-\infty$) and beyond the end ($+\infty$). To make chain renumbering easier, pointers to the minimum and maximum points are kept, rather than their actual pseudo times.

| *Point* | *a* | *b* | *c* | *d* | *e* |
|---|---|---|---|---|---|
| *Pseudo-time*[8] | 1 | 1000 | 2000 | 3000 | 4000 |
| *Minimum-pseudo* | $-\infty$ | $-\infty$ | $-\infty$ | 1 | 1 |
| *Maximum-pseudo* | 3000 | $+\infty$ | $+\infty$ | $+\infty$ | $+\infty$ |

Note we have $a < d$, $a <= b$, $b <= c$, $a < e$, and so on - correctly. Now assert $c < e$:

| *Point* | *a* | *b* | *c* | *d* | *e* |
|---|---|---|---|---|---|
| *Pseudo-time* | 1 | 1000 | 2000 | 3000 | 4000 |
| *Minimum-pseudo* | $-\infty$ | $-\infty$ | $-\infty$ | 1 | 2000 |
| *Maximum-pseudo* | 3000 | 4000 | 4000 | $+\infty$ | $+\infty$ |

This method is flexible enough to handle any combination of relations, easy to understand and implement, and maintains the constant time in-chain checking.

To evaluate whether a given relation holds, or to determine the strongest relation constraining two points, we now need to consider that there may be numerous paths between the points, some of which may represent strict relations, and some non-strict. To determine the relation between two points, one must continue finding paths until a strict path is found or all such paths have been found. Within each chain, the strictness of the relation between points is easily calculated in constant time (the range comparison), and used in the path determination. It is only where there are several possible cross-chain links leaving a point that there is a possibility for paths of different strictness. In the worst case, this could require $O(m^2)$ time, where $m$ is the number of cross-chain links. Since we stop with the first strict path found, this maximum will rarely be reached[9].

With the additional capabilities for handling strict and nonstrict ordering, we now need some way of telling the temporal specialist which ordering to use. This was one reason, among others, for some changes to the set of temporal predicates recognized by the specialist. In Taugher's thesis, the following predicates were used for entry of intervals/events:

| | | |
|---|---|---|
| *a before b [c]* | => | *a end <= b start*<br>(both within time frame *c* if specified) |
| *a after b [c]* | => | *a start >= b end*<br>(both within time frame *c* if specified) |
| *a equal b* | => | *a end = b start* |
| *a during b [c]* | => | if *c* specified, *a between b and c*;<br>otherwise *a within b* |

During evaluation, some additional relations could be determined: *overlaps, overlapped-by, contains*, as well as some weaker ones: *starts-before, ends-before, starts-after, ends-after, starts-equal*, and *ends-equal*.

The predicates used in the temporal specialist implemented in this paper are essentially the same ones that Taugher used, with some exceptions. The weaker predicates have not been implemented. *During* is used only in the sense of "within", enabling the option of a timeframe argument. *Between* was introduced to handle the case where one argument is to be inserted "during" two others (see above). Except for *between*, all predicates now have the optional third argument representing a time frame (instead of just *before* and *after*). Usage of this argument leads to fewer chains being built in the timegraph, resulting in a more nearly optimal timegraph. All the predicates may be used for both entry (assertion) and evaluation for consistency.

The original predicates are now considered "stems" and may have one or two strictness values appended (*stem[-strict1[-strict2]]*). Strictness values are:

| Strictness Value | Meaning |
|---|---|
| *-1* | strict (< or >) |
| *-0* | meets (end points abut - i.e. are equal) |
| *nil* | non-strict (<= or >=). |

Some examples of the new predicates:

| Literal | Temporal Meaning |
|---|---|
| *[a before-1 b]* | $a < b$ |
| *[a between-0-1 b c]* | *[a after-0 b] & [a before-1 c]* |
| *[a during--1 b]* | *start of a >= start of b*, and<br>*end of a < end of b*. |

The new specialist is quite flexible, accepting episodes, time points, or absolute times in any combination as arguments for the predicates (except the timeframe, which must be an episode), where Taugher's programs were quite rigid in the argument patterns accepted. On assertion, if an argument is an absolute time instead of a named time point or episode, the appropriate absolute time bound of the other argument is updated. For example,

*(a before (date '1987 '04 '01 '00 '00 '00))*

would update the upper bound of *a*'s absolute time (the maximum). For evaluation, the appropriate bound is compared against the absolute time, or two absolute times may be compared.

In addition, some new predicates have been introduced to handle durations: *at-most-before, at-most-after, at-least-before, at-least-after, exactly-before*, and *exactly-after*. These take three arguments, of which the first two may be events or time points (no absolute times) and the third denotes the duration between the two in seconds. *At-most-* implies maximum duration, *at-least-* minimum duration, and *exactly-* involves both. To determine the duration between any two points, an exhaustive search must be done between those points, following both in-chain and cross-chain links, to get the best duration bounds (the greatest minimum, and the smallest maximum). Duration information on arcs and implicit in absolute times is used. Details are in [Mil88].

[8] The pseudo-times are numbers generated by the system which reflect the ordering on the chain.

[9] With some additional effort, we could improve the algorithm to save partial results between points so that each cross-chain link need only be looked at once. This would bring the time required back to $O(m)$, the same as for a search without strictness requirements.

## 3. Use of The Temporal Specialist in a More General Environment

The main system for representing knowledge and making inferences uses a resolution-based theorem prover featuring automatic classification of propositions, topical access of clauses for resolution, and type inheritance through a type hierarchy [HaS86]. This system is designed to handle large, diverse bodies of knowledge efficiently. Although the topical access and type hierarchy improve the performance of the theorem prover immensely, it can still suffer from the computational explosions to which all theorem provers are prone. This is especially true when working with the transitive relations involved in temporal inference, so that it is desirable to delegate temporal inference to the temporal specialist as far as possible.

Interesting proofs we can ask the system to do involve "mixed" inference (i.e. involving both temporal relations and others). For example, in the story of Little Red Riding Hood, deciding whether the wolf was alive when everyone was eating the goodies in the basket requires using knowledge that one can only be alive before one is killed, and temporal inference to determine that the episode of eating the goodies came after the episode of killing the wolf. Other examples of the uses of such "mixed" reasoning in planning and problem-solving can be found in [AlK85].

Further extensions were necessary to integrate the specialist into the main system. The main system organizes modal propositions into subnets, with one subnet for each person's mental world. As the subnets may contain contradictory information, the temporal specialist also maintains a separate time-graph (and metagraph) for each subnet[10]. Simple temporal evaluations may be done within the subnets, although at this stage no real modal inference is done by either the temporal specialist or the main theorem prover. Upon completion of proofs by contradiction, the main system retracts all changes made. To remain consistent, the temporal specialist had to be extended to have this capability as well, which is used extensively in generalized resolving and factoring.

Schubert et al. [SPT87] suggest that a specialist can assist a theorem prover in literal evaluation, and generalized resolution and factoring. In addition, the specialist can be used to simplify literals by evaluating functional terms. Assertions must also be entered into the specialist representation for use in future evaluations. Most of the requirements for simple literal evaluation and entry have already been discussed. Generalized resolution and factoring make use of both the entry and evaluation phases to check for resolving or factoring actions.

Figure 2 shows an example of the system in operation, trying to answer a question about the story of Little Red Riding Hood, similar to the one mentioned earlier.

[10] This assumes that all temporal propositions within a mental world are consistent.

The question the system tries to answer is

*Was the wolf alive at some time after everyone talked or after everyone ate the goodies?*[11]

which we translate to

*?[E x_episode [W alive x] & [[x after all-talk] or [x after-1 (end-of eat-goodies)]]]*[12].

In this translation, it is taken for granted that there is a noun phrase referent determination process which has selected the episode here called *all-talk* as the referent of *everyone talked*, and *eat-goodies* for *everyone ate the goodies*. Similarly, *W* is the individual referred to as *the wolf*. It is sometimes possible to handle this with existential quantification (e.g. *[E x_episode [everyone eat goodies x] & [x ...]]*), but this only works satisfactorily when a "yes" answer is expected. If a "no" answer is expected, the system currently answers "unknown", as there may be events fitting that description it doesn't know about (default reasoning would be required otherwise). The example in Figure 2 required about 57.5 seconds - just under a minute. This was with a knowledge base consisting of about 150 propositions (which are normalized into over 450 clauses), some for general knowledge and some for the simplified version of the the Little Red Riding Hood story we use for testing (it consists of over 70 propositions, 20 of which temporally relate episodes in the story). The number of steps required for the proof was small, although the operations accelerated by the specialist would normally have required many applications of temporal axioms.

These uses of the temporal specialist differ from previous approaches in that there are more possiblities for the temporal specialist to assist. In a system like that of Allen et al. [AGF84], for example, which computes transitive closure of input assertions to give constant time lookup later, we would be restricted to the literal evaluation phase only. As their system requires that closure be done for each entry ($O(n^3)$ time) and has no facilities for retraction, generalized resolving and factoring in it would be inefficient, if at all possible.

In the next sections we gloss over the interface mechanism between the theorem prover and the temporal specialist, and concentrate on the operation of the temporal specialist. Details on the interface itself may be found in [Mil88].

### 3.1. Function Evaluation

During literal simplification, the temporal specialist may evaluate temporal functional terms to a more usable entity than the original term. The functions used most commonly are *start-of* and *end-of*, which return the time point for the start and

[11] Although somewhat awkward, this question illustrates all the areas of temporal specialist assistance without being overwhelming.

[12] The question is phrased using *after-1*, and specifying *end-of* to show that the system can actually handle predicates of different strictness, and combinations of time points and events. The questions could just as easily have been phrased as *[E x_episode [W alive x] & [[x after all-talk] or [x after eat-goodies]]]*. Also note that a sort tag (*_episode*) is attached to some terms; this is used to differentiate terms so that the temporal specialist is called only when appropriate.

```
==> ?(E x_episode (W allve x) & ((x after all-talk) or (x after-1 (end-of eat-goodies))))
Entering disproof clauses:
(W ALIVE SCON-380) (depth 1)
((SCON-380 AFTER ALL-TALK) | (SCON-380 AFTER-1 (END-OF EAT-GOODIES))) (depth 1)

  Time Specialist: END-OF (EAT-GOODIES) evaluated to EAT-GOODIESEND
((SCON-380 AFTER ALL-TALK) | (SCON-380 AFTER-1 EAT-GOODIESEND)) (depth 1)

  Time Specialist: Trying to factor (SCON-380 AFTER ALL-TALK) and (SCON-380 AFTER-1 EAT-GOODIESEND)
  Time Specialist: Factored to (SCON-380 AFTER ALL-TALK)
(SCON-380 AFTER ALL-TALK) (depth 1)

  Resolved (W ALIVE SCON-380) in the disproof clause
    (W ALIVE SCON-380)
    against (~ U-VAR-2 ALIVE EPISODE-VAR-1) in ((~ U-VAR-1 KILL U-VAR-2 EPISODE-VAR-2)
        | (~ EPISODE-VAR-1 AFTER EPISODE-VAR-2) | (~ U-VAR-2 ALIVE EPISODE-VAR-1))
    yielding ...
((~ SCON-380 AFTER EPISODE-VAR-1) | (~ U-VAR-1 KILL W EPISODE-VAR-1)) (depth 2)

  Resolved (~ U-VAR-1 KILL W EPISODE-VAR-1) in the disproof clause
    ((~ SCON-380 AFTER EPISODE-VAR-1) | (~ U-VAR-1 KILL W EPISODE-VAR-1))
    against (WOODCUTTER KILL W WOLF-DEMISE) in (WOODCUTTER KILL W WOLF-DEMISE)
    yielding ...
(~ SCON-380 AFTER WOLF-DEMISE) (depth 3)

  Time Specialist: Trying to resolve (~ SCON-380 AFTER WOLF-DEMISE) against (SCON-380 AFTER ALL-TALK)
  Time Specialist: Resolved with residues null
  Resolved (~ SCON-380 AFTER WOLF-DEMISE) in the disproof clause
    (~ SCON-380 AFTER WOLF-DEMISE)
    against (SCON-380 AFTER ALL-TALK) in (SCON-380 AFTER ALL-TALK)
    yielding the null clause.
  NO
```

Figure 2. Example of ECoNet in operation[13]

end of an episode, respectively, and *date* which returns an absolute time representation recognized by the temporal specialist.

### 3.2. Literal Evaluation

When evaluating whether a literal is true, one or more of the evaluation techniques already discussed is used. If the predicate is a duration predicate (e.g. *at-most-before*), the duration is calculated and compared to the one given. For other predicates, if any arguments are absolute times, an absolute time comparison is done. Otherwise, the question is split into several time point evaluation questions, each of which uses the metagraph. The example shown in Figure 2 does not overtly show incidences of literal evaluation, but it was used during the generalized factoring and resolving steps there.

### 3.3. Generalized Resolution and Factoring

Schubert et al. [SPT87] show some examples of when generalized resolving and factoring can be done by a temporal specialist. There are many more cases where these operations can be useful, dependent mainly on information already asserted in the timegraph. Determining whether two literals are resolvable or factorable involves similar methods, so they will be described together. The temporal specialist determines possible unifications that may lead to resolving or factoring actions, and uses the timegraph as a medium in which to compare the literals after substitution.

When resolving or factoring, unification of the arguments of the two literals is required. Since the predicates in the two literals are not necessarily identical, there is no restriction that the arguments be unified in the typical order (i.e. first from literal1 with first from literal2, and so on). The temporal specialist tries all possible unifications, testing after each to see if there is a resolving or factoring action that can be taken. Note that the two literals do NOT need to have the same number of arguments. For example:

| Literals | Unifications |
|---|---|
| (x after y) vs (e1 between e2 e3) | (x/e1, y/e2), (x/e1, y/e3), (x/e2, y/e1), (x/e2, y/e3), (x/e3, y/e1), (x/e3, y/e2) |

For each unification, we try to enter one literal into the time graph, and compare the other one to it within the timegraph. Use of the timegraph allows comparison against the newly entered literal (what we started out to do), as well as other temporal relations asserted earlier (which enables the time specialist to shortcut the proof so drastically). During a generalized resolving attempt, we are looking for a unification that makes the two literals incompatible, or incompatible with the negation of a residue[14]. During a generalized factoring attempt, we look for a unification that makes one literal unnecessary. Details of the algorithm may be found in

[13] This example shows actual output of the system, edited for clarity and brevity. Bold print is user input; the rest, system output. The existentially quantified variable *x* in the question has been converted to the skolem constant *SCON-380* used in the proof. The timegraph contained, among other relations, that *all-talk* is during *eat-goodies*, and that *eat-goodies* is after *wolf-demise*, the episode corresponding to the woodcutter killing the wolf.

[14] A residue (from Stickel's partial theory resolution [Sti83] is a literal (or set of literals) whose negation would make the two literals incompatible (resolvable in one or more steps to the null clause).

[Mil88].

## 4. Conclusions

The intent of this research was to incorporate an efficient temporal reasoner that handled all our temporal inference requirements into a general reasoning environment. The extensions to the temporal specialist itself enabled it to handle more possible temporal inferences, including reasoning with both strict and non-strict relations, at modest computational expense. The specialist accelerates the main theorem prover by performing literal evaluation and generalized resolution and factoring, as well as simplifying literals by functional term evaluation. The resulting hybrid can do some proofs that would normally require numerous steps in just a few.

There is a trade-off between the development time for a specialist and the efficiency it adds to the combined reasoner. The time spent in developing and enhancing the temporal specialist was well invested, since temporal reasoning is so essential in many AI applications. Although the time specialist slows down single steps of the system (because of the greater complexity of tests for resolvability or factorability), it can shorten the number of proof steps drastically. Some problems that previously were not feasible can now be done in reasonable time. One drawback in such a system is that some of the inferences are hidden within the temporal specialist, and are thus "invisible". However, since the type of temporal inference performed by the specialist is well understood and almost "obvious", justification is not essential - provided bugs and "holes" have been eliminated.

Work is underway to add more specialists to the system (namely a number/arithmetic specialist, a color specialist, and a set/list specialist), and to examine the interactions and possible communication between them.

## References


[All83] James F. Allen, Maintaining Knowledge about Temporal Intervals, *Communications of the ACM 26*, 2 (1983), 832-843.

[All84] James F. Allen, Towards a General Theory of Action and Time, *Artificial Intelligence 23*, 2 (1984), 123-154.

[AGF84] J.F. Allen, M. Giuliano and A.M. Frisch, The HORNE Reasoning System, Tech. Rep. 126, Computer Science Department, University of Rochester, Rochester, NY, 1984.

[AlK85] J.F. Allen and H.A. Kautz, A Model for Naive Temporal Reasoning, in *Formal Theories of the Commonsense World*, J.R. Hobbs and R.C. Moore (ed.), Ablex, Norwood, NJ, 1985, 251-268.

[HaS86] Johannes de Haan and Lenhart K. Schubert, Inference in a Topically Organized Semantic Net, *Proc. AAAI-86 I*, (1986), 334-338.

[Mil88] Stephanie A. Miller, Time Revisited, M.Sc. Thesis, Department of Computing Science, University of Alberta, 1988.

[SPT87] L.K. Schubert, M.A. Papalaskaris and J. Taugher, Accelerating Deductive Inference: Special Methods for Taxonomies, Colours and Times, in *The Knowledge Frontier*, N. Cercone and G. McCalla (ed.), 1987.

[Sti83] Mark E. Stickel, Theory Resolution: Building in Nonequational Theories, *Proc. AAAI-83*, Washington, D.C., 1983, 391-397.

[Tau83] J. Taugher, An Efficient Representation for Time Information, M.Sc. Thesis, Department of Computing Science, University of Alberta, 1983.

[ViK86] Marc Vilain and Henry Kautz, Constraint Propagation Algorithms for Temporal Reasoning, *Proc. AAAI-86 I*, (1986), 377-382.

# Reasoning in Temporal Domains: Dealing with Independence and Unexpected Results

Scott D. Goodwin
Department of Computing Science,
University of Alberta,
Edmonton, Alberta, Canada, T6G 2H1
scott@alberta.uucp

**Abstract**

Much interest has been focused on nonmonotonic reasoning in temporal domains since Hanks and McDermott discovered that intuitive temporal axiomatizations give rise to the multiple model/multiple extension problem. Here we consider nonmonotonic reasoning in temporal domains from the perspective of theory formation. We claim that this framework can be applied to temporal reasoning in a simple and intuitive way, and we discuss why approaches that advocate explicit axiomatization of causality are not desirable. Two problems concerning model ordering/theory preference approaches are described and a solution to one of these problems is given. Finally, we discuss why the second of these problems is not as easy to solve as has been suggested; we discuss requirements for its solution.

Keywords: Nonmonotonic Reasoning, Theory Formation, Temporal Reasoning, Frame Problem, Knowledge Independence

## 1 Introduction

Recently, much interest has been focused on nonmonotonic reasoning in temporal domains. Since Hanks and McDermott [HM86] discovered that temporal axiomatizations give rise to the multiple model or multiple extension problem, many solutions have been proposed. Just as researchers were beginning to reach a consensus that what was needed was some sort of model ordering (or theory preference) scheme (predominantly, based on the chronological order of actions [Kau86,Sho86,GG87a]), a serious challenge was issued. It was suggested that it may be necessary to take a different view of the domain by explicitly including causality in our ontology. In their proposals, Loui [Lou87], Lifschitz [Lif87], and Haugh [Hau87] show that the multiple model or multiple extension problem can be avoided by directly axiomatizing causal relations. Haugh goes further to show that chronological preference methods fail to correctly model common sense when the domain involves (what we call) independent relations whose truth is unknown (we explain this further in section 4) or when the result of a sequence of actions is different from the expected result.

Regardless of Haugh's claims, the causal axiomatization approach has serious drawbacks. There is good reason to claim that this approach is not desirable. Hanks and McDermott have criticized it as follows:

> "... if we adopt one of these solutions we have in effect allowed technical problems in the logic to put too much pressure on our knowledge representation. Part of the presumed appeal of expressing theories in logic is that the language should allow us to explore our intuitions about domains like naive physics .... What we should be thinking about at the logical level are issues like whether time is continuous or discrete, how many modalities there are, whether time is made out of points or intervals, etc. But if we follow the example of those who would have us change our ontology, we in effect have to phrase our axioms in one particular way ... just to get around the inadequacies of our inference mechanism." [HM87]

Any solution to the problem of nonmonotonic reasoning in temporal domains must address both Hanks' and McDermott's criticisms and the concerns raised by Haugh.

In what follows, nonmonotonic reasoning in temporal domains is considered from the perspective of theory formation [PGA87]. We claim that the observe-hypothesize-predict-test-revise framework can be applied to temporal reasoning in a simple and intuitive way, and that Hanks' and McDermott's criticisms and Haugh's concerns can both be addressed in the framework. In the next section, we begin by explaining the theory formation framework and reviewing the basic elements of one theory formation approach: Theorist. Section 3 reviews an earlier proposal for applying the framework to temporal domains. In section 4, we discuss the first problem described by Haugh, namely, that chronological preference methods give incorrect results when there are independent relations. We show how this problem arises and how to deal with it without the concessions made in Loui's, Lifschitz's, and Haugh's proposals. In the following section, we mention the second problem Haugh raises: the problem of dealing with sequences of actions which don't result in their normally expected outcome. We do not solve this problem, but indicate that the

solution involves distinguishing between the task of *predicting* the expected outcome of a sequence of actions and the task of *explaining* an anomalous result. We will conclude by summarizing the main points of the paper.

## 2 A Theory Formation Framework

The development of nonmonotonic reasoning systems was, in part, motivated by the inability of logical deduction to capture the forms of rational inference typically involved in common sense reasoning. Israel has argued that nonmonotonic reasoning should be considered within the framework of scientific theory formation [Isr80]. In the spirit of Israel's proposal, Poole and his colleagues have been investigating the theory formation approach to common sense reasoning in the Theorist project [PGA87]. Based on a philosophy inspired by Popper [Pop58], Theorist views reasoning as scientific theory formation (rather than as deduction). Science is concerned, not merely with collecting facts, but also with finding explanations, making predictions, and testing and revising theories. Reasoning in the Theorist framework involves building theories that explain observations[1]. A *theory* $D$, consisting of instances drawn from a set of *possible hypotheses* $\Delta$, is said to *explain* a set of *observations* $G$ if the theory, together with the *facts* $F$, logically implies the observations; it must also be consistent with the facts. Formally,

> $D$ explains $G$ if $D \subseteq$ instances of elements of $\Delta$ such that,
> $F \cup D \models G$, and $F \cup D$ is consistent.

In most implementations of Theorist, the representation language is full first-order clausal logic; in this case, $F$, $D$, and $O$ are sets of sentences, and $\Delta$ is a set of sentence schemas.

To illustrate, the well known birds fly example is axiomatized below. The syntax used in the axiomatization is a slight simplification of that used in current Theorist implementations (cf. [PG87]). The statement **fact** *Clause* means that *Clause* $\in F$ and the statement **default** *Clause* means that *Clause* $\in \Delta$.[2]

> **fact** bird(tweety).
> **fact** bird(X) ← penguin(X).
> **fact** ¬flies(X) ← penguin(X).
> **default** flies(X) ← bird(X).

Now we can explain $G = \{$*flies(tweety)*$\}$ with the theory $D$ = $\{$*flies(tweety)* ← *bird(tweety)*$\}$ formed by instantiating the default *flies(X)* ← *bird(X)* with $X$ = *tweety*. Should we later learn that *penguin(tweety)* then the theory $D$ would no longer be consistent with the facts and, consequently, we are no longer able to explain $G = \{$*flies(tweety)*$\}$.

Note that, in general, there may be multiple theories that explain the observations. These correspond to multiple minimal models in circumscription [McC80] and to multiple extensions in default logic [Rei80]. To discriminate among multiple competing theories, theory preferences may be specified. This meta-level knowledge can be used to prune theories that are irrelevant, or to order theories by utility or likelihood, etc. We will describe an example of theory preference in the next section where we examine the use of Theorist for temporal reasoning.

## 3 Temporal Reasoning using Theorist

The Theorist framework has been applied to temporal reasoning to deal with the frame problem (for the details, see [Goo87,GG87a]). Here we provide a brief sketch of the results.

In Theorist, temporal domain knowledge can be represented as facts, possible hypotheses, and theory preferences; but before this can be done, some choice of ontology is necessary—for simplicity, the ontology of situation calculus is used. In the simplest case, where there is complete knowledge of the initial situation and the effects of actions, facts are used to represent these. The non-effects (or things unaffected by actions) are expressed by representing the usual frame axioms as defaults. As Hanks and McDermott noted, this style of representation leads to the multiple extension problem. To illustrate, consider the well known Yale Shooting Scenario (YSS) [HM86]. The scenario starts with a gun that is initially unloaded and a potential victim who is alive. The sequence of actions, load the gun, wait, and shoot is considered. We are interested in reasoning about the result of the actions. Below is an axiomatization of this scenario.

> **fact** ¬loaded(0).
> **fact** alive(0).
> **fact** loaded(do(load,S)).
> **fact** ¬alive(do(shoot,S)) ← loaded(S).
> **fact** ¬loaded(do(shoot,S)).
> **default** loaded(do(A,S)) ↔ loaded(S).
> **default** alive(do(A,S)) ↔ alive(S).

For brevity, we introduce the following synonymous names for situations of interest.

$$1 \equiv \text{do(load,0)},\ 2 \equiv \text{do(wait,1)},\ 3 \equiv \text{do(shoot,2)}$$

The above axiomatization gives rise to many theories that describe the possible ways the truth-value of relations might persist over the action sequence: load, wait, shoot. These theories are the subsets of the set of instances of elements of $\Delta$ that are consistent with $F$, that is, they are the consistent subsets of:

> {loaded(1) ↔ loaded(0), alive(1) ↔ alive(0),
> loaded(2) ↔ loaded(1), alive(2) ↔ alive(1),
> loaded(3) ↔ loaded(2), alive(3) ↔ alive(2)}

We are ignoring instances of $\Delta$ that are irrelevant to the sequence of actions being considered (e.g., alive(do(wait,0)) ↔ alive(0)). These could be pruned using theory preference, but, in practice, this is not necessary because the resolution-based implementation of Theorist only considers

[1] Goodwin and Gagné [GG87b] and Poole [Poo88] have proposed elaborations of the Theorist framework incorporating prediction.

[2] See [Poo88] for an explanation of various kinds of possible hypotheses—here we are only concerned with defaults.

instances of $\Delta$ relevant to $G$. The above set of instances of $\Delta$ has two maximal[3] consistent subsets, namely,

$$\begin{aligned} D_1 &= \{\text{loaded}(2) \leftrightarrow \text{loaded}(1), \\ &\quad \text{alive}(1) \leftrightarrow \text{alive}(0), \\ &\quad \text{alive}(2) \leftrightarrow \text{alive}(1)\} \end{aligned}$$

and

$$\begin{aligned} D_2 &= \{\text{loaded}(3) \leftrightarrow \text{loaded}(2), \\ &\quad \text{alive}(1) \leftrightarrow \text{alive}(0), \\ &\quad \text{alive}(2) \leftrightarrow \text{alive}(1), \\ &\quad \text{alive}(3) \leftrightarrow \text{alive}(2)\}. \end{aligned}$$

Here $F \cup D_1 \models \neg alive(3)$ and $F \cup D_2 \models alive(3)$. In spite of these multiple theories, the common sense expectation in the shooting scenario is that the gun remains loaded while waiting and the victim dies (See [HM86]). The theory corresponding to this is $D_1$.

The primary result in [Goo87,GG87a] is that multiple competing theories can be discriminated on the basis of a chronological preference method called *chronological maximization of persistence*. By specifying theory preference knowledge as a partial ordering on the set of all possible theories, a preferred theory (in this case $D_1$) can be selected. The partial ordering is similar to the ordering proposed for circumscription approaches (cf. [Kau86]), except that instead of chronologically minimizing clippings, we chronologically maximize (in the sense of set inclusion[4]) the occurrence of frame defaults. The intuition behind this ordering is that since most things are unaffected after performing an action, we should maximize persistence. In addition, to reflect that actions occur in sequence, we maximize persistence—step by step—in the order the actions occur, i.e., chronologically. Thus for instance, $D_1$ is preferred over $D_2$ since up to the completion of the wait action, $D_1$ contains all of the frame defaults that $D_2$ contains, but $D_2$ does not contain all of the frame defaults contained by $D_1$ (for instance, loaded(2) $\leftrightarrow$ loaded(1) is not in $D_2$).

Having represented the YSS as described above, we can *predict* the outcome of a sequence of actions, for instance, whether $\neg alive(3)$ is expected, by finding an explanation for it whose underlying theory is the preferred theory from the space of possible theories (when there are multiple preferred theories, we make predictions from their disjunction). Further elaborations of the approach to incorporate default action effects, communication conventions, weak constraints, error recovery, etc. are still under investigation. Having given this brief overview of Theorist applied to temporal reasoning, we are now in a position to consider the concerns raised by Haugh.

## 4 Dealing with Independent Knowledge

The first of the problems Haugh draws our attention to is that chronological preference methods arrive at incorrect conclusions when there is (what we call) independent knowledge. Haugh illustrates this problem, which we call the *knowledge independence problem*, with the following example. Suppose we have a scenario similar to the YSS except that while the wait action is performed someone will attempt to unload the gun[5] and will be successful if he knows how. Further, we make no commitment as to whether the person knows how to unload the gun. We might axiomatize this by adding the axiom:

**fact** ¬loaded(do(wait,S)) ← knows_how.

to the YSS axiomatization given earlier.

Since the truth-value of *knows_how* is unknown, the new fact does not constrain the set of consistent theories so the maximal consistent theories are $D_1$ and $D_2$ as before (see section 3). If chronological preference methods are applied to the axiomatization, the conclusion arrived at is that the gun remains loaded and the victim dies; that is, the preferred theory is $D_1$. This theory involves an implicit assumption that the person does not know how to unload the gun (i.e., $\neg knows\_how$ is derivable from $F \cup D_1$). Clearly this is not the desired result. The result should be (according to common sense) that, because we do not know whether the person knows how to unload the gun, we do not know if the gun is successfully unloaded, and therefore, we do not know if the victim will be killed.

Note that if *knows_how* were known to be false, chronological preference would correctly conclude that the victim dies (i.e., $D_1$ would be the preferred theory) and if *knows_how* were known to be true chronological preference would correctly conclude that the victim lives (i.e, since $D_1$ would be inconsistent $D_2$ would be the preferred theory). The problem arises when the truth-value of *knows_how* is unknown because, as a side effect of making persistence assumptions about the *loaded* relation, we conclude something about the *knows_how* relation. When the truth-value of *knows_how* is not known, the conclusions drawn should be conditional. For the above example, the answer should be $alive(3)$ if *knows_how* and $\neg alive(3)$ if *knows_how*.

The knowledge independence problem is not due to any inadequacy of the chronological preference methods; rather, it is due to a weak axiomatization—knowledge about the independence of relations (*knows_how*) has not been represented. To solve this problem, we need to establish what independence means and how it can be represented. Intuitively, independence of a relation means that its truth-value constrains the assumptions that may be made in drawing conclusions and not vice versa, i.e., the truth-value of independent relations are a priori determined though perhaps unknown presently. So characterizing a domain involves deciding what knowledge is independent of the assumptions. Once this is decided, the next problem is how to represent and reason about knowledge independence.

With circumscription, it is well known that, depending on the circumstances, certain relations should be allowed to vary and others should remain fixed [McC86,Lif86]. One useful criterion is that relations should be fixed if they are

[3] If we interpret defaults as meaning normally then, intuitively, we want to maximize normality (in some sense).

[4] Actually, the proposal was more general; it provided for domain dependent partial orderings other than subset inclusion.

[5] We can view the effect of the wait action as the combined effect of concurrent actions.

determinants and should be allowed to vary if they are resultants. The notion of fixed relations seems to capture at least some of the intuition behind independence. Whether it completely characterizes independence is not clear, but for now we will take the two to be equivalent.

In the modified YSS example above, *knows_how* is an independent relation and, therefore, should be treated as fixed. Poole [Poo87] has proposed a method to incorporate the concept of fixed and varying relations in Theorist. This elaboration allows the side effects of making assumptions to be controlled and made explicit by allowing conditional answers. We first review Poole's proposal and then see how it leads to a solution of the problem of dealing with independence.

In addition to the set of facts $F$ and the set of possible hypotheses $\Delta$, Poole introduces a set of fixed relations $\Theta$. This new set is intended to include relations which we do not want to make implicit assumptions about. Poole defines two new forms of explanation: conditional and unconditional.

**Definition 1 ([Poo87])** We say that $g$ is *conditionally explainable* from $F$, $\Delta$ and $\Theta$, if there is a set $D$ of instances of elements of $\Delta$, and a formula $C$ made of instances of elements of $\Theta$ (under conjunction, disjunction, negation), such that

1. $F \wedge C \wedge D \models g$
2. $F \wedge C \wedge D$ is consistent
3. if $F \wedge C \wedge D \models \theta$, where $\theta$ is a formula made from elements of $\Theta$, then $F \wedge C \models \theta$.

$D$ is said to be the theory that explains $g$, and $C$ is the condition for D.

**Definition 2 ([Poo87])** We say that $g$ is *unconditionally explainable* from $F$, $\Delta$, $\Theta$ if $g$ is conditionally explainable with theories $D_i$ and corresponding conditions $C_i$ for $i = 1, n$, such that $F \models C_1 \vee ... \vee C_n$.

The problem of dealing with independence can now be solved by first declaring the independent relations as fixed and then answering the query, "Do you expect $g$ to be true?" with "yes"—*unconditionally predicting g*—if there is an unconditional explanation of $g$ whose underlying theory is the preferred theory (according to chronological preference), or answer with "yes if $C$ is true"—*conditionally predicting g* if $C$—if there is a conditional explanation for $g$ whose underlying theory is preferred. For instance, in the modified YSS example, we should add

**fixed** knows_how.

to the axiomatization. Here **fixed** *Atom* means that *Atom* $\in \Theta$. Now, by definition 1, we can conditionally explain $\neg$*alive(3)* with the condition $C = \{\neg$*knows_how*$\}$ and the theory $D_1$ (see section 3). Since $D_1$ is the preferred theory (of theories consistent with $F \cup C$), we conditionally predict $\neg$*alive(3)* if $\neg$*knows_how*. We can also conditionally explain *alive(3)* with the condition $C = \{$*knows_how*$\}$ and the theory $D_2$. In this case, $D_1$ is inconsistent with $F \cup C$ and $D_2$ is the preferred theory. Therefore, we conditionally predict *alive(3)* if *knows_how*. We should note that there is a conditional explanation of *alive(3)* with condition $C = \{\neg$*knows_how*$\}$ and theory $D_2$, but in this case, $D_2$ is not the preferred theory since $D_1$ is consistent with $F \cup C$ and $D_1$ is better than $D_2$. Therefore, we cannot conditionally predict *alive(3)* if $\neg$*knows_how*.

More importantly, we cannot unconditionally predict either $\neg$*alive(3)* or *alive(3)*. In the first case, this is because there is no unconditional explanation of $\neg$*alive(3)* since every conditional explanation of it has the condition $C = \{\neg$*knows_how*$\}$ and the requirement from definition 2 that $F \models C$ is not satisfied. In the second case, there is an unconditional explanation of *alive(3)* with theory $D_2$ since the disjunction of the conditions of the conditional explanations for *alive(3)* (see above) follows from the facts as required by definition 2 (i.e., $F \models \{$*knows_how*$\} \vee \{\neg$*knows_how*$\}$). Nevertheless, $D_2$ is not the preferred theory since $D_1$ is consistent with $F \cup \{$*knows_how* $\vee \neg$*knows_how*$\}$ and $D_1$ is better than $D_2$. Therefore, we cannot unconditionally predict *alive(3)*.

Besides solving the problem of dealing with independence in temporal reasoning, this approach has an added advantage for planning. Forming conditional plans is easier because the relevant conditions are automatically identified via conditional explanation. We should note that conditional and unconditional explanation have been implemented for Theorist [You87], but conditional and unconditional prediction has not as yet been implemented (though we foresee no great difficulties in doing this). Let us now turn to the second problem raised by Haugh.

## 5 Dealing with Unexpected Results

The final example given by Haugh illustrates another interesting problem. When the result of a sequence of actions conflicts with the normally expected result, applying chronological preference methods doesn't give the intuitive explanation of the anomalous result. Let us illustrate this using Haugh's robot example. In the scenario, there is a robot whose motion is controlled be gears which can be locked or unlocked. Initially a person has the ability to lock the robot's forward gears, the reverse gears are locked, the forward gears are unlocked, and the robot is not moving either forward or reverse. After the sequence of actions: *wait, lock_fwd, try_moving*, the robot is observed moving. The Theorist axiomatization corresponding to the one given by Haugh for this scenario is as follows:

**fact** can_lock_fwd(0).
**fact** locked_rev(0).
**fact** ¬locked_fwd(0).
**fact** ¬moving_fwd(0).
**fact** ¬moving_rev(0).
**fact** locked_fwd(do(lock_fwd,S)) ← can_lock_fwd(S).
**fact** moving_fwd(do(try_moving,S)) ← locked_rev(S).
**fact** moving_rev(do(try_moving,S)) ← locked_fwd(S).
**fact** ¬moving_rev(S) ← moving_fwd(S).
**default** can_lock_fwd(do(A,S)) ↔ can_lock_fwd(S).
**default** locked_rev(do(A,S)) ↔ locked_rev(S).

**default** locked_fwd(do(A,S)) ↔ locked_fwd(S).
**default** moving_fwd(do(A,S)) ↔ moving_fwd(S).
**default** moving_rev(do(A,S)) ↔ moving_rev(S).

This is obtained by using Haugh's axiom T9 to translate *Causes(p,c,e)* to *T(p,s)* ∧ *Result(c,s,s')* ⇒ *T(e,s')* which translates to *e(do(c,S))* ← *p(S)*. Whenever *T(p,s)* is true in Haugh's axiomatization, *p(S)* is true in ours, and vice versa. Haugh uses the predicate *Causes* to define his minimality criteria: *Potential_cause* and *Determined_cause*. As we are not concerned with these criteria here, we do not need to include *Causes* in our axiomatization.

Haugh goes on to explain that normally we would expect the robot to be unable to move after the first two actions, since we expect both gears to be locked. Nevertheless, the robot is observed moving after the final action so somehow one of the gears must have become unlocked. Haugh observes that the chronological preference methods would conclude that the reverse gear must have been unlocked for no good reason. Common sense gives us no basis to decide which gear is unlocked.

While Haugh's claim is correct, there is a slight technical problem with the axiomatization he gives: from the last three facts above, we can derive

$$\neg locked\_rev(S) \leftarrow locked\_fwd(S)$$

(the corresponding statement is derivable in Haugh's axiomatization). Apparently Haugh overlooked this in his analysis. As it stands, the conclusion that the reverse gear is unlocked is the expected result of locking the forward gear.

The axiomatization can be adjusted to represent the scenario Haugh had intended by replacing the two facts describing the *try_moving* action with:

**fact** moving_fwd(do(try_moving,S)) ←
    locked_rev(S), ¬locked_fwd(S).
**fact** moving_rev(do(try_moving,S)) ←
    locked_fwd(S), ¬locked_rev(S).

With this corrected axiomatization, the point Haugh was trying to make is indeed true. Chronological preferences methods will conclude that the reverse gear is unlocked, even though there is no basis to decide which gear is unlocked.

The source of the difficulty here is that, while the expected state of affairs after performing the sequence of actions is that both gears are locked and no movement occurs, the actual state of affairs is different. Chronological preference methods are inappropriate for explaining anomalous results. This is because they maximize the normality (persistence of truth-values) of each action in the order of their occurrence. The strategy of chronological preference, in effect, determines the normal effect of an action assuming that the actions preceding it had their normal effect. When each action has its normal effect, the outcome is the one predicted by chronological preference. When the observed outcome is not the expected outcome, one explanation is that one or more actions didn't have its normal effect. Chronological preference, however, postpones the abnormality as late as possible. Consequently, chronological preference has a built-in assumption that if one of the actions in a sequence is abnormal, it is expected to be the last one. Clearly, since there is no justification for this assumption, chronological preference is an inappropriate heuristic for explaining anomalous results.

There are other possible explanations for anomalous results as well, namely: the specification of the initial situation is incorrect, or the specification of the action effects is incorrect, or the actual sequence of actions was different from the specified sequence of actions, or the observation of the outcome was faulty. The problem of determining whether specifications are faulty is what Hayes has called the prediction problem [Hay71] and is part of the larger problem of belief maintenance and revision.

In light of all the possible explanations for anomalous results, it should be clear that the solution is far from easy. Since the source of the anomaly may be a faulty axiomatization, approaches that suggest giving up the preferred theory and moving to the next best theory to explain the anomaly cannot be considered complete solutions. If the axioms are faulty, we should not have much confidence in their corresponding theories. Probably what's needed is more knowledge about the ways in which actions can be abnormal. With this knowledge, a diagnosis approach can be used to explain unexpected results and possibly prescribe treatments, i.e., revisions to the axioms. This remains an area for further research.

## 6 Conclusion

We have seen how the scientific theory formation approach can be applied to temporal reasoning through the use of a theory preference criterion called chronological maximization of persistence. We have also reviewed Haugh's criticisms of this technique, particularly, that it draws incorrect conclusions when there is independent knowledge. A solution to this problem was proposed; that is, independent relations should be declared as fixed predicates. In conjunction with this, unconditional prediction can be used to determine if a result is expected independently of the truth or falsity of the independent relations. As well, conditional prediction can be used to determine the conditions underwhich particular results are expected.

Some of the advantages of this approach over that suggested by Haugh (and similar proposals by Loui and Lifschitz) are:

1. It addresses Hanks' and McDermott's criticisms; we are not forced to phrase our axioms in terms of causality;
2. We don't have to decide when to use potential causes and when to use determined causes (cf. [Hau87]), and
3. We can easily form conditional plans because conditional prediction automatically identifies the relevant conditions.

Nevertheless, further study is necessary to answer questions about whether there is a more general notion of independence than that expressible by fixed relations. For instance, do we need to talk about independence of objects, indepen-

dence of formulae, independence of relations w.r.t. certain assumptions but not others, etc. We need a specification of general independence and a means to represent and reason about it. A general specification of independence is obviously related to probabilistic independence, and further understanding of representations based on probability is warranted (e.g., [Ale87,Bac88]). In any case, independence seems to be an important concept that should be further investigated.

Finally, we saw how the problem of dealing with anomalous results is more complex than sometimes portrayed. Solving it may require extra domain knowledge about the way in which anomalous results can arise. Perhaps a combination of diagnostic reasoning and belief maintenance is needed for this problem. The Theorist project is attempting to bring these and other seemingly disparate aspects of hypothetical reasoning within one unified framework. Much work remains to be done.

## Acknowledgements

I have benefited greatly from the ideas, criticisms, and encouragement of many people. Deep thanks to my supervisor, Randy Goebel. Thanks also to George Ferguson, Patrick Fitzsimmons, Abdul Sattar, Gurminder Singh, and Bonita Wong at the University of Alberta and David Poole, Eric Neufeld, Denis Gagné at the University of Waterloo who have contributed to this work through many discussions and debates. Special thanks to an anonymous reviewer for valuable criticisms and suggestions.

## References

[Ale87] R. Aleliunas. *Mathematical Models of Reasoning: Competence Models of Reasoning about Propositions in English & Their Relationship to the Concepts of Probability.* Research Report CS-87-31, Department of Computer Science, University of Waterloo, Waterloo, Ontario, July 1987.

[Bac88] F. Bacchus. Statistically founded degrees of belief. In *Proceedings of the Seventh Canadian Conference on Artificial Intelligence*, June 1988. [in this volume].

[GG87a] R.G. Goebel and S.D. Goodwin. Applying theory formation to the planning problem. In *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*, pages 207–232, Morgan Kaufmann, Los Altos, California, April 1987.

[GG87b] S.D. Goodwin and J.D.D. Gagné. Explanation and prediction. May 1987. [unpublished].

[Goo87] S.D. Goodwin. *Representing Frame Axioms as Defaults.* Research Report CS-87-48, Department of Computer Science, University of Waterloo, Waterloo, Ontario, July 1987.

[Hau87] B. Haugh. Simple causal minimizations for temporal persistence and projection. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 218–223, Morgan Kaufmann, Los Altos, California, July 1987.

[Hay71] P.J. Hayes. A logic of actions. *Machine Intelligence*, 6:495–520, 1971.

[HM86] S. Hanks and D.V. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 328–333, Morgan Kaufmann, Los Altos, California, August 1986.

[HM87] S. Hanks and D.V. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, November 1987.

[Isr80] D.J. Israel. What's wrong with non-monotonic logic? In *Proceedings of the First National Conference on Artificial Intelligence*, pages 99–101, Stanford University, Stanford, California, August 18-21 1980.

[Kau86] H. Kautz. The logic of persistence. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 401–405, Morgan Kaufmann, Los Altos, California, August 1986.

[Lif86] V. Lifschitz. Pointwise circumscription: Preliminary report. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 406–410, Morgan Kaufmann, Los Altos, California, August 1986.

[Lif87] V. Lifschitz. Formal theories of action. In *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*, pages 35–57, Morgan Kaufmann, Los Altos, California, April 1987.

[Lou87] R.P. Loui. Response to Hanks and McDermott: Temporal evolution of beliefs and beliefs about temporal evolution. *Cognitive Science*, 11, 1987.

[McC80] J. McCarthy. Circumscription—A form of nonmonotonic reasoning. *Artificial Intelligence*, 13(1 & 2):27–39, 1980.

[McC86] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28(1):89–116, 1986.

[PG87] D.L. Poole and S.D. Goodwin. A Theorist to Prolog compiler. August 1987. [unpublished].

[PGA87] D.L. Poole, R.G. Goebel, and R. Aleliunas. Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352, Springer-Verlag, New York, 1987.

[Poo87] D.L. Poole. *Fixed Predicates in Default Reasoning.* Research Report CS-87-11, Department of Computer Science, University of Waterloo, Waterloo, Ontario, February 1987.

[Poo88] D.L. Poole. Default and abductive reasoning: An architecture for explanation and prediction. *Journal of Intelligent Systems*, 1988. [submitted].

[Pop58] K. Popper. *The Logic of Scientific Discovery.* Harper & Row, New York, 1958.

[Rei80] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1 & 2):81–132, 1980.

[Sho86] Y. Shoham. Chronological ignorance: Time, nonmonotonicity, necessity and causal theories. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 389–393, Morgan Kaufmann, Los Altos, California, August 1986.

[You87] M. Young. Implementing fixed predictates in Theorist. In *Experiments in the Theorist Paradigm: A Collection of Student Papers on the Theorist Project*, Department of Computer Science, University of Waterloo, May 1987. [Research Report CS-87-30].

# A Syntactic Approach to Mental Correspondence*

*Anthony S. Maida*

Department of Computer Science
333 Whitmore Laboratory
Pennsylvania State University
University Park, PA 16802

**Abstract**

Mental correspondence occurs when two cognitive agents are attending to (i.e., thinking about) the same thing. This paper shows a method to represent beliefs about mental correspondence. We illustrate the method on a problematic example in which one of two agents has a rather vague concept of some entity, and the other agent is thinking about *whatever* the first agent is thinking of. In the course of treating the example we address the following: 1) specificity verses non-specificity of objects of belief; 2) quantification across belief contexts; and, 3) belief nesting.

**Keywords.** Knowledge, belief, quotation, quantifying-in, intension.

## Introduction.

One approach to the representation of natural language belief sentences involves the use of quoted sentences in first-order logic. This paper applies this "quoted-language" approach to the representation of sentences which convey information about mental correspondence. In particular, our goal is to represent, using quoted predicate calculus (QPC), significant components of the following sentence first used by Geach (1967, p. 628): *Hob thinks a witch blighted Bob's mare, and Nob wonders whether she (the same witch) killed Cob's sow.* Clearly, Nob is thinking about whatever Hob is thinking of. However, Hob may not have a specific witch in mind. Cresswell (1985, p. 143) and Pendlebury (1982, p. 347) ask how Nob could be thinking of what Hob is thinking of if Hob is not thinking of anything in particular. This paper shows how such "mental correspondence" information can be represented in a syntactically-based belief reasoner. To achieve this, we must cope with the following: 1) specificity verses non-specificity of objects of belief; 2) correspondence of mental representations in different belief contexts; and, 3) belief nesting. The problem of mental correspondence, and our approach to it, is important for the following reason. The approach illustrates a more flexible means to quantify across belief contexts than by wiring in assumptions about standard names or rigid designators. This flexibility is necessary in commonsense domains, such as natural language understanding.

**Relation to Other Work.** The approach described here is based on Haas (1986) and Perlis (1985) in that it uses quoted predicate calculus. This work evolved from the earlier work of Moore and Hendrix (1979) and Konolige (1982), which identified an agent's beliefs with formulas in a first-order language. Perlis (1985, 1986) showed that the quoted-language approach is not doomed to inconsistency. Haas (1986) showed that reasoning about beliefs could be done efficiently by use of a process called simulative reasoning. Simulative reasoning was known before Haas (cf., Creary, 1979; Konolige, 1982; Moore, 1977, p. 224) but he showed it could be applied to situations involving some kinds of incomplete knowledge.

## Non-specificity of Belief.

It is difficult to represent many simple natural language assertions of belief. One well known sentence taken from Quine (1956) appears below.

{1} Ralph believes there is a spy.

The sentence has two interpretations (cf., Quine, 1956). One interpretation is that Ralph believes there is a spy but Ralph does not suspect any individual in particular. This, we will call the "non-specific" reading. The other interpretation is that Ralph does suspect some individual in particular. This, we will call the "specific" reading.

**Semantics: Ralph's Data Base State.** We assume that Ralph, and other agents, maintain beliefs by use of expressions in a particular mental language, namely, quoted predicate calculus (QPC). Therefore, we know what expression is in Ralph's mind when he believes "non-specifically" that there exists a spy. Ignoring variable renamings, an expression something like: *(exists (x) (is-spy x))* will reside in Ralph's mental data base. For the specific reading, there is less certainty as to what Ralph's belief is.

* - This research was supported in part by NSF grant CRR-8716776. Thanks to John Barnden, Mingqi Deng, Minkoo Kim, and Joe Niederberger for discussion of ideas in this paper.

The following are example propositional sentences, which if any resided in Ralph's data base, would make the specific reading true: *(is-spy Ortcutt)* or *(is-spy (father-of Seymor))*. For the specific reading to be true, there must exist some representation in Ralph's data base with some, as yet, unspecified properties such as being a proper name or perceptual representation. We are unable to fully specify these properties other than mentioning that they may require full-fledged knowledge-based reasoning about designators. An alternative approach based on standard names, and which does not make assertions about designators, has been explored by Konolige (1982).

### Definition of the Mental Language.

**Quotes.** We refer to propositions and parts of propositions by use of a single quote mark prefixed to the object we are refering to. A quoted list is equal to the list of quoted elements. In complex formulae, a preceeding comma unquotes an implicitly quoted element.

**Explicit Belief of an Agent.** There is a predicate "believes-that" which takes two arguments. The first denotes an agent. The second argument denotes a sentence. An expression of the form: *(believes-that Agent Propos)* is true just in case: 1) the propositional sentence designated by "propos" resides in the data base for the designated agent; and, 2) the propositional sentence is marked as true in that data base. When these two circumstances are satisfied, we will say that the sentence is in the agent's belief set.

### Returning to Ralph's Beliefs.

We can represent that Ralph non-specifically believes there is a spy:

{2} (believes-that Ralph '(exists (x) (is-spy x)))

This expression states that the sentence *(exists (x) (is-spy x))* resides in Ralph's data base and is marked as true. We can represent the specific interpretation by the expression:

{3} (exists (x) (believes-that Ralph (list 'is-spy x)))

Expression {3} is true if there exists a quoted term (e.g, "'Fido,") such that the expression "(list 'is-spy x)" is equal to a designator for a propositional sentence and this sentence is in Ralph's belief set. If the quoted term were "'Fido," the sentence would be "(is-spy Fido)." Representations {2} and {3} have been used by Perlis (1985).

**Skolemizing Formulas.** We will skolemize formulas because non-specific existential assertions can set up mental handles that can be subsequently referred to in natural language discourse. Consider the sentence sequence below:

> Ralph believes (non-specifically) that there is a spy. Ralph wondered whether the spy was clever.

The phrase "the spy" in the second sentence accesses a handle that was set up by the first sentence; but the first sentence said that Ralph did not have any particular individual in mind, so where does this handle come from? One way to get this handle is by skolemizing existential assertions. This is because the representational form for the non-specific reading matches that for a specific reading. This technique, we will apply to the Geach sentence.

**Representing Unresolved Ambiguity.** Let us skolemize expressions {2} and {3}. We now assume that Ralph's mental language is *skolemized* QPC. Therefore we can skolemize the term designating the expression that would be used in his data base. This gives us expression {5}. The skolemized version of {3} would be {6}.

{5} (believes-that Ralph '(is-spy skolem-1))
{6} (believes-that Ralph (list 'is-spy skolem-2))

Now expression {6} is only approximately correct because if the expression: *(is-spy skolem-3)* resided in Ralph's data base, then it would be sufficient to make expression {6} true, which is undesireable because {6} would be true when Ralph did not have someone specific in mind.

We assume the agent is sensitive to whether a term is a skolem constant and realizes when it is using a skolem constant. With a skolem constant, there is no prospect for the agent to perform any sort of semantic attachment in order to gather more information about the nature of the entity refered to by the constant. This is in contrast to, for instance, perceptually-based terms that might be generated by looking at an entity. In that circumstance the agent would know of the possibility of acquiring more information by looking again. The act of looking would be a form of semantic attachment (cf., Weyrauch, 1980).

We can fix {6} as follows:

{7} (and
(believes-that Ralph (list 'is-spy skolem-2))
(is-not-sk-const skolem-2))

Expression {7} now stipulates that whatever skolem-2 refers to, the referent must not itself be a skolem constant.

There is an interpretation where the listener has not disambiguated between the specific and non-specific interpretation. In fact, humans often do not notice the ambiguity of sentence {1}. We can represent this by expression {8} where we are uncommitted about the nature of the designator Ralph is using.

{8} (believes-that Ralph (list 'is-spy skolem-2))

We then have the option of later enriching the representation with an assertion saying whether or not the object referred to by "skolem-2" is a skolem constant. Asserting that it is a skolem constant gives us the non-specific interpretation. Asserting that it is not a skolem constant gives us the specific interpretation. In summary, a representation which does not distinguish between the specific and non-specific interpretation of sentence {1} is given in expression {8}. Both the specific and non-specific interpretations can represented by incrementally augmenting the knowledge base.

## Correspondence of Mental Representations.

**Relative Denotation.** Different agents may take the same designator in their respective mental languages to denote different entities. To express this kind of knowledge, we will introduce a function called "relative-den" which accepts two arguments, an agent and a designator.[1] The instantiated function term denotes what the agent takes the designator to denote. For example, the expression: *(relative-den John 'Fido)* denotes whatever John thinks "Fido" denotes.[2] The observer can represent that he and John agree about what "Fido" denotes by the expression:

{9} (= Fido (relative-den John 'Fido))

We will call assertions like expression {9} assertions of *correspondence*.[3]

**Semantics of Relative Denotation.** We give a semantics for *relative denotation* by use of a hierarchy of models. Having equated the explicit beliefs of an agent with a set of sentences in the agent's mental language, we can treat an agent's belief set as a logical theory in QPC for which we can supply a *model*. When an agent maintains a mental designator, we can speak of its denotation by virtue of its causal connections (e.g., perceptual-motor and behavioral) to the world. Further, if an agent has two mental designators whose causal connections indicate that they are used to denote the same entity, then it seems reasonable to say that the agent believes that the designators corefer. Were this not the case, the agent would not have beliefs *about* anything.

An example is illustrated in Figure 1. In the area enclosed by the box labelled "model-1," we have two distinct ordinary dogs. The box labelled "Ralph's theory" contains part of Ralph's belief set. In particular, Ralph has two distinct representations which by virtue of their causal connections to the world, denote the same entity. Consequently, it follows that he believes that the designators (i.e., representations) corefer.[4]

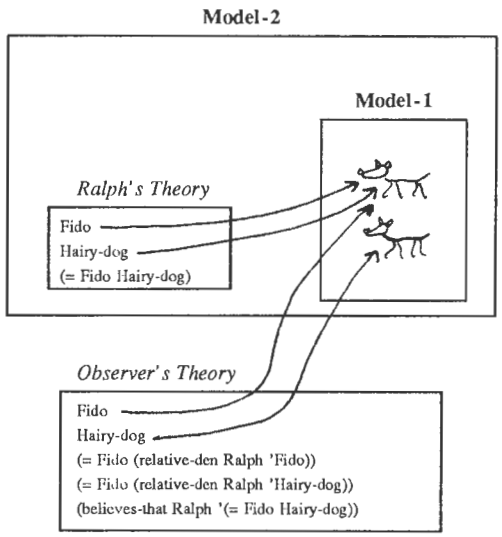



**Figure 1. A domain and interpretation illustrating the use of the "relative-den" function symbol.**

An *observer* of Ralph also has a theory. His theory, by virtue of its causal connections, is about both Ralph's mental structure and the ordinary world that Ralph has beliefs about (i.e., the contents in the box labelled "model-2"). The observer uses the designator "Fido" to denote the same entity that Ralph uses the designator "Fido" to denote. Furthermore, the observer believes this, as indicated by the correspondence assertion *(= Fido (relative-den Ralph 'Fido))* that exists in the observer's theory. Notice that the observer uses the designator "Hairy-dog" to denote a different entity than Ralph does. These considerations give the following identity.

```
(forall (x y)
   (≡ (= (relative-den Ralph x) (relative-den Ralph y))
      (believes-that Ralph '(= ,x ,y))))
```

**De Re Beliefs.** We turn to a more complicated variant of {1}, shown in {10}.

{10} Ralph believes that the Queen of Thebes is a spy.

This sentence has two interesting interpretations deriving from the way the phrase "the Queen of Thebes" is interpreted. In the first case (the *de re* interpretation), Ralph believes that some person (who, possibly unknown to Ralph, happens to be the Queen of Thebes) is a spy. In the

[1] The decision to use *relative denotation* emerged during discussions with Joe Niederberger.

[2] If the designator "Fido" does not reside in John's data base, then the instantiated term cannot designate anything. In this case, we will say that it designates the distinguished object NOTHING.

[3] Various apparatus has been used to express similar notions. Creary (1979), and later Barnden (1986), used a "concept-of" predicate. Martin (1979) used co-descriptors. Rapaport and Shapiro (1984) asserted extensional equivalence across belief contexts. Smith (1986) talks about the more general notion of circumstantial relativity. The use of *relative denotation* differs from the other proposals in that it remains in the confines of denotational semantics.

[4] Just as in a logical theory, we would say that A equals B relative to a model if and only if A and B denote the same entity in the model (cf., Genesereth & Nilsson, 1987, p. 25). The interpretation function would have to be wired into Ralph's perceptual/motor system.

second case (the *de dicto* interpretation), the speaker believes that Ralph believes that the person in question is indeed the Queen of Thebes and is indeed a spy. We will only discuss the former interpretation because the latter is well understood.

**Truth Conditions of De Re Ascriptions.** The *de re* interpretation might be intended after Ralph sees a person walking into restricted areas of a castle while not realizing that this person is the Queen of Thebes. Given this, the truth conditions of the *de re* sentence would be:

```
{11} (exists (desg)
       (and (believes-that Ralph (list 'is-spy desg))
         (= The-Q-of-Thebes (relative-den Ralph desg))))
```

An equivalent to this sentence has not been represented by practitioners of the quoted-language syntactic approach (i.e., Haas, 1986; Perlis, 1985).

Moore and Hendrix (1979) stated the truth conditions of the *de re* interpretation to a belief sentence. However, they did not show how the truth conditions translate into a knowledge representation formalism. Let us look at their statement.

> A sentence of the form "A believes S" is true if and only if the individual denoted by "A" has in his belief set a formula P that meets the following two conditions: first, the subexpressions of "S" that are interpreted *de dicto* must express the meaning for him of the corresponding subexpressions of P; second, the subexpressions of "S" that are interpreted *de re* must have the reference for him of the corresponding subexpressions of P, and he must be able to pick out the reference of P (p. 19).

In our case, the agent is Ralph, S is "the Queens of Thebes is a spy," the subexpression of S that is interpreted *de dicto* are the words "is a spy," and the subexpression of S that is interpreted *de re* is the words "the Queens of Thebes." P is expression {11}. When Moore and Hendrix use the phrase "reference for him," we use the relative denotation function. Where Moore and Hendrix use the phrase "be able to pick out the reference," we have no corresponding technical device. Moore and Hendrix distinguish between the meaning and reference of an expression. We take the meaning of an expression for Ralph to be a copy of the actual expression Ralph uses.

Expression {11} contains two parts. The subexpression *(believes-that Ralph (list 'is-spy desg))* represents the portion of the sentence which should be taken *de dicto* and attempts to actually describe the expression Ralph uses. The subexpression *(= The-Q-of-Thebes (relative-den Ralph desg))* encodes the *de re* portion of S and asserts, as Moore and Hendrix stipulate, that the phrase "the Queen of Thebes" has the reference for the observer of the corresponding expression of P, i.e., desg.

## Belief Nesting.

We want to represent that some observer believes that Ralph believes (specifically) that there is a spy. The observer's data base would contain the following formulas.

```
{15} (believes-that Ralph (list 'is-spy skolem-2))
{16} (is-not-sk-const skolem-2)
```

We, as observer of the observer (Observer Two), must describe the state of the observer's data base. This is done below.

```
{17} (believes-that observer
       '(believes-that Ralph '(is-spy ,skolem-3)))

{18} (is-sk-const skolem-3)

{19} (believes-that observer
       '(is-not-sk-const (relative-den Ralph ,skolem-3)))

{20} (= Ralph (relative-den observer 'Ralph))
```

Expressions {17} and {18} state that expression {15} resides in the observer's data base. Expression {19} states that expression {16} resides in the observer's data base. "Skolem-3" in expressions {17}-{19} is used to denote "skolem-2" of expression {15} and {16}. That is why in expression {18} we assert that "skolem-3" denotes a skolem constant. However, Ralph uses "skolem-2" to denote something that is not a skolem constant. That is why, in {19}, we say that the denotation for Ralph of what we denote by "skolem-3" (i.e., "skolem-2") is not a skolem constant. Finally, {20} asserts that Observer Two agrees with the original observer on what "Ralph" denotes.

## Mental Correspondence.

We now return to the Geach sentence armed with tools for treating non-specificity of belief, correspondence across belief contexts, and belief nesting. We simplify the sentence to remove verbs that are beyond the scope of this paper. This gives us {21}.

> {21} Hob believes (non-specifically) that a witch killed the horse and Nob believes that the (same) witch killed the pig.

The observer has beliefs about two people, Hob and Nob. This means we must specify the state of Hob and Nob's data bases prior to specifying the state of the observer's data base. Specifying Hob's data base is based on analogy with {8}.

```
(believes-that Hob (list 'kills skolem-1 'horse))
(is-sk-const skolem-1)
(believes-that Hob (list 'is-witch skolem-1))
```

These are the observer's beliefs about Hob's beliefs. The middle expression can be left out if we do not commit ourselves to whether the belief is specific or non-specific. Since the observer believes that Nob believes that the very same (non-specific!) witch killed the pig, this entails that Nob believes that Hob believes (non-specifically) that a witch killed the horse. These can be expressed by analogy with the previous section as shown below. It is reminiscent of pushing down environments, described in Wilks & Bein (1983) and Wilks & Ballim (1987).

(believes-that Nob '(believes-that Hob '(kills ,skolem-2 horse)))

(is-sk-const skolem-2)

(believes-that Nob '(is-not-sk-const (relative-den Hob ,skolem-2)))

(= Hob (relative-den Nob 'Hob))

Nob independently believes about some object that it killed a pig and is a witch. This is expressed below.

(believes-that Nob (list 'kills skolem-3 'pig))

(believes-that Nob (list 'is-witch skolem-3))

Furthermore, Nob believes a correspondence ({22}) and the observer believes a correspondence ({23}).

{22} (believes-that Nob '(= ,skolem-3 (relative-den Hob ,skolem-2)))

{23} (= (relative-den Hob skolem-1) (relative-den Nob skolem-3))

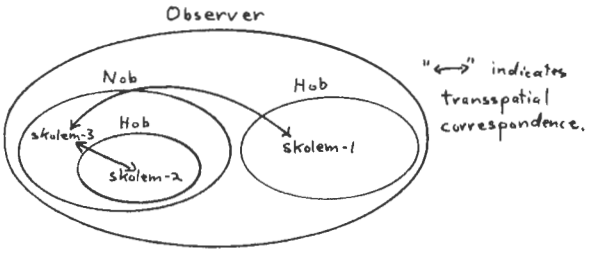



**Figure 2. The correspondences for the Geach sentence, expressions {22} and {23}.**

## Discussion.

Since the problems of resolving referents in natural language — *definite descriptions, anaphoric pronouns, and the like* — are inherently problems of determining mental correspondence, the approach seems applicable to those circumstances as well. Very little, however, has been said about the semantics of these skolem constants. For this paper, we assume that their semantics is identical to those of normal terms and their non-specific nature is based on the inability to use them in procedural attachment. In our current work, we have come to believe that mental representations can not be denotational in a model-theoretic sense. However, that is a topic of a future paper.

## References


[1] Barnden, J. A. (1986) Imputations and explications: Representational problems in treatments of propositional attitudes. *Cognitive Science,* 10, 319-364.

[2] Creary, L. G. (1979) Propositional attitudes: Fregean representation and simulative reasoning. *Proc. IJCAI,* 6, 176-181.

[3] Cresswell, M. J. (1985) *Structured meanings: The semantics of propositional attitudes.* Cambridge, MA: MIT Press.

[4] Fauconnier, G. (1985) *Mental spaces: aspects of meaning construction in natural language.* Cambridge: MIT Press.

[5] Geach, P.T. (1967) Intentional identity. *The Journal of Philosophy,* 64, 627-632.

[6] Genesereth, M. & Nilsson, N. (1987) *Logical foundations of artificial intelligence.* Los Altos, CA: Morgan Kaufman.

[7] Hass, A. R. (1986) A syntactic theory of belief and action. *Artificial Intelligence,* 28 (3), 245-292.

[8] Kaplan, D. (1971) Quantifying in. In L. Linsky (Ed.) *Reference and modality.* Oxford: Oxford University Press.

[9] Konolige, K. (1982) A first-order formalization of knowledge and action for a multi-agent planning system. In J.E. Hayes, D. Michie, & Y.H. Pao (Eds.) *Machine Intelligence 10,* pp. 120-147, Chinchester, England: Ellis Horwood.

[10] Maida, A. (1983) Knowing intensional individuals and reasoning about knowing intensional individuals. *Proc. IJCAI,* 8, 382-384.

[11] Maida, A. (1985) Selecting a humanly understandable knowledge representation for reasoning about knowledge. *International Journal of Man-Machine Studies,* 22, 151-161.

[12] Maida, A. (1986) Introspection and reasoning about the beliefs of other agents. *The Eighth Annual Conference of the Cognitive Science Society,* Amherst, MA, 187-195.

[13] Martin, W. A. (1979) Roles, co-descriptors, and the formal representation of quantified English expressions. Technical Report No. MIT/LCS/TM-139, 1979, 545 Technology Square, Cambridge, Massachusetts 02139.

[14] Moore, R. C. & Hendrix, G. G. (1979) Computational models of belief and the semantics of belief sentences. SRI Artificial Intelligence Center *Technical Note 187,* SRI International, Menlo Park, CA.

[15] Niederberger, J. (1987) A syntactic approach to Kripke's "A puzzle about belief." Master's paper, Department of Computer Science, Penn State University, University Park, PA 16802.

[16] Pendlebury, M. (1982) Hob, Nob, and Hecate: The problem of quantifying out. *Australasian Journal of Philosopy,* 60, 346-354.

[17] Perlis, D. (1985) Languages with self-reference I: Foundations (or We can have everything in first-order logic!). *Artificial Intelligence,* 25 (3), 301-322.

[18] Perlis, D. (1986) Self-reference, knowledge, belief, and modality. *Proc. AAAI-86,* Vol. 1, 416-420.

[19] Quine, W. V. O. (1956) Quantifiers and propositional attitudes. *Journal of Philosophy,* 1956, 53, 177-187.

[20] Rapaport, W. J. & Shapiro, S. C. (1984) Quasi-indexical reference in propositional semantic network. *Proceedings of the Tenth International Conference on Computational Linguistics,* Stanford, California, July 2-6, 1984, pp. 65-70.

[21] Smith, B. C. (1986) Varieties of self-reference. In J. Halpern (Ed.) *Theoretical aspects of reasoning about knowledge* (pp. 19-43). Los Altos, CA: Morgan Kaufmann.

[22] Van Lehn, K. (1978) Determining the scope of English quantifiers. MIT AI Lab Report AI-TR-483.

[23] Weyrauch, R. (1980) Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence, 13,* 133-170.

[24] Wilks, Y. & Ballim, A. (1987) Multiple Agents and the Heuristic Ascription of Belief. In *Proceedings of IJCAI-87,* Milan, Italy, 118-124.

[25] Wilks, Y. & Bien, J. (1983) Beliefs, points of view, and multiple environments. *Cognitive Science,* 7, 95-119.

# Statistically Founded Degrees of Belief

Fahiem Bacchus*
Department of Computing Science
The University of Alberta
Edmonton, Alberta
T6G 2H1

## Abstract

A logic, called **Lp**, is developed which can express a large variety of statistical knowledge. This logic takes a novel approach to the problem of mixing probabilities with first order logic. In the logic there is a probability distribution over the domain of discourse, while in previous work, e.g., Nilsson's probability logic, the probability distribution is over the set of possible worlds. It is shown how logics with a probability distribution over the set of possible worlds are incapable of expressing statistical knowledge, e.g., "The majority of birds can fly," whereas, the logic developed here can. It is shown how this statistical knowledge can be used to induce a degree of belief in sentences which make reference to specific individuals, e.g., "Tweety can fly." These degrees of belief are generated through a simple and intuitive inductive assumption. The induced degrees of belief display non-monotonic properties and provide an alternative formalism for expressing and reasoning with notions of statistical typicality. The formalism has the advantage of possessing a transparent semantics, based on sets and probabilities of those sets, as well as a sound and complete syntactic proof theory.

## 1 Introduction

The main contribution of this work is the development of a formal logic capable of representing, and reasoning with, a wide variety of statistical knowledge The logic, called **Lp**, is a type of probability logic. It is, however, very different from most previously developed probability logics. Whereas most previous probability logics posit a probability distribution over the set of possible worlds, in **Lp** the probability distribution is over the domain of discourse. This allows **Lp** to express statistical knowledge inexpressible in the possible worlds approach. The logic is an extension of ordinary first order logic, and it possesses a sound and complete proof theory which can reason not only with formulas of first order logic but also with the statistical knowledge.

**Lp** is, however, incapable of assigning probabilities to closed formulas (i.e., formulas in which all of the variables are bound, also called sentences). It will be demonstrated how the statistical knowledge expressed in **Lp** can be used to generate degrees of belief for a broad class of closed formulas. This is accomplished through a simple and intuitive inductive assumption. These induced degrees of belief display non-monotonic behaviour, i.e., they can change (drastically) with the *addition* of new knowledge.

By itself **Lp** is capable of representing the kind of poorly quantified statistical knowledge typically found in diagnostic domains, e.g., the domain of medical diagnosis. It is also capable of reasoning with this knowledge in a sound and complete manner. In fact its reasoning abilities logically subsume all previous probability based reasoners. Furthermore, it possesses a solid semantic foundation.

The combination of **Lp** and the inductive generation of degrees of belief offers an alternative to the default logic of Reiter [1] in those situations where a statistical interpretation of the default is available. In particular, multiple inheritance hierarchies with exceptions can be given a natural treatment with this formalism [2]. Furthermore, because the generated degrees of belief are based on probabilities, they can be used for decision theoretic planning in those situations where costs are available (see, e.g., Luce [3]), taking advantage of strong results due to De Finetti and others (see [4, chapters 3 and 7]) which show that measures over a propositional lattice of sentences used to guide decisions *must be probability measures* if the decision maker is to secure the possibility of a net gain (or at least 'breaking even') in all decision-making situations.

Section 2 demonstrates the difference between the possible worlds approach and **Lp**, presents the intuition behind the induction of degrees of belief, and discusses the expressiveness of **Lp**. Section 3 gives, in a condensed form, the formal details of **Lp**, including the deductive proof theory. Section 4 does the same for the inductive mechanism. Section 5 consists of some examples of the types of reasoning possible. Finally, section 6 makes some conclusions and discusses some open problems.

## 2 Motivation

### 2.1 Lp

Most previous work on probability logics in AI, [5,6], and in philosophy, [7,8,9,10,11,12], that the author is aware of, has posited a probability distribution over the set of possible worlds, where a possible world in this context is considered

*This research was supported by the University of Alberta through their Dissertation Fellowship

to be one possible complete specification of the truth values of the sentences of the logic.[1] In this approach the probability of any sentence becomes the measure of the set of possible worlds in which that sentence is true. This leads to an essential difficulty when trying to express statements about the proportion of objects which possess a certain property, as for example, in the statement "More than 50% of all dogs bark." This statement makes a claim about dogs in general, and in a first order language the only plausible means of representing it is by assigning a probability to the universal sentence $\forall x Dog(x) \rightarrow Bark(x)$. However, if the knowledge base contains a *single* instance of an individual dog who cannot bark, then the universal sentence will be false in all possible worlds; thus, the probability of the universal sentence will be *zero*, even if every other individual dog in the knowledge base is known to bark.

In contrast, in **Lp** the probability distribution is over the domain of discourse. This approach allows the expression of statistical knowledge through probability terms which contain open formulas (i.e., formulas with free variables). For example, the previous statement can be expressed with the **Lp** sentence $[Bark(x)|Dog(x)]_x > .5$. This sentence is formed from the '>' predicate symbol, the constant '.5', and a probability term which contains two open formulas, $Bark(x)$ and $Dog(x)$. Intuitively, the probability term represents the proportion of dogs, $x$, which bark. These probability terms have a completely different semantics from the semantics of ordinary universal sentences and can be used to express a wide variety of statistical knowledge.

A key innovation which contributed to the expressiveness of the logic was to make the logic two sorted, by including a totally ordered field of numbers in the semantics. One sort of entity in the logic is the set of objects $\mathcal{O}$, and the other sort is a field of numbers. The intention is that the set of objects consists of things of interest (e.g., cars, people, kinds of cars, etc.), while the field of numbers are the denotations of the probability terms.

Probabilistic knowledge is encoded in **Lp** through the formation of probability terms from open formulas. These terms are field terms, i.e., they refer to particular, but unspecified, numbers in the field. Symbols representing ordering relations and field functions are also included; so, sentences can be formed from a mixture of these symbols and the probability terms. Sentences formed in this manner can be very expressive. For example, knowledge like "It is more likely that a politician is a lawyer than an engineer" can be expressed with an **Lp** sentence formed with the less than field predicate symbol, '<'.

The two sortedness of the logic also allows the creation of 'measuring' functions which map from the set of objects to the field of numbers. These functions can be viewed as being metrics which are applicable to the objects. For example, one could say $Weight\_in\_kgs(Jack) = 80$ to indicate that the measuring function $Weight\_in\_kgs$ maps the object $Jack$ to the number 80, the obvious interpretation being that Jack weighs 80 kgs. With these measuring functions it is possible to express functional probabilistic information like "The more a bird weighs the less likely it is that it will be able to fly."

## 2.2 Belief Formation

**Lp** can express statistical knowledge, but no probability can be assigned to a closed formula like "$Bark(Fido)$." In the logic closed formulas are either true or false. On the other hand, the possible worlds approach can assign probabilities to closed formulas, but is incapable of expressing statistical knowledge. Hence, these two approaches are in a sense two parts of a complete picture.

The statistical knowledge in **Lp** can be used to generate degrees of belief, which can be viewed as assignments of probability, for a broad class of closed formulas. These statistically induced degrees of belief have an advantage over the subjective probabilities representable in the possible worlds approach. The advantage is that they are founded on objective information about the world, information which could in principle be accumulated by a rational agent through its experience with the world.

The degrees of belief are generated through a simple inductive assumption which has a long history. It is similar to the way in which we make sense of statements like "the probability that a coin will show heads when flipped is 0.5." For a particular instance of flipping a coin it is necessarily the case that the coin will show either heads or tails, i.e., the truth value of *Show_heads* will be either zero or one. When we state that the probability of *Show_heads* is 0.5 we are randomizing the particular coin. In other words, we know that 50% of the instances of flipping coins yield heads (assuming that there are as many coins biased to heads as to tails), and since we do not have any information that distinguishes this particular coin, it is reasonable to believe *Show_heads* to degree 0.5. Similarly for formulas like "$Bark(Fido)$" where $Fido$ is a dog, if it is known that (say) 90% of all dogs bark and all that is known about $Fido$ is that he is a dog, then the inductive assumption would impart a degree of belief of 0.9 to the (closed) formula "$Bark(Fido)$" by assuming that $Fido$ was a randomly selected dog.

Inductive assumptions of randomization have appeared before in the philosophy of science literature, at least as early as Reichenbach (1949 [13]) and more recently in work

[1] Work in philosophy posits a probability distribution over the propositional lattice formed by the equivalence classes of the sentences in the logic. The bases for this probability distribution are sentences which are long conjunctions and which fix the truth value of all other sentences. Corresponding to each such long conjunction is a possible world—the long conjunction specifies the truth values in that possible world. Hence, a probability distribution over these basis sentences is equivalent to a probability distribution over the set of possible worlds. When the logic is first order logic universally quantified sentences are assigned a probability which is equal to the product of the probabilities of all the instantiations of that universal sentence. This is called the substitutional interpretation (see LeBlanc [11]). (Technical details differ from author to author). In fact, this is the only reasonable interpretation if one also wishes to preserve the normal semantic meaning of universal sentences. When using the substitutional interpretation one false instantiation (i.e., an instantiation with probability zero) will force the probability of the universal to be zero. For example if we know that *Tweety* is a bird who cannot fly, i.e., $Fly(Tweety)$ has probability zero, then we must also have that the probability of $\forall x Bird(x) \rightarrow Fly(x)$ is zero (cf. the possible worlds approach described in body of text).

by Kyburg [14,15]. Similar inductive assumptions have also appeared *implicitly* in most of the expert systems which deal with uncertainty. For example, in the MYCIN system most of the rules which have certainty factors are in the form "The certainty of infection D given symptoms A, B and C is $x$." Here an implicit randomization is taking place over the space of patients. When diagnosis is performed on a particular patient it is assumed that these certainties are applicable to that patient; this is identical to an inductive assumption of randomization.

The inductive assumption must deal with situations where there is conflicting knowledge. The example of Fido the barking dog can be used to illustrate this point. The inductive assumption generates a single degree of belief for the sentence "$Bark(Fido)$" only when all that is known about Fido is that he is a dog (i.e., all that is deducible about Fido from the knowledge base using **Lp** deduction). This situation is rare; usually much more is known about named individuals. For example, the sentences "$Dingo(Fido)$" or "$Black(Fido)$" may also be in the knowledge base. In general, the degree of belief in "$Bark(Fido)$" induced from the knowledge that Fido is a dog will be completely different from the degree of belief induced from, say, the knowledge that Fido is a dingo. That is, considering Fido to be a randomly selected dog yields a different degree of belief than when Fido is considered to be a randomly selected dingo (dingos don't bark). Thus, the knowledge base can generate a range of different degrees of belief for any sentence, dependent on what knowledge is used in the inductive step of randomization. These different degrees may conflict and there may be no reason to choose one over the other. This problem has a long history and is known as the problem of choosing the proper reference class (see Kyburg [15]). There is, however, a natural preference criterion which can be applied in many situations.

This preference criterion is based on the simple intuition that the more knowledge that is used to generate the degree of belief the better is that degree of belief. More knowledge has a simple interpretation in **Lp**; i.e., the sentence $\alpha$ represents more knowledge than $\beta$ if $\alpha \rightarrow \beta$ is deducible from the knowledge base.[2] For example, the fact that $Dingo(Fido) \rightarrow Dog(Fido)$ (as dingos are a subset of the set of dogs) indicates that the knowledge $Dingo(Fido)$ should be preferred when inducing a degree of belief in $Bark(Fido)$.

# 3 The Logic Lp

Due to space limitations all proofs are omitted; also, to simplify the presentation, only models with finite sets of objects are considered. For the proofs as well as a treatment which allows infinite domains of discourse see Bacchus [16]. This longer version also gives a more detailed exposition of everything discussed here.

The letters $n$ and $m$ are used as meta-linguistic variables denoting natural numbers.

## 3.1 Symbols

We start with a denumerable set of symbols. This set includes symbols which denote constants, functions of any arity, predicates of any arity, and variables. The constants and predicates can be of two types, either field or object. (When there is a danger of confusion the field symbols will be written in a **bold** font). The function symbols come in three different types: object, field and measuring functions. The measuring functions will usually have special names like *Weight* or *Size*.

Along with these symbols we also have a set of distinguished symbols, symbols whose interpretation is fixed. There are the following field symbols: the constants 1 and 0, the binary predicates $=$ and $\geq$, and the binary function symbols $+$, $-$, $\times$, and $\div$. The symbol $=$ is also used to represent the object equality predicate.

Finally, we have the logical connective $\wedge$, the quantifier $\forall$, and the probability term former $[o]$.

## 3.2 Formulas

The major difference between formulas in **Lp** and in first order logic is the manner in which terms are built up.

**T0)** A single object variable or constant is an *o-term*; a single field variable or constant is an *f-term*.

**T1)** If $f$ is an n-ary object (field) function symbol and $t_1, \ldots, t_n$ are o-terms (f-terms) then $ft_1 \ldots t_n$ is an *o-term* (*f-term*). If $\nu$ is an n-ary measuring function symbol and $t_1, \ldots, t_n$ are o-terms then $\nu t_1 \ldots t_n$ is an *f-term*.

**T2)** If $\alpha$ is a formula and $\vec{x}$ is a vector of $n$ object variables, $\langle x_1, \ldots, x_n \rangle$, then $[\alpha]_{\vec{x}}$ is an *f-term*.

The formulas of **Lp** are built up in the standard manner, with the added constraint that predicates can only apply to terms of the same type. A notable difference with first order logic is that f-terms can be generated from formulas by the probability term former.

The connectives $\vee$ and $\rightarrow$, and the quantifier $\exists$ are defined in the standard manner from the given primitives. It is also convenient to define an extended set of field inequality predicates, $\leq$, $<$, $>$, and $\in$ (denoting membership in an interval), from the primitive $\geq$. The predicate symbols $=$ and $\geq$ as well as the function symbols $+$, $\times$, $-$, and $\div$, are written in the more readable infix form. Furthermore, standard conventions of scope and precedence are used to limit the use of parentheses.

Conditional probabilities are represented in **Lp** with the following abbreviation.

**Definition 3.1**

$$[\alpha|\beta]_{\vec{x}} =_{df} [\alpha \wedge \beta]_{\vec{x}} \div [\beta]_{\vec{x}},$$

*if* $[\beta]_{\vec{x}} \neq 0$*; otherwise, it is undefined.*

[2] In general, it is undecidable in first order logic (and thus in **Lp** which is an extension) as to whether or not $\alpha \rightarrow \beta$ is deducible from the knowledge base. One can, however, always be conservative and assume that $\alpha \rightarrow \beta$ is *not* deducible if a deduction is not found before some resource limit is exceeded. By assuming that no deduction exists one is forced to consider both degrees of belief.

## 3.3 Examples of Representation

1. Notions of typicality, e.g., "Most birds can fly:"

$$[fly(x)|bird(x)]_x > \mathbf{c},$$

where '**c**' is some field constant in the open interval $(0.5, 1)$. Thus, we avoid specific numbers. **c** has a specific denotation (i.e., the percentage of flying birds has a specific value), but we do not have access to it (i.e., we don't what the percentage is, all that we know are some crude bounds).

2. Functional probabilistic relations, e.g., "Heavier birds are less likely to be able to fly:"

$$\forall \mathbf{y}([fly(x)|bird(x) \wedge weight(x) < \mathbf{y}]_x > [fly(x)|bird(x) \wedge weight(x) > \mathbf{y}]_x).$$

3. Mixing universal quantification and probabilities, e.g., "The probability of finding a given type of animal at a zoo is a function of the expense of acquiring and maintaining that type of animal:"

$$\forall x(animal\_type(x) \rightarrow [at(x,y)|zoo(y)]_y = \mathbf{f}(expense(x))),$$

where *expense* is a measuring function symbol and **f** is a field function symbol. **f** could be declared to be non-decreasing:

$$\forall \mathbf{xy}(\mathbf{x} > \mathbf{y} \rightarrow \mathbf{f}(\mathbf{x}) > \mathbf{f}(\mathbf{y})).$$

Thus, even if we do not know **f**'s denotation we could still reason about different values of **f**.

4. Knowledge of independence:

$$[P(x) \wedge Q(x)|R(x)]_x = [P(x)|R(x)]_x \times [Q(x)|R(x)]_x.$$

Thus we can represent finely grained notions of independence and are not stuck with global assumptions.

5. Notions from Statistics, e.g, "The height of adult male humans is normally distributed with mean 177cm and standard deviation 13cm:"

$$\forall \mathbf{xy}([height(x) \in (\mathbf{x}, \mathbf{y})|Adult\_male(x)]_x = \mathbf{normal}(\mathbf{x}, \mathbf{y}, 177, 13)).$$

Here **normal** is a field function which, given an interval $(\mathbf{x}, \mathbf{y})$[3], a mean, and a standard deviation, returns the (rational number approximation of the) integral of a normal distribution, with specified mean and standard deviation, over the given interval.

[3]One would probably want to constrain the values of **x** and **y** further, for example, $\mathbf{x} < \mathbf{y}$. Also, in example 2, **y** could be constrained to keep it in some reasonable bounds, e.g., non-negative.

## 3.4 Semantic Model

An **Lp**-Structure is defined to be the tuple $\mathcal{M}$:

$$\langle (\mathcal{O}, R_\mathcal{O}, F_\mathcal{O}), (\mathcal{F}, R_\mathcal{F}, F_\mathcal{F}), \Psi, \{ \mu_n \mid n = 1, 2 \ldots \} \rangle$$

Where:

**a)** $(\mathcal{O}, R_\mathcal{O}, F_\mathcal{O})$ represents a finite (see [16]) for infinite domains) set of individual objects $\mathcal{O}$, a set of relations, $R_\mathcal{O}$, and a set of functions $F_\mathcal{O}$, both of any arity. Included in $R_\mathcal{O}$ is the equality relation.

**b)** $(\mathcal{F}, R_\mathcal{F}, F_\mathcal{F})$ similarly represents a totally ordered field of numbers along with a set of relations and functions which include the equality and greater than or equal relations as well as the field operations addition, multiplication, and their inverses. $\mathcal{F}$ contains two distinguished elements which are the units of addition (zero) and multiplication (one).

**c)** $\Psi$ represents a set of measuring functions, functions from $\mathcal{O}^n$ to $\mathcal{F}$.

**d)** $\{ \mu_n \mid n = 1, 2, \ldots \}$ is a sequence of probability functions. Each $\mu_n$ is a set function whose domain is the field of subsets of $\mathcal{O}^n$ defined by the formulas of **Lp**, whose range is $\mathcal{F}$, and which satisfies the axioms of a probability function (i.e., $\mu_n(A) >$ zero, $\mu_n(A \cup B) = \mu_n(A) + \mu_n(B)$ if $A \cap B = \emptyset$, and $\mu_n(\mathcal{O}^n) =$ one ).

The sequence of probability functions is a sequence of product measures. That is, for any two sets $A \in \mathcal{O}^n$ and $B \in \mathcal{O}^m$ and their Cartesian product $A \times B \in \mathcal{O}^{n+m}$, if $A \in domain(\mu_n)$ and $B \in domain(\mu_m)$, then $A \times B \in domain(\mu_{n+m})$ and $\mu_{n+m}(A \times B) = \mu_n(A) \times \mu_m(B)$. This constraint insures that *different* variables bound by the probability term formers behave in an independent manner.

Independence ensures that the probability terms satisfy certain conditions of coherence. For example, the order of the variables cited in the probability terms makes no difference, e.g., $[\alpha]_{x,y} = [\alpha]_{y,x}$. Universal quantification also displays this property, e.g., $\forall x \forall y \alpha = \forall y \forall x \alpha$. Another example is that the probability terms are unaffected by tautologies, e.g., $[P(x) \wedge (R(y) \vee \neg R(y))]_{\langle x,y \rangle} = [P(x)]_x$.

It should be noted that this constraint on the probability functions does not make any implicit assumptions of independence of the form commonly found in probabilistic inference engines (e.g., the independence assumptions of the Prospector system [17], see Johnson [18]). This constraint affects the values of probability terms with different variables, also, complex probability terms, e.g., $[[\alpha]_x = \mathbf{x}]_y$. (This can be also be seen from axiom (P7), presented in the next section, which expresses the constraint.) The constraint does not make any presumptions concerning the independence of formulas which contain the *same* set of probability variables. That is, in general $[\alpha \wedge \beta]_{\vec{x}} \neq [\alpha]_{\vec{x}} \times [\beta]_{\vec{x}}$. See [16] for further discussion of this point.

## 3.5 Semantics of Formulas

Meaning is given to the formulas of **Lp** in the standard manner, i.e., by defining a correspondence between the formulas and the **Lp**-Structure $\mathcal{M}$ augmented by the truth values $\top$ and $\bot$ (true and false). Such a correspondence is called an interpretation. An interpretation maps all of the symbols to appropriate entities in the **Lp**-Structure. For example, it maps every object constant symbol to an element of $\mathcal{O}$, every field predicate symbol to an element of $R_{\mathcal{F}}$, etc. It maps the distinguished symbols to the expected entities; e.g., it maps the distinguished constant 0 to the zero (unit of addition) of $\mathcal{F}$. An interpretation also gives an initial assignment to all of the variables; i.e., it maps all of the object variables to elements of $\mathcal{O}$ and all of the field variables to elements of $\mathcal{F}$.

These assignments serve as the inductive basis for an interpretation of the formulas. This interpretation is built up in the same way as in first order logic, with the added consideration that universally quantified object variables range over $\mathcal{O}$ while universally quantified field variables range over $\mathcal{F}$. The only thing which needs to be demonstrated is the semantic interpretation of the probability terms.

Let $\sigma(x/a)$ denote a new interpretation which is identical to $\sigma$ except that it assigns the individual $a$ to the variable $x$ (types must match). More generally, let $\sigma(\vec{x}/\vec{a})$, where $\vec{a} = \langle a_1, \ldots, a_n \rangle$ and $\vec{x} = \langle x_1, \ldots, x_n \rangle$ are vectors of individuals and variables (of matching type), denote a new interpretation identical to $\sigma$ except that $(x_i)^{\sigma(\vec{x}/\vec{a})} = a_i$, $(i = 1, \ldots, n)$.

The probability terms are given the following semantic interpretation:

- For the f-term $[\alpha]_{\vec{x}}$,

$$([\alpha]_{\vec{x}})^{\sigma} = \mu_n\{\vec{a} | \alpha^{\sigma(\vec{x}/\vec{a})} = \top\}.$$

Since $\mu_n$ is a probability function which maps to the field of numbers $\mathcal{F}$, it is clear that $[\alpha]_{\vec{x}}$ denotes an element of $\mathcal{F}$ under the interpretation $\sigma$; thus, it is a valid f-term. As mentioned before, the domain of $\mu_n$ is the field of subsets of $\mathcal{O}^n$ defined by the formulas of **Lp**. It can be proved that this set of subsets of $\mathcal{O}^n$ is in fact a field of subsets [16], thus showing that $\mu_n$ is a well defined probability function.

**Definition 3.2** *An interpretation* $\sigma$ satisfies *a formula* $\alpha$ *(set of formulas* $\Phi$*) if* $\alpha^{\sigma} = \top$ *(*$\beta^{\sigma} = \top$ *for every* $\beta \in \Phi$*), written* $\sigma \models \alpha$ *(*$\sigma \models \Phi$*). A set of formulas* $\Phi$ entails *a formula* $\alpha$ *(written* $\Phi \models \alpha$*) if every interpretation* $\sigma$ *which satisfies* $\Phi$ *also satisfies* $\alpha$*.*

## 3.6 Deductive Proof Theory

This section provides a deductive proof theory for **Lp**. The proof theory consists of a set of axioms and rules of inference, and can be shown to be both sound and complete. The proof theory for **Lp** is similar to the proof theory for ordinary first order logic. A major change is that in the axioms of **Lp** two new sets of axioms must be introduced, one to deal with the logic of the probability function, and another set to define the logic of the field $\mathcal{F}$.

The probability terms introduce a new way of binding variables in formulas. This invalidates all of the standard first order results on variable binding as well as those results on variable substitution. These results can, however, be reconstructed in **Lp** from new definitions which take into account the binding effects of the probability terms. The construction of these definitions along with proving their formal properties represents the majority of the work involved in the generation of a deductive proof theory for **Lp**. All of these details will be omitted. For this presentation it is sufficient to say that variable substitution in **Lp** works in a manner which is a natural extension of the way it works in first order logic.

### Axioms of Lp

If $\alpha$ is a formula of **Lp**, then a *generalization* of $\alpha$ is any formula of the form $\forall x_1 \ldots \forall x_n \alpha$, where $\{x_1, \ldots, x_n\}$ is a set of not necessarily distinct variables of either type.

### First order Axioms

All of the axioms of first order logic, see, for example, Bell [19].

### Field Axioms

All of the axioms of a totally ordered field, see, for example, Maclane [20].

### Probability Function Axioms

**P1)** $\forall x_1 \ldots \forall x_n \alpha \rightarrow [\alpha]_{\vec{x}} = 1$,
where $\vec{x} = \langle x_1, \ldots, x_n \rangle$ and every $x_i$ is an object variable.

**P2)** $[\alpha]_{\vec{x}} \geq 0$.

**P3)** $[\alpha]_{\vec{x}} + [\neg\alpha]_{\vec{x}} = 1$.

**P4)** $[\alpha]_{\vec{x}} + [\beta]_{\vec{x}} \geq [\alpha \vee \beta]_{\vec{x}}$.

**P5)** $[\alpha \wedge \beta]_{\vec{x}} = 0 \rightarrow [\alpha]_{\vec{x}} + [\beta]_{\vec{x}} = [\alpha \vee \beta]_{\vec{x}}$.

**P6)** $[\alpha]_{\vec{x}} = [\alpha(x_i/z)]_{\vec{x}(x_i/z)}$,
where $z$ is an object variable which does not appear in $\alpha$ and $\vec{x}(x_i/z)$ is the new vector of object variables $\langle x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_n \rangle$.

**P7)** $\forall \mathbf{z}_1 \mathbf{z}_2 [[\alpha]_{\vec{x}} = \mathbf{z}_1]_{\vec{y}} = \mathbf{z}_2 \rightarrow ([\alpha]_{\langle \vec{x}, \vec{y} \rangle} \geq \mathbf{z}_1 \times \mathbf{z}_2)$.

### Generalization

**G1)** All generalizations of the preceding axioms.

### Rule of inference

The only rule of inference is *modus ponens*, i.e.,

- From $\{\alpha, \alpha \rightarrow \beta\}$ infer $\beta$.

**Definition 3.3** *A deduction is a finite sequence of formulas where the formulas are either axioms, from a set of hypotheses, or inferred from earlier formulas by modus ponens. A deduction whose last formula is $\alpha$ is called a deduction of $\alpha$. $\Phi \vdash \alpha$ means there is a deduction of $\alpha$ from the set of hypotheses $\Phi$, and $\vdash \alpha$ means that there is a proof of $\alpha$. A proof is a deduction from an empty set of hypothesis, i.e., a deduction which just uses the axioms.*

The above axioms comprise a sound and complete proof theory when $\mathcal{O}$ is finite.

**Theorem 3.4 (Completeness)** *If $\Phi \models \alpha$, then $\Phi \vdash \alpha$.*

**Theorem 3.5 (Soundness)** *If $\Phi \vdash \alpha$, then $\Phi \models \alpha$.*

**Lemma 3.1** *The following are provable in* **Lp***:*

a) $([\alpha \rightarrow \beta]_{\vec{x}} = 1 \wedge [\beta \rightarrow \alpha]_{\vec{x}} = 1) \rightarrow [\alpha]_{\vec{x}} = [\beta]_{\vec{x}}$.

b) $[\alpha \vee \beta]_{\vec{x}} = [\alpha]_{\vec{x}} + [\beta]_{\vec{x}} - [\alpha \wedge \beta]_{\vec{x}}$.

It is also easy to prove Bayes' theorem in **Lp**.

**Theorem 3.6 (Bayes Theorem)** *The following is provable in* **Lp***:*

$$[\beta|\alpha]_{\vec{x}} = [\alpha|\beta]_{\vec{x}} \times \frac{[\beta]_{\vec{x}}}{[\alpha]_{\vec{x}}}.$$

# 4 Induction

The semantics of **Lp** allows a clear demonstration of the need for an inductive mechanism. A universally quantified formula is true for all objects. Hence, it is necessarily true for any particular object. The probability terms, however, state the proportion of objects for which a formula is true. There is no mention of which individuals satisfy the formula. The probabilities expressed in **Lp** do not apply to any particular individual. Thus, they do not apply to closed formula which mention particular individuals.

This section presents an inductive mechanism of belief formation which can use the general, non-specific statistical information expressed in **Lp** to generate degrees of belief in closed formulas. When the truth value of the formula is not entailed by the knowledge base, the mechanism is capable of generating graded degrees in the range 0–1; when the truth value is entailed, the mechanism can assign a degree representative of the entailed truth value, i.e., 0 or 1.

First, we define a new set of formulas $\{\mathcal{B}(\alpha|\beta)\}$, where $\alpha$ and $\beta$ are closed formulas of **Lp**. Intuitively these formulas represent the belief in $\alpha$ given the knowledge $\beta$. No formal semantics are, however, given for these formulas. Nor is any logic (with connectives) based on these formulas presented. In the context of this work the meaning of these formula is imparted through the inductive evaluation function defined below. This function assigns a degree to these belief formulas.

Denote by $\alpha(\vec{c}/\vec{x})$ the new formula which results from replacing every occurrence of $c_i$ with $x_i$ in the formula $\alpha$. Also, let $KB$ denote the knowledge base.

**Definition 4.1 (Inductive Evaluation Function)** *Given that a closed formula, $\alpha$, contains the vector of object constants $\vec{c}$ (and no other object constants), the belief formula $\mathcal{B}(\alpha|\beta)$ is assigned a degree equal to the following* **Lp** *probability term:*

$$degree(\mathcal{B}(\alpha|\beta)) = [\alpha(\vec{c}/\vec{x})|\beta(\vec{c}/\vec{x})]_{\vec{x}},$$

*where $\vec{x}$ is a vector of object variables which do not occur in $\alpha$ or $\beta$.*

The intuitive interpretation of this evaluation function is simple. We are saying the degree of belief in $\alpha(\vec{c})$ given $\beta(\vec{c})$, should be equal to the extent that a random tuple $\vec{x}$, with all the properties $\beta$ given for $\vec{c}$, is likely to have properties $\alpha$. So for example, the degree of belief in $Bark(Fido)$ given the knowledge $Dog(Fido)$ will be equal to the probability that a random dog $x$ can bark (i.e., the probability that a random object with the property $Dog$ also possesses the property $Bark$).

If we are interested in the sentence $\alpha$ there may be many different belief formulas, $\mathcal{B}(\alpha|\beta)$, about $\alpha$, each based on different knowledge $\beta$. In general the knowledge base will contain information about the degrees of various of these formulas, and these degrees may be conflicting. For example, the degree assigned to $\mathcal{B}(\alpha|\beta)$ can be very different from the degree assigned to $\mathcal{B}(\alpha|\delta)$. If the assertion of interest is $\alpha$ it may be impossible to choose between these competing beliefs.

The intuitive interpretation of $\mathcal{B}(\alpha|\beta)$ gives, however, a natural preference criterion which can in many cases decide which belief is better—beliefs based on more knowledge are to be preferred. This yields the following preference criterion:

**Definition 4.2 (Preference Criterion)** *The belief $\mathcal{B}(\alpha|\beta)$ is to be preferred to the belief $\mathcal{B}(\alpha|\delta)$, written*

$$\mathcal{B}(\alpha|\beta) >> \mathcal{B}(\alpha|\delta),$$

*if $KB \vdash \forall\vec{x}\beta(\vec{c}/\vec{x}) \rightarrow \delta(\vec{c}/\vec{x})$. Also, we say that a belief $\mathcal{B}(\alpha|\beta)$ is* well founded *if $KB \vdash \beta$.*

If the truth value of $\alpha$ is entailed, i.e., $KB \vdash \alpha$ or $KB \vdash \neg\alpha$, then for any belief $\mathcal{B}(\alpha|\beta)$ the belief $\mathcal{B}(\alpha|\beta \wedge (\neg)\ \alpha)$ is to be preferred. If $KB \vdash \alpha$ then $\mathcal{B}(\alpha|\beta \wedge \alpha)$ is well founded and has degree 1. Similarly, if $KB \vdash \neg\alpha$ then $\mathcal{B}(\alpha|\beta \wedge \neg\alpha)$ is well founded with degree 0. That is, when the truth value of $\alpha$ is entailed the degree of the most preferred, well founded belief about $\alpha$ is representative of its entailed truth value.

It has long been noted in AI that probabilities display non-monotonic behaviour ([21], [22], [23]), and in fact, the preference criterion allows for non-monotonic behaviour. If a new sentence $\delta$ is added to $KB$ (representing an increase in knowledge), then new preferred beliefs may be formed based on $\delta$. These new beliefs may have degrees which are completely different from the degrees of the beliefs they superseded. (See the example on inheritance).

The preference criterion can be given two simple justifications. First, if $\forall\vec{x}\beta(\vec{c}/\vec{x}) \rightarrow \delta(\vec{c}/\vec{x})$ then $\beta \leftrightarrow \beta \wedge \delta$. Thus, it

can be shown that the degree of $\mathcal{B}(\alpha|\beta)$ is equivalent to the degree of $\mathcal{B}(\alpha|\beta \wedge \delta)$. Hence, $\mathcal{B}(\alpha|\beta)$ is equivalent to a belief founded on more knowledge.

Semantically, when the degree of belief is assigned we are considering the constants which appear in $\alpha$ to be indistinguishable from all of the vectors which satisfy $\beta(\vec{c}/\vec{x})$. If $\forall\vec{x}\beta(\vec{c}/\vec{x}) \rightarrow \delta(\vec{c}/\vec{x})$ then it is the case that the set of vectors satisfying $\beta(\vec{c}/\vec{x})$ is included in the set of vectors satisfying $\delta(\vec{c}/\vec{x})$. Hence, we are losing less information when $\vec{c}$ is considered to be indistinguishable from the vectors which satisfy $\beta(\vec{c}/\vec{x})$ than when $\vec{c}$ is considered to be indistinguishable from the vectors which satisfy $\delta(\vec{c}/\vec{x})$, simply because, there are more vectors in the latter set.

# 5 Examples of Reasoning

**Example 5.1** *Nilsson's Probabilistic Entailment*

Nilsson [5] develops a probability logic based on the possible worlds approach. He shows how the probabilities of sentences in the logic are constrained by known probabilities, i.e., constrained by the probabilities of a base set of sentences. For example, if $[P \wedge Q] = 0.5$ then the values of $[P]$ and $[Q]$ are both constrained to be $\geq 0.5$. These constraints are in Nilsson's terms probabilistic entailments.

Nilsson gives some methods for calculating these entailments. The important point, however, is that these bounds are simply consequences of the laws of probability. In fact, the theorem

$$[\alpha \vee \beta]_{\vec{x}} = [\alpha]_{\vec{x}} + [\beta]_{\vec{x}} - [\alpha \wedge \beta]_{\vec{x}},$$

along with the fact that the probability terms are non-negative (Axiom **P2**), gives the full set of constraints from which all probabilistic entailments are derived. This theorem is true in **Lp**. And, since the proof theory of **Lp** is complete, constraints similar to Nilsson's can be deduced in **Lp**. Numerically the constraints are identical, i.e., the best bounds deducible in **Lp** are the same numbers as the best probabilistic entailments; however, the probabilities must be interpreted differently.

For example, if the base set in Nilsson's logic is $\{[P] = 0.6, [P \rightarrow Q] = 0.8\}$, probabilistic entailment gives the conclusion $0.4 \leq [Q] \leq 0.8$. If the symbols $P$ and $Q$ are written as one place predicates, this knowledge could be represented by the following set: $\{[P(x)]_x = 0.6, [P(x) \rightarrow Q(x)]_x = 0.8\}$. It is easy to show that $0.4 \leq [Q(x)]_x \leq 0.8$ is deducible from this knowledge.

**Example 5.2** *Comparative Probabilities*

If our knowledge base consisted of a set of rankings, e.g., the set

$$\{[H_1(x)|E(x)]_x > [H_2(x)|E(x)]_x, [H_2(x)|E(x)]_x > [H_3(x)|E(x)]_x\},$$

then using the field axioms, in particular the transitivity axiom $(\forall \mathbf{xyz})\ \mathbf{x}{>}\mathbf{y} \wedge \mathbf{y}{>}\mathbf{z} \rightarrow \mathbf{x}{>}\mathbf{z}$, it is possible to rank the degrees of belief formulas of the form $\mathcal{B}(H_i(c)|E(c))$. That is, it is deducible, e.g., that the degree of $\mathcal{B}(H_1(c)|E(c))$ is greater than the degree of $\mathcal{B}(H_3(c)|E(c))$. Rankings of this sort may be sufficient when all that is required is to choose among the alternative hypotheses. For example, in choosing between competing diagnoses.

**Example 5.3** *Inheritance*

For example, we may have the following information: "Elephants are gray", "Royal Elephants are elephants", "Royal Elephants are not gray", "*Clyde* is a Royal Elephant", and "*Clyde* is an Elephant." This knowledge can be encoded in **Lp** with the following set of sentences:

$$\begin{aligned}\{\quad & [Gray(x)|Elephant(x)]_x > \mathbf{c},\\ & \forall x\, Royal\_Elephant(x) \rightarrow Elephant(x),\\ & [Gray(x)|Royal\_Elephant(x)]_x < 1{-}\mathbf{c},\\ & Royal\_Elephant(Clyde), Elephant(Clyde) \quad \},\end{aligned}$$

where $\mathbf{c}$ is some field constant close to 1. Given this knowledge, we have that the degree of

$$\mathcal{B}(Gray(Clyde)|Royal\_Elephant(Clyde)) < 1{-}\mathbf{c},$$

while the degree of

$$\mathcal{B}(Gray(Clyde)|Elephant(Clyde)) > \mathbf{c}$$

However, it is deducible from the knowledge base that $\forall x\, Royal\_Elephant(x) \rightarrow Elephant(x)$. Hence, $\mathcal{B}(Gray(Clyde)|Royal\_Elephant(Clyde))$ is a preferred belief. It is less than $1 - \mathbf{c}$; thus, *Clyde* is probably not gray.

If it was not known that *Clyde* is a Royal elephant, just that he is an elephant, then belief formation would assign a degree greater than $\mathbf{c}$ to the belief that *Clyde* is gray, based on the knowledge that he is an elephant. If the new information that *Clyde* is a Royal elephant is now added to the knowledge base, this old belief would be retracted. That is, the preferred belief, based on the knowledge that *Clyde* is a Royal elephant, would now be obtainable from the knowledge base. This is an example of non-monotonic behaviour.

**Lp** is also capable of dealing with inheritance in the presence of composite classes formed with logical connectives and with inheritable relations, see [16]. A more traditional graph based approach which uses **Lp** as its underlying semantics is presented in [2].

# 6 Conclusions and Open Problems

A formal logic with predicates and a probabilistic component has been developed. By having a field of numbers explicitly in the semantics it can represent a far wider range of probabilistic information than heretofore possible with other probability logics. An essential difference between the type of knowledge expressible in this logic and probability logics using the possible worlds approach has been identified and with it the need for an inductive mechanism. A sound and complete proof theory has been presented for the logic, a

proof theory which can reason both with sentences in ordinary first-order logic as well as with the probabilistic information expressible in **Lp**. A simple inductive mechanism has been presented. It allows degrees of belief to be formed which display non-monotonic behaviour.

One very attractive area for future research is learning, i.e., accumulating the statistical knowledge expressed in **Lp** automatically, through experience. There are a large set of methods developed in statistics which may be applicable. This would have an impact on knowledge acquisition in systems where training cases are available, e.g., diagnosis systems.

Since the effects of probabilistic schemes of inference can be duplicated in **Lp**, it is possible to construct a theory of diagnosis. The author is currently working on a theory of diagnosis from statistical principles which should be usable in domains like medical diagnosis.

# 7 Acknowledgements

Len Schubert's comments have been an invaluable aid. In particular, the early insights into the inductive mechanism were his. Thanks also to Jeff Pelletier for providing some useful references to the philosophical literature, and to the referees for some helpful criticisms.

# References

[1] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13, 1980.

[2] Fahiem Bacchus. *A Heterogeneous Inheritance System Based on Probabilities.* Technical Report 87–03, Alberta Center for Machine Intelligence, University of Alberta, Edmonon, Alberta, Canada. T6G–2E9, 1987.

[3] R.D. Luce and H. Raiffa. *Games and Decisions.* John Wiley, New York, 1957.

[4] R. Carnap and R.C. Jeffrey. *Studies in Inductive Probability.* Univ. of California Press, Berkeley and Los Angeles, CA, 1971.

[5] Nils J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[6] Alan Bundy. Incidence calculus: a mechanism for probabilistic reasoning. *Journal of Automated Reasoning*, 1:263–283, 1985.

[7] Rudolf Carnap. *Logical Foundations of Probability.* University of Chicago Press, 1962.

[8] H. Gaifman. Concerning measures in first order calculi. *Israel Journal of Mathematics*, 2:1–18, 1964.

[9] H. Field. Logic, meaning, and conceptual role. *Journal of Philosophy*, 77:374–409, 1977.

[10] B. van Fraassen. Probabilistic semantics objectified. *Journal of Philosophic Logic*, 10:371–394, 1981.

[11] H. LeBlanc. Alternatives to standard first-order semantics. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic. Vol II*, pages 225–258, Reidel, Holland, 1983.

[12] C. Morgan. Weak conditional comparative probability as a formal semantic theory. *Zeit. fur Math. Log.*, 30:199–212, 1984.

[13] Hans Reichenbach. *Theory of Probability.* University of California Press, Berkeley and Los Angeles, CA., 1949.

[14] Henry E. Kyburg, Jr. *The Logical Foundations of Statistical Inference.* D. Reidel, 1974.

[15] Henry E. Kyburg, Jr. The reference class. *Philosophy of Science*, 50(3):374–397, September 1983.

[16] Fahiem Bacchus. *Statistically Founded Degrees of Belief.* Technical Report 87–02, Alberta Center for Machine Intelligence, University of Alberta, Edmonon, Alberta, Canada. T6G–2E9, 1987.

[17] Richard O. Duda, Peter E. Hart, and Nils J. Nilsson. Subjective bayesian methods for rule-based inference systems. In Bonnie Lynn Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 192–199, Morgan Kaufmann, 1981.

[18] R.W. Johnson. Independence and bayesian updating methods. *Artificial Intelligence*, 29:217–222, 1986.

[19] John Bell and Moshé Machover. *A Course in Mathematical Logic.* Elsevier, Netherlands, 1977.

[20] S. MacLane and G. Birkhoff. *Algebra.* Macmillan, New York, 1968.

[21] E. Rich. Default reasoning as likelihood reasoning. In *AAAI–83*, pages 348–351, 1983.

[22] Matthew L. Ginsberg. Does probability have a place in non-monotonic reasoning? In *Proceedings of the 9th IJCAI*, pages 107–110, August 1985.

[23] Benjamin N. Grosof. Non-monotonicity in probabilistic reasoning. In *Proceedings of the AAAI/RCA Workshop on Uncertainty and Probability in Artificial Intelligence*, pages 91–98, 1986.

# A New Normative Theory of Probabilistic Logic

*Romas Aleliunas*

*Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6*

ABSTRACT *By probabilistic logic I mean a normative theory of belief that explains how a body of evidence affects one's degree of belief in a possible hypothesis. A new axiomatization of such a theory is presented which avoids a finite additivity axiom, yet which retains many useful inference rules. Many of the examples of this theory—its models—do not use numerical probabilities.*

*Another goal is to indicate the general reasons why recent theories of decision-making and belief (non-monotonic "logic," the Dempster-Shafer theory, etc.) do not represent anything really new.*

Every rational decision-making procedure is founded on some theory of probabilistic logic (also called a theory of rational belief). This article proposes practical and logically correct theories of rational belief that are more general than the probability theory we learn in school, yet remain capable of many of the inferences of this latter theory. We also show why recent attempts to devise novel theories of rational belief usually end up being either incorrect or reincarnations of the more familiar theories we learned in school.

## What is a Probabilistic Logic ?

Following KEYNES [1921], I take probabilistic logic to be any scheme for relating a body of evidence to a potential conclusion (a hypothesis) in a rational way. We assign *degrees of belief* (which we also call *probabilities* ) to the possible relationships between hypotheses and pieces of evidence. These relationships are called *conditionals*. We will use the expression "f(P|Q)" to stand for "the conditional probability of P given the evidence Q as given by the probability assignment f." We have chosen to encode the potential hypothesis, P, and the collected evidence, Q, as finite sentences of some language **L**. We take *probabilistic logic* to be synonymous with *theory of rational belief* , *probability* to be identical to *degree of belief*, and *probability assignment* to be the same as *belief function* . Normative theories of rational belief have had a long history (KEYNES [1921], CARNAP [1950], FINE [1973]).

Let the set of probabilities be **P**. We begin by regarding this as a set of *uninterpreted formal marks*. A theory of probabilistic logic is chiefly concerned with identifying the characteristics of a family **F** of functions from **L**x**L** to **P**. **F** is the *set of permissable belief functions* (also called the *set of possible probability assignments* ) from which a rational agent is permitted to choose. Our convention is that for any f in **F**, f(P|Q) represents the probability assigned by f to the hypothesis P given the evidence Q. P and Q can be any statements in **L**—no distinction is made between hypothesis statements and evidence statements in probabilistic logic.

The set **F** of rational probability assignments clearly cannot be the set of all functions from **L**x**L** to **P**. Both **F**, and each individual element in it, must be subject to some constraints. Of course different theories of rational belief may vary in their choices for these constraints, but I believe they all share some in common. One such common constraint is: if "I believe P is certain when I see Q" then it cannot also be the case that "I believe ~P is possible when I see Q." Moreover, the correctness of these constraints—and therefore of the rules of inference they engender—cannot depend on the particular application that they are put to. Decision "theory" does not dictate the standards of correctness for inferences in probabilistic logic (TUKEY [1960]).

Does the set of probabilities **P** have any internal structure? We must, at the very least, be able to assert that some conditionals are more believable than others, or else there is no practical use for this theory. This implies that we have, at the very least, a partial ordering among conditionals. If we

want to approach the subject algebraically, we can, without loss of generality, introduce a new probability symbol into **P** for each pair of sentences drawn from the language **L** and assign whatever (partial) set of orderings between these symbols that we wish. In this way we can reproduce any partial ordering whatsoever between the conditionals as an ordering of the elements of **P**. A theory which adopts this approach is called a *weak comparative theory of probability* in FINE [1973].

A theory of rational belief is not a substitute for a decision-making procedure. Probabilistic logic does not prescribe "rules of detachment," "rules of commitment," or other decision-making rules. The only judgements a theory of probabilistic logic tells you how to make are conditional judgements—how to assign a value to a conditional probability. Conditional judgements of probability are the only judgements that can be infallible. Moreover, conditional judgements are the natural inputs to a decision-making apparatus.

## What is a Decision Procedure ?

A *decision procedure* tells you how to bet. I refrain from using the term *decision theory* since it is not a theory in the usual sense. A decision procedure is, instead, a concrete thing—a particular set of rules describing the behaviour of a specific physical machine. Decisions sometimes have to be made with little or no evidence; decision procedures make mistakes.

Regardless how it may be implemented, a decision procedure can always be described in a certain standard way. To formulate this kind of description we must first identify the general principles that govern the procedure's behaviour. I find it convenient to divide such a description into two parts. The first part describes the principles of evidence—the probabilistic logic—that the decision-making machine (implicitly) employs. The second part describes how these principles are applied in this specific implementation. Ideally the only possible causes of bad decisions are misapplications of probabilistic logic.

The principles governing the application of probabilistic logic to decision-making fall into several subgroups. These concern the problems of statistics, utility assignment, and the formulation of decision rules.

*Statistics* is concerned with how probabilities are assigned and how the statistical model is built. Statistical problems include the choice of language for encoding evidence and hypothesis statements, the choice of a set of hypotheses, and the choice of simplifying independence assumptions. (A *hypothesis* is merely a statement whose expected utility is relevant to the decision-making machinery.)

A *utility assignment* describes how the decision procedure ranks the costs and rewards of various outcomes. *Decision rules* describe how probabilities and utilities combine to affect decisions.

Suppose, for instance, that we have a decision procedure called Alpha that always decides in favour of the hypothesis with maximum likelihood (GOOD [1983]). Then we can describe Apha as a procedure which (implicitly) assigns equal prior probabilities and equal utilities to all hypotheses (outcomes).

Suppose we have another procedure called Beta that merely scans down an ordered list of hypotheses and chooses the first one that is consistent with the current evidence. Then Beta's probabilistic logic simply judges hypotheses to be either possible (probability $\neq 0$) or impossible (probability $= 0$) on the given evidence—a matter of testing logical consistency. Beta (implicitly) assigns non-zero prior probability exactly to those hypotheses which appear on its list, and Beta (implicitly) uses utilities ordered in the same fashion as the list.

The standards of correctness of a probabilistic logic cannot be dictated by the application to which it is put (TUKEY [1960]). But what many writers have failed to realize—and this is a novel feature of the theory presented here—is that many other aspects of a probabilistic logic can be tailored to suit the application. We have, for example, a great deal of latitude in choosing a convenient set of probabilities **P**. After all, if the decision-procedure does not (implicitly) use numerical utilities, then why should it use numerical probabilities? There is no reason to use a probabilistic logic that makes fine discriminations between degrees of belief if these discriminations serve no practical end.

## New Axioms for Probabilistic Logic

These axioms are flexible enough to permit one to tailor the set of probabilities to the application at hand. The probabilities are therefore treated as uninterpreted formal marks in the axiomatization given below. We will examine possible interpretations later—each interpretation will give rise to a probabilistic logic.

The axioms describe the constraints that are appropriate for any rational theory of belief. The axioms fall into three groups: (1) axioms about the domain and range of each probability assignment f in **F**, (2) axioms stating consistency constraints that each individual f in **F** must obey,

and (3) axioms that say something about **F** itself. Finite additivity does not appear as an axiom since it essentially forces probabilities to be numbers.

The usual semantics for probabilistic logic uses "possible worlds.*" We associate to each proposition P a set of *situations* or *possible worlds* S(P) in which P holds. ("Possible world" is left as a primitive unanalyzed concept.) When Q is given as evidence, the conditional probability f(P|Q) is some measure of the fraction of the set S(Q) that is occupied by the subset S(P&Q). (The individual points in S(Q) need not have equal weight. They need not even have any weight at all as is often the case in Lebesgue measure theory.) This fraction is given by a (possibly non-numerical) function $\mu^f$( S(P&Q), S(Q)) whose values range over the set **P**. (The familiar numerical theory of probability relies on this same semantics except that the relative sizes of the fractions must be coded numerically in this case.) The correctness of any axioms for probabilistic logic can be grasped by a semantics of this form.

*AXIOMS for PROPOSITIONAL PROBABILISTIC LOGIC*

*Axioms about the domain and range of each* f *in* **F**.

**1.** The set of probabilities, **P**, is a partially ordered set. The ordering relation is " ≤ ."

**2.** The set of sentences, **L**, is a free boolean algebra with operations &, v, and ~, and it is equipped with the usual equivalence relation " ≈ ." The generators of the algebra are a countable set of *primitive propositions*. Every sentence in **L** is either a primitive proposition or a finite combination of them. (See BURRIS *et al.* [1981] for more details about boolean algebra.)

**3.** If P ≈ X and Q ≈ Y, then f(P|Q)=f(X|Y).

*Axioms that hold for all* f *in* **F**, *and for any* P, Q, R *in* **L**.

**4.** If Q is absurd (*i.e.* Q ≈ R&~R), then f(P|Q) = f(P|P).

**5.** f(P&Q|Q) = f(P|Q) ≤ f(Q|Q).

**6.** For any other g in **F**, f(P|P) = g(P|P).

**7.** There is a monotone non-increasing total function, i, from **P** into **P** such that f(~P|R) = i( f(P|R) ).

**8.** There is an order-preserving total function, h, from **P**x**P** into **P** such that f(P&Q|R) = h(f(P|Q&R), f(Q|R)). Moreover, if f(P&Q|R) = 0*, then f(P|Q&R) = 0* or f(Q|R) = 0*, where we define 0* = f(~R|R) as a function of f and R.

**9.** If f(P|R) ≤ f(P|~R) then f(P|R) ≤ f(P|Rv~R) ≤ f(P|~R).

* Historically, the possible world semantics of probability theory seems to have inspired KRIPKE's semantics for modal logics [1980, p.16].

*Axioms about the richness of the set* **F**.

Let **1** = Pv~P. For any **distinct primitive** propositions A, B, and C in **L**, and for any arbitrary probabilities a, b, and c in **P**, there is a probability assignment f in **F** (not necessarily the same one in each case) for which—

**10.** f(A|**1**) = a, f(B|A) = b, and f(C|A&B) = c.

**11.** f(A|B) = f(A|~B) = a and f(B|A) = f(B|~A) = b.

**12.** f(A|**1**) = a and f(A&B|**1**) = b, whenever b ≤ a.

## Commentary on the Axioms

**1.** You cannot do anything useful with a probabilistic logic if you cannot compare some degrees of belief with others. On the other hand not all applications require a total ordering for probabilities.

**2&3.** We clearly do not sacrifice generality by letting **L** be a free algebra.

The laws of boolean equivalence state, to pick one particular example, that the truth conditions of the sentence P&R are the same as those of R&P. In other words each sentence picks out the same set of situations: S(P&R)= S(R&P). These two conjunctions must therefore have the same probability given Q since f(P&R|Q) = $\mu^f$(S((P&R)&Q), S(Q)) = $\mu^f$(S((R&P)&Q), S(Q)). (Indeed, a weaker notion than boolean equivalence is sufficient for most of the results below.)

The *prior probability* of P, which we will write as f(P), can be defined to be f(P) = f(P| P v ~P). This is correct since ~P v P is vacuous when taken as an evidence statement.

**4.** Absurd evidence (such as ~R&R) provides no basis for judging the probability of any hypothesis since S(~R&R) is the empty set. Assigning f(P|~R&R)=f(P|P) in these cases is a mathematically elegant way to treat this anomaly consistently (MORGAN [1982]). (An equally acceptable, alternative, treatment is to restrict the domain of each f so that absurd evidence is always excluded. Adopting this second approach does not alter any of the results stated later, but it does complicate the statement of the axioms.)

**5&6.** These axioms establish useful scaling conventions among probabilities and probability assignments.

**7.** R. COX [1946, 1961] first used the idea that the probability of ~P given R can be determined from the probability of P given R as an axiom in his study of probabilistic logics using real numbers. He did not, however, assume that i was order inverting.

**8.** This is a weak form of the product rule for probabilities: f(P&Q) = f(P|Q)·f(Q). We will call h the *product of probabilities*. R. COX [1946, 1961] also introduced this axiom, but without insisting that h be order preserving or that it have no non-trivial zeroes (see below).

The contrapositive form of the last part of axiom 8 states that the product of two non-zero probabilites is also not zero. This is to say that, given the evidence R, if P&Q is impossible, then either Q is impossible, or else P is impossible given Q, or both. We say that h *has no non-trivial zeroes* when this condition holds.

**9.** A *finite additivity axiom* is an axiom that asserts the existence of a function h* such that for any f, P, Q, and R—

f(PvQ | R) = h*(f(P | R), f(P&~Q | R)).

Finite additivity requires that the probability of PvQ can be determined from the probabilities assigned to just the two mutually exclusive sentences P and ~P&Q (ACZEL [1966]).

Suppose P is statistically independent of R, which we write as f(P|R) = f(P|~R). Then it should follow that f(P) = f(P|R) = f(P|~R) since the probability of P does not depend on whether R holds or not. Axiom 9 permits this conclusion to be drawn. So does finite additivity, but we are avoiding this axiom in favor of the weaker axiom 9.

Consider next the following argument conducted in English: "If this coin lands heads I will almost certainly end up winning the entire game. If it lands tails then it is still likely that I will win (because I 'm so far ahead). So even before we toss the coin, it is clear right now that my chances of winning are somewhere between likely and almost certain." The probabilities mentioned in this argument are not numbers, so no addition formula is possiblc, yet thc conclusion is compelling. Axiom 9 is sufficient to draw the conclusion from the premises in this argument also.

**10, 11 & 12.** These three axioms guarantee that the set **F** of probability assignments is rich enough to freely model: (1) any behaviour of growing chains of evidence, (2) statistically independent events, and (3) partial probabilty assignments to conjunctions of potential hypotheses.

Axiom 10 shows that the product of probabilities is associative since f(P&Q&R) = h(h(f(R|P&Q),f(Q|P)), f(P)) = h(f(R|P&Q), h(f(Q|P), f(P))).

Axiom 11 can be used to show that the product is commutative by using the equation f(P&Q) = h(f(P|Q), f(Q)) = h(f(Q|P),f(P)). Axiom 9 is used in this proof to calculate f(P) and f(Q) from f(P|~Q) = f(P|Q) and f(Q|~P) = f(Q|P).

Hence we see that (**P**, h) forms a commutative semigroup, so we may write p·q instead of h(p,q) whenever this is convenient.

Axiom 12 is another way of stating our conviction that any conditional must, in principle, have a probability. If we know, for example, that

f(March winds) = a

f(March winds & April showers) =b

then there must be some probability r such that

f(April showers | March winds) = r.

Of course we must have b = r·a (axiom 8), and b ≤ a (axiom 5) or else this probability assignment is inconsistent. FUCHS [1963] calls any partially ordered semigroup **naturally ordered** just in case there always exists a solution r to the equation q = r·p whenever q ≤ p. Axiom 12 only requires that a solution exist, not that it be unique.

## Possible Interpretations of this Theory

These axioms do not presume the existence of probabilities called 0 and 1 in the set **P**. It easily follows from the axioms, however, that f(P|P) = f(Q|Q) = g(Q|Q) for arbitrary f, g, P and Q, and therefore the convention 1 = f(P|P) leaves the probability 1 well-defined. Likewise 0 = f(~P|P) is a good definition of probability 0. Thus every **P** that is consistent with the above axioms must have a 0 and a 1. Probability 1 denotes complete certainty, and probability 0 denotes impossibility. If I say "Q is possible," I mean f(Q)≠0 for my choice of f.

The probabilities 0 and 1 are therefore special elements of every possible **P**. To give a mathematical model for a **P** which satisfies these axioms we must therefore give a consistent interpretation to each component of the structure (**P**, ≤, h, i, 0, 1).

*(Model 1) 0,1-Probabilistic Logic*

**P**(**2**) = {0, 1} and 0·0 = 0·1 = 0, 1·1 = 1, i(0) = 1, i(1) = 0, and 0 ≤ 1.

*(Model 2) Simple Real Probabilistic Logic*

**P**(**R**) = [0,1], the closed interval of real numbers from 0 to 1. Let p·q be the ordinary numerical product, and let i(p) = 1 – p. Use the usual ordering relation.

For more examples suppose (**P**, ≤) is a totally ordered set with n elements. Then the table below displays the number,

N(n), of non-isomorphic models consistent with this constraint—

| n | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| N(n) | 1 | 1 | 2 | 3 | 7 | 16 |

Yet another model of these axioms is the following one mimicing English probabilities. My aim here is not to claim that this is the correct model for English probabilities, but instead to show that a good approximation to English probabilities will not be ruled out by these axioms.

*(Model 3) Simplified English Probabilistic Logic*

Let **P(E)** be the algebra of probabilities generated by the two formal symbols {LIKELY, UNLIKELY}, subject to the following additional constraints:

(C1) UNLIKELY = i(LIKELY )

(C2) 0 < UNLIKELY < LIKELY < 1.

The elements of **P(E)** are strings of symbols generated by—

(G1) concatenating previously constructed strings in **P(E)**,

(G2) forming i(s) where s is a string in **P(E)**,

(G3) introducing the formal symbol s(p,q) for any pair of elements p and q in **P(E)** whenever $p \leq q$ and there does not yet exist a solution, r, in **P(E)** for the equation $p = q \cdot r$.

The only ordering relations are those that can be inferred from (C1), (C2), and the properties of the functions h and i. This set is not totally ordered.

The symbols introduced by (G3) do not, as far as I know, have English names. But not being able to name them in English does not mean they do not exist.

## Is This Theory Useful ?

This theory is useful for several reasons—

(U1) The number of distinct degrees of probability is a feature left up to the user. This makes it possible to tailor the probabilistic logic to the goals of the decision procedure.

(U2) Many useful inferences can still be carried out.

(U3) Some statistical problems, like assigning prior probabilities, can be considerably simpler to solve in a non-numerical system. For example in the (unique) three-valued system you only have to decide if an event is certain, impossible, or neither.

To illustrate (U2) I will exhibit a calculation in the system **P(E)** (Model 3). Let's start with the three premises—

(P1) *It is possible that it's cloudy.*

(P2) *If it's cloudy, it's likely to rain.*

(P3) *If it rains then the street will get wet.*

The following conclusion can be drawn from these premises—

*If it's cloudy, the chances are better than likely that the street will get wet.*

Let us code these premises symbolically for convenience. They become—

(P1) $f(C) \neq 0$.

(P2) $f(R|C) = \text{LIKELY}$.

(P3) $f(W|R) = 1$.

Now by (P3) we have $f(\sim W|R)=0$. Hence $0=f(\sim W|R)\cdot f(R)=f(\sim W\&R)\geq 0$, and therefore $f(\sim W\&R\&C)= 0$ by axiom 5. Now $0=f(\sim W\&R\&C)=f(\sim W|R\&C)\cdot f(R|C)\cdot f(C)$, and because $f(R|C)\neq 0$ and $f(C)\neq 0$, it must be the case that $f(\sim W|R\&C)=0$ since products have no non-trivial zeroes. Thus $f(W|R\&C)=1$. It's now easy to find $f(W|C) \geq f(W\&R|C) = f(W|R\&C)\cdot f(R|C)=1\cdot\text{LIKELY}=\text{LIKELY}$. Hence $f(W|C)\geq\text{LIKELY}$.

There are several points to note—

(1) We used the following trivial theorems:

$0 \leq p \leq 1$ for any probability p,

$p \cdot q \leq q$ for any pair of probabilities p and q.

(2) Premise (P1) is needed in this particular derivation.

(3) The inequality is a necessary part of the conclusion. After all, maybe the street can get wet in some other way.

(4) This argument holds regardless of whether it is cloudy or not at the moment, and regardless of whether it ever rains or not. This argument is about the state the world could be in—it does not confine itself to the unique actual state the world is in now. Probabilistic logic treats all conditionals, counterfactual or not, in the same way.

(5) The conclusion is completely convincing even though LIKELY can have no numerical interpretation.

## The Tar-Pit Theorem

Under what conditions does the familiar numerical theory of probability become the only possible theory of rational belief? Richard COX [1946, 1961] first investigated this question under the assumption that probabilities were numbers. ACZEL [1966] reports stronger results under the assumption of finite additivity for probabilities. The following theorem is in the same vein, but it drops these two assumptions and even some of the axioms. I call it the Tar-Pit Theorem because it shows how easy it is to get stuck in stuff that has been lying around for a long time.

*Definition*

**P** is *archimedean ordered* if, for any $p \neq 1$ and any $q \neq 0$ in **P**, the repeated product $p \cdot p \cdot \ldots \cdot p = p^n$ becomes smaller than q for some positive integer n.

*The Tar-Pit Theorem*

Under the conditions established by axioms 1 through to 12, the following two statements are logically equivalent:

(1) The set of probabilities **P** is totally ordered and also archimedean ordered.

(2) The algebraic structure (**P**, ≥, h, i, 0, 1) is isomorphic to a subalgebra of the system (**P**(**R**), ≥, ·, i, 0, 1) given as Model 2, namely Simple Real Probabilistic Logic.

Dropping both axiom 11 and the assumption that products have no non-trivial zeroes does not affect the result.

A short proof of this is given in the Appendix. The requirement for archimedean ordering appearing this theorem can be removed and replaced by either cancellativity (*i.e.* for any p,q,r ≠0 in **P**, if p·r = q·r then p = q), or else by finite additivity (with some other minor adjustments in hypotheses). Space is too limited to state these results precisely.

## On Reinventing the Wheel

Several theories of belief have been recently developed that have claimed some novelty over the familiar probability theory with real numbers. The DEMPSTER-SHAFER theory is one example, and the MYCIN theory is another (see BUCHANAN & SHORTLIFFE [1984] for a description of both). Both of these theories use numbers to measure degrees of uncertainty. In light of the Tar-Pit theorem it is no surprise, therefore, that KYBURG [1987] has shown that the DEMPSTER-SHAFER theory can be reinterpreted and understood in the language of familiar probability theory, and that HECKERMAN [1986] has done a similar thing for the MYCIN theory. In the process of uncovering the exact relationship, each author has exposed the general statistical assumptions that were surreptitiously imposed by each of these two systems. Whatever novelty these systems may have had rested on these questionable statistical assumptions.

KYBURG [1987] showed that the DEMPSTER-SHAFER theory manipulates convex sets of probability assignments. (A set **S** of probability assignments is *convex* if for any two probability assignments f and g in **C**, and any number z between 0 and 1, the function z·f + (1-z)·g is also in **C**. As a consequence of this the range of values assumed by f(P|Q) for fixed P and Q, when you let f range over all members of **C**, forms a closed interval.)

HECKERMAN [1986], on the other hand, found that a MYCIN "certainty factor" is a monotonic increasing function of the likelihood ratio f(P|Q) / f(P|~Q).

Both KYBURG and HECKERMAN observed, in the respective systems they studied, that certain statistical assumptions about conditional independence were built into the inference rules themselves. HECKERMAN, for example, showed how MYCIN's implicit assumptions made MYCIN incapable of correctly handling a statistical inference problem that involved three or more mutually exclusive and exhaustive possible outcomes.

Non-monotonic "logics," which originated with REITER [1980] and MCDERMOTT & DOYLE [1980], are another recent novelty. Given a set of premises, say $Q1\&Q2\&\ldots\&Qn$, a non-monotonic "logic" is a non-deterministic machine for deriving "plausible inferences." We will write "P **d** $Q1\&Q2\&\ldots\&Qn$" if P is derivable from the stated premises by one of these machines.

At the very least a non-monotonic "logic" must tell me what the good bets are. If "P **d** $Q1\&Q2\&\ldots\&Qn$" does not mean that P is a good bet given I have already seen $Q1\&Q2\&\ldots\&Qn$, then I see no practical point to non-monotonic "logic." I am indebted to Charles MORGAN for showing me why a non-monotonic "logic" cannot be a logic in the usual sense of the word. This still leaves open the possibility, however, that non-monotonic "logic" is some sort of betting system. Here are three possible explanations for "plausible inference" in terms betting rules—there are, of course, an endless number of possibilities:

(D1) P **d** Q ⇒ u(P)·f(P|Q) > c,

(D2) P **d** Q ⇒ min {u(P)·g(P|Q) | g is in **S**} > c,

(D3) P **d** Q ⇒ f(P|Q) > c·f(P|~Q),

where u(P) denotes the utility of outcome P, f is some fixed probability assignment, c is some fixed constant, and **S** is some special subset of **F**.

Suppose Q holds. P is a good bet, according to (D1), when my expected utility from it exceeds some threshold. (D2) is more conservative: it says P is a good bet only when the minimum I expect to gain (over some set of probability assignments) is large enough. (D3) uses likelihoods to pick out good bets.

For example REITER's system [1980], to a first approximation, judges hypotheses only on the basis of consistency and it assumes an equal assignment of utilities to all outcomes. It is therefore no surprise that such a scheme is too simple to be very useful. (To be precise, REITER's system is based on "P **d** Q⇒ f(P|Q&B)>0 ," where B is some base set of facts that is never doubted. This criterion can be simplified to "P **d** Q⇒ g(P|Q)>0 " because, for

fixed B, the equation g(X|Y)=f(X|Y&B) defines g to be another valid probability assignment in **F**.)

Non-monotonic "logic" is further hampered by not manipulating probabilities and utilities explicitly. Suppose we adopt the criterion that any event with probability exceeding 0.80 is a good bet, and suppose I have built a non-monotonic "logic" that is sound by this criterion. Toss a fair die (with six faces). Let L(i) code the event that face i shows. Let Q denote the background information about dice tossing. Then certainly L(i) will not be a non-monotonic inference from Q for any i in this system. The disjoint event L(1) v L(2) v L(3) v L(4) v L(5), however, is a good bet according to our chosen criterion. But how can an inference like this be made in general within a non-monotonic "logic" that excludes probabilities from its calculations?

Fuzzy "logic" (ZADEH [1965], GUPTA *et al.* [1979]) is another novelty. This theory leads to bad bets because, among other things, it is inconsistent with axiom 8. Fuzzy "logic" replaces the functional equation f(H&T)=h(f(H|T),f(T)) with f (H&T)=h(f(H),f(T)) where h is still a function with no non-trivial zeroes.

To demonstrate the difficulties this alteration leads to, consider tossing a fair coin. By symmetry, fuzzy "logic" must assign equal "possibility levels" to the two outcomes so that f(H)=f(T)=p for some p≠0. Now according to fuzzy "logic's" rules, f(H&T) = min(f(H),f(T)) = p ≠ 0. But the event H&T is impossible! Interpreted charitably, this means that 0 is not the only way of coding impossibility in fuzzy "logic." We have just seen that the value p also must code for impossibility. But then both possible outcomes of the coin toss are now marked impossible—an absurdity. I am quite prepared to bet money against this unrealistic "logic" since I do not expect to lose.

The onus is clearly on the inventor of a new theory of rational belief (or rational decision-making procedure) to demonstrate its novelty and its correctness. None of the theories mentioned in this section are, simultaneously, a new and a good guide to betting.

## Appendix: Proof of the Tar-Pit Theorem

Actually, this proof continues to hold even if we omit axiom 11 (independence) and the last part of axiom 8 (h has no non-trivial zeroes).

That (2) implies (1) is obvious. So we will concentrate on proving the converse, namely any archimedean totally ordered set **P** that satisfies the remaining axioms must be a set of real numbers under ordinary multiplication.

This proof relies on another theorem that was first found by O. HOLDER at the turn of the century, and later rediscovered by A. H. CLIFFORD (see FUCHS [1963, p.165]). HOLDER and CLIFFORD proved that any totally and naturally ordered archimedean semigroup must be isomorphic to a subsemigroup of one of the following three. The *non-zero* and *non-unit* elements of these three distinguished semigroups are isomorphic to—

(1) the open interval of reals (0,1) under ordinary multiplication,

(2) the half-open interval of reals [1/e, 1), under the binary operation max(x·y, 1/e)

(3) the interval [1/e, 1) augmented by an artificial element §, under the following operation:

$$x \cdot y \text{ if } x \cdot y \geq 1/e$$
$$\S \text{ if } x \cdot y < 1/e$$

where e is the base of the natural logarithms, and § < 1/e.

According to our axioms i(i(p)) = p, and so the function i is an anti-isomorphism of **P** which inverts **P**'s ordering relation. The semigroups resulting from (2) and (3) can therefore be ruled out because no such i can ever be found for them. This leaves only possibility (1), which, after we restore the 0 and 1 elements, is the closed interval [0,1] under multiplication. This proves the theorem.

## Acknowledgements

The Natural Sciences and Engineering Research Council of Canada supported my work while I was at Waterloo. Some of this work was prepared for presentation while I was a Visiting Scientist at the IBM Canada Laboratory. The preparation costs of this particular paper have been met by a President's Research Grant from Simon Fraser University.

I have a benefited from discussions with Harvey Abramson, Robert Hadley, Peter Ludemann, Charles Morgan, Len Schubert and my colleagues from the Logic Programming and AI Group at Waterloo, including Robin Cohen, Randy Goebel, Eric Neufeld, David Poole, and Maarten van Emden.

## References

Janos ACZEL [1966] **Lectures on Functional Equations and Their Applications**, Academic Press, New York.

Bruce G. BUCHANAN and Edward H. SHORTLIFFE, editors [1984] **Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project**, Addison-Wesley, Reading, Massachusetts.

Stanley BURRIS and H.P. SANKAPPANAVAR [1981] **A Course in Universal Algebra**, Springer-Verlag, New York.

Rudolph CARNAP [1950] **Logical Foundations of Probability**, University of Chicago Press, Chicago.

— [1962] "The aim of inductive logic," in **Logic, Methodology, and Philosophy of Science**, eds. E. Nagel, P. Suppes, and A. Tarski, Stanford University Press, Stanford, pp.303-318.

Richard T. COX [1946] "Probability, Frequency, and Reasonable Expectation," *American Journal of Physics* , Vol. 14, pp.1-13.
— [1961] **The Algebra of Probable Inference**, John Hopkins Press, Baltimore.

Terrence L. FINE [1973] **Theories of Probability: An Examination of Foundations**, Academic Press, New York and London.

Simon FRENCH [1984] "Fuzzy Decision Analysis: Some Criticisms" in **Studies in the Management Sciences: Fuzzy Sets and Decision Analysis**, Vol. 20, North-Holland, Amsterdam and New York.

Laszlo FUCHS [1963] **Partially Ordered Algebraic Systems**, Pergamon Press, Oxford-London-New York-Paris.

I. J. GOOD [1983] **Good Thinking, The foundations of probability and its applications,** University of Minnesota Press, Minneapolis, 1983.

Madan M. GUPTA, Rammohan K. RAGADE, Ronald R. YAGER, editors [1979] **Advances in Fuzzy Set Theory**, North-Holland, Amsterdam-New York-Oxford.

David HECKERMAN [1986] "Probabilistic Interpretation for MYCIN's Certainty Factors," in **Uncertainty in Artificial Intelligence**, Laveen N. Kanal and John F. Lemmer (eds.), North-Holland, New York, 1986, pp.167-196.

John Maynard KEYNES [1921] **A Treatise on Probability**, Macmillan, London.

Bernard O. KOOPMAN [1940] "The Axioms and Algebra of Intuitive Probability," *Annals of Mathematics* , Vol. 41, No. 2, pp. 269-292, April 1940.
— [1941] "Intuitive Probabilities and Sequences," *Annals of Mathematics* , Vol. 42, No. 1, pp. 169-187, January 1941.

Henry E. KYBURG, Jr. [1987] "Bayesian and Non-Bayesian Evidential Updating," *Artificial Intelligence,* Vol. 31, No.3, March 1987, pp.271-293.

Saul KRIPKE [1980] **Naming and Necessity**, Harvard Uinversity Press, Cambridge, 1980, p.16.

D. MCDERMOTT & J. DOYLE [1980] "Non-monotonic Logic I," *Artificial Intelligence* , Vol 13, pp.27-39.

D. MCDERMOTT [1982] "Non-monotonic Logic II: non-monotonic modal theories," *Journal of the ACM* , Vol 21, No. 1, pp.33-57.

Charles G. MORGAN [1982] "There is a Probabilistic Semantics for Every Extension of Classical Sentence Logic," *Journal of Philosophical Logic* , Vol. 11, pp.431-442.

David POOLE, Randy GOEBEL, Romas ALELIUNAS [1987] "Theorist: A Logical Reasoning System for Defaults and Diagnosis," in the **Knowledge Frontier**, Nick Cercone & Gordon McCalla (editors), Springer-Verlag, New York, 1987, pp.331-352.

Raymond REITER [1980] "A logic for default reasoning," *Artificial Intelligence* , Vol. 13, pp.81-132.

John W. TUKEY [1960] "Conclusions vs Decisions," *Technometrics* , Vol. 2, No. 4, November, 1960.

Lotfi A. ZADEH [1965] "Fuzzy Sets," *Information and Control,* Vol. 8, pp.338-353.

# On Using Modal Structures to Represent Extensions to Epistemic Logics

**Sharon J. Hamilton**[1] and **James P. Delgrande**[2]

School of Computing Science
Simon Fraser University
Burnaby, B.C.
Canada V5A 1S6

## Abstract

Modal structures have recently been developed as an alternative to Kripke structures for providing the semantic basis for various modal logics. While modal structures are mathematically equivalent to Kripke structures in a certain sense, it is argued that modal structures provide a simpler and more intuitive basis for representing particular states of the modal notions of knowledge and belief. Since these notions, rather than the traditional notions of necessity and possibility, are of particular interest to Artificial Intelligence, it is of interest to examine the applicability and versatility of these structures. This paper presents an investigation of modal structures, by examining how they may be extended to account for generalizations of Kripke structures. In particular we show how they may be extended to situations, or partial possible worlds, to a generalized accessibility relation among situations, and to a full first-order system. In all cases the extensions are shown to be equivalent to the corresponding extension of Kripke structures. In addition, a three step transformation from Kripke to modal structures is presented to clarify the relationship between the two models.

**Keywords:** modal structures, Kripke structures, epistemic logics, possible world semantics.

## 1. Introduction

Kripke structures were proposed in [Kripke 63] as a semantic basis for modal logics of necessity and possibility. The general idea is that a proposition $\alpha$ may either happen to be true (for example, it is true that I happen to be sitting at my desk at the present time) or may necessarily be true (for example, every square must of necessity have four sides). The latter proposition may be written as $L\alpha$, to distinguish it from simple truth. The truth of $L\alpha$ is expressed relative to the notion of "possible worlds": $L\alpha$ is true just when $\alpha$ is true in all worlds that are possible given the present world. By varying the notion of what worlds are accessible from a particular world, one may specify various differing notions of necessary truth. [Halpern and Moses 85] provides a highly readable introduction to such modal logics.

The notion of a possible world in this context seems highly intuitive, since necessity is readily interpreted as "truth in all possible worlds". However, modal logics have also been used to model (among other things) the epistemic notions of knowledge and belief, and for these notions, the possible worlds basis is perhaps not as intuitive. In the semantics of such systems, $B\alpha$ (that is, the agent believes $\alpha$) is true just when $\alpha$ is true in all worlds that are consistent with what the agent believes. [Fagin, Halpern, and Vardi 84] and [Fagin and Vardi 85] argue that it is not clear how Kripke structures reflect particular states of knowledge and belief, and introduce *modal structures* as a formally equivalent but arguably more intuitive alternative. These structures consist of a number of *levels*, where level 0 corresponds to what is true in the real world, level 1 corresponds to what the agent directly believes about the world, level 2 corresponds to what the agent believes that it believes about the world, and so on. Hence, the levels of modal structures correspond neatly to levels of meta-knowledge, and arguably allow individual states of knowledge to be clearly represented.

In this paper we examine the extensibility of modal structures by examining how they may be augmented to account for various generalizations of Kripke structures. In Section 2, we provide background information by describing modal structures in general, giving their formal equivalence to Kripke structures, and describing a particular modal structure that models the logic weak S5 [Fagin, Halpern, and Vardi 84], [Fagin and Vardi 85]. We also describe an original method of deriving modal structures from Kripke structures. In Section 3, we investigate the extensibility of modal structures to the logics of explicit and implicit belief described in [Levesque 84a] and [Lakemeyer 87]. In these systems, the notion of possible worlds in Kripke structures is generalized to that of partial possible worlds, or *situations*. Section 4 describes briefly how the first-order logic of belief described in [Levesque 84b] can be represented in modal structures. The formal equivalence of the two approaches has been demonstrated for all extensions; for brevity, however, the theorems have been omitted. The theorems and their proofs, together with further details, may be found in [Hamilton 87].

[1]This research was supported in part by a Postgraduate Award from Bell-Northern Research and a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship.

[2]This research was supported in part by the Natural Sciences and Engineering Research Council of Canada grant A0884.

## 2. Kripke Structures and Modal Structures

This section briefly reviews the semantic structures with which we are concerned. A Kripke structure consists of a set of *states G*, a binary *accessibility relation R* over those states, and an assignment of truth values $\pi$ to primitive propositions at states. For $w,v \in G$, if *wRv*, then *v* is said to be *accessible* from *w*. Informally, the states are interpreted as *possible worlds*, and the accessibility relation gives those worlds that are considered possible with respect to a given world. For epistemic logics (*i.e.*, logics of knowledge and belief), the accessibility relation gives those worlds that are consistent with what an agent believes at a particular "real" world. An agent at a world *w* is said to *believe* a proposition *p* if and only if *p* is true in all worlds accessible from *w*.

Different restrictions placed on the accessibility relation enable one to model different properties of knowledge or belief. Consistency of belief is obtained in a Kripke structure by requiring that the accessibility relation be *serial* (for every world *w*, *wRv* for some *v*). *Positive introspection*, wherein an agent knows everything that it knows, is obtained by requiring that the accessibility relation be transitive (if *wRv* and *vRx*, then *wRx*); and *negative introspection*, wherein if an agent does not know something, it knows that it does not know it, is obtained by the Euclidean restriction (if *wRv* and *wRx*, then *vRx*). The logic *weak S5*, which is obtained by requiring that the accessibility relation be serial, transitive, and Euclidean, gives a kind of belief that is consistent, although not necessarily accurate with regard to the "real world". The logic S5 is obtained by requiring that the accessibility relation be *reflexive* instead of serial (for every world *w*, *wRw*). This requirement ensures that the agent's beliefs accurately reflect the "real world", and distinguishes *knowledge* from belief.

[Fagin, Halpern, and Vardi 84] and [Fagin and Vardi 85] introduce *modal structures* as an alternative to Kripke structures. A modal structure models a particular world *w* in a Kripke structure; it tells both what is true at *w* and what is believed by an agent **A** at world *w*.[3] It consists of a series of *levels*, where each level models a particular "depth" of the agent's knowledge or meta-knowledge. For example, the fact that primitive proposition *p* is true is recorded at level 0; the fact that agent **A** believes that *p* is true is recorded at level 1; and the fact that **A** believes that **A** believes that *p* is true is recorded at level 2.

The correspondence to Kripke structures is roughly as follows. World *w* in a Kripke structure describes a specific "state of affairs"; this (single) state of affairs is described at level 0 in the modal structure for world *w*. The agent is considered to be "at" level 0. The set of worlds immediately accessible from *w* models the agent's beliefs about *w*; these worlds constitute level 1 in a modal structure. The sets of worlds accessible from those accessible from *w* gives the agent's beliefs about its beliefs about *w*; these worlds are contained at level 2 of the modal structure, along with the worlds through which they are accessible. This process can be repeated arbitrarily many times to allow for arbitrarily deep nestings of beliefs. A modal structure thus gives the full accessibility information in a Kripke structure from a single world; to obtain *all* the information present in a particular Kripke structure, one requires in general as many modal structures as there are worlds in the Kripke structure.

A modal structure can now be formally defined [Fagin and Vardi 85]. **P** is a fixed, finite set of primitive propositions.[4]

> **Definition 1:** $f_0$: **P** $\rightarrow$ {*true, false*} is a *0th-order assignment.*

Intuitively, $f_0$ assigns truth values to the primitive propositions in **P** at level 0 of the modal structure.

> **Definition 2:** The tuple $<f_0>$ is called a *1-ary world* (or simply a *world*), because it contains a single element. In general, a tuple $<f_0, f_1,...,f_{k-1}>$ is called a *k-ary world* because it contains k elements.

Let $\mathbf{W_k}$ be the set of all k-ary worlds.

> **Definition 3:** $f_k$: {**A**} $\rightarrow$ Powerset($\mathbf{W_k}$) is a *$k^{th}$-order assignment*, for $k \geq 1$.

Intuitively, the assignment $f_k$ associates with the agent a set of possible k-ary worlds that are compatible with its depth-k beliefs. $f_k$(**A**) is the set of k-ary worlds associated with agent **A** by $f_k$, at level k of the modal structure.

> **Definition 4:** A *modal structure* is an infinite sequence $<f_0,f_1,...,>$ if the *prefix* $<f_0,...,f_{k-1}>$ is a k-ary world for every $k \geq 1$.

Kripke structures can contain worlds with duplicate truth assignments, such that two distinct worlds $v_1$ and $v_2$, where $v_1$ and $v_2$ have identical truth assignments, are both accessible from world *w*. In a modal structure, identical k-ary worlds are collapsed into one, so that every level k contains only distinct k-ary worlds. Modal structures can be drawn as trees, and the relationship between Kripke and modal structures can be clarified by showing the transformation from a Kripke structure to a modal structure drawn as a tree. Figure 2-1 shows a three-step transformation from a Kripke structure to a modal structure, for world *w* of the Kripke structure. The Kripke structure contains five worlds, of which $v_1$, $v_2$, and $v_3$ have identical truth assignments. In the first step, the Kripke structure is "unraveled" to create a *Kripke tree* rooted at world *w*. In step 2, *collapsed Kripke trees* of successive depths are obtained from the original tree by cutting off the tree at each depth and recursively, from the leaves to the root, collapsing any duplicate subtrees. The collapsed Kripke tree of depth 1 in the figure, for example, has only one leaf because worlds $v_1$, $v_2$, and $v_3$ are collapsed into one. Similarly, the collapsed Kripke tree of depth 2 has only two subtrees because the subtrees rooted at $v_1$ and $v_2$ are collapsed into one. The

[3] Modal structures are actually designed for multi-agent logics. However, since the extensions described in this paper are all to single-agent logics, and since the extension to a multi-agent structure is a straightforward one, the single-agent version is described here.

[4] Kripke structures assume an infinite set of primitive propositions.

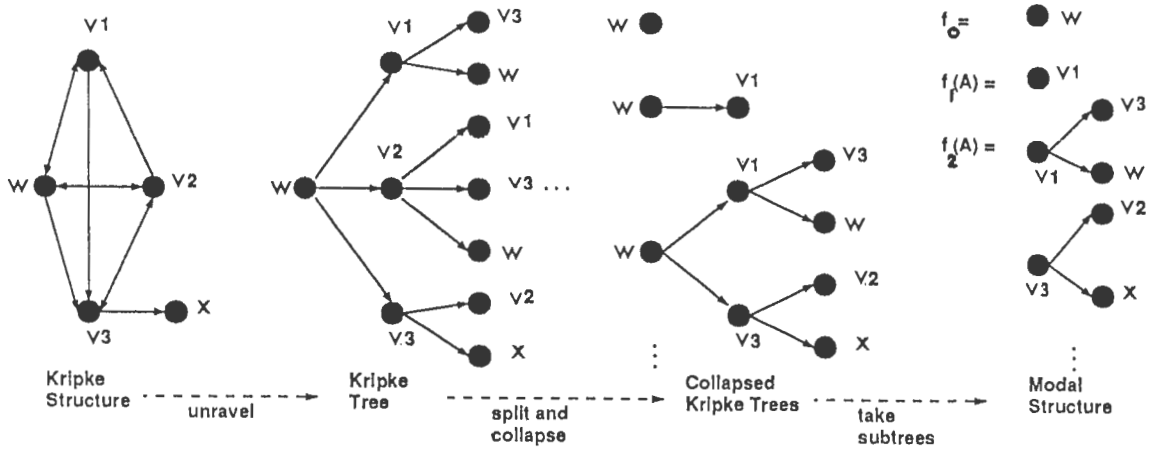



**Figure 2-1:** Three-Step Transformation from a Kripke Structure to a Modal Structure

final step of the transformation produces the modal structure for world *w*. At level 0, $f_0$ is the truth assignment $\pi$ for world *w*. At every level k above level 0, $f_k(\mathbf{A})$ is the set of depth-(k-1) subtrees of the root *w* in the collapsed depth-k Kripke tree. Each such subtree is represented by $\langle g_0,g_1,\ldots,g_{k-1}\rangle$, where $g_0$ is the truth assignment at the root of the subtree, and $g_m(\mathbf{A})$ is the set of depth-(m-1) subtrees of the collapsed Kripke subtree rooted at world $g_0$, for $1 \le m \le k-1$. Notice that each subtree at every level of the modal structure is itself a collapsed Kripke tree for a modal structure representing the world at its root. $g_{k-1}(\mathbf{A})$ is called the *suffix* of the subtree rooted at $g_0$, and contains all the depth-(k-2) subtrees of $g_0$.

Because of this collapsing of duplicate subtrees, a single modal structure corresponds to an infinite number of Kripke structures. The collapsing of duplicate subtrees together with the use of a finite set of primitive propositions ensures that each level contains a finite set of k-ary worlds. This feature, along with the clear separation of levels of meta-belief, makes modal structures more suitable than Kripke structures for representing particular states of knowledge and belief. In fact, a particular state of belief corresponds to exactly one modal structure, but to an infinite number of Kripke structures.

The language **L** consists of a set **P** of primitive propositions (infinite for Kripke structures, but fixed and finite for modal structures), the connectives $\wedge$ and $\neg$, and the modal operator B, where B$\alpha$ is read "the agent believes $\alpha$". The connectives $\vee$, $\supset$, and $\equiv$ are introduced by definition. The *depth* of a sentence of **L** is the deepest nesting of modal operators in the sentence. For example, the sentence B(Bp $\wedge$ B(Bp $\vee$ q)) has depth 3, where p,q $\in$ **P**. The support relations for logics modeled by Kripke structures are shown below. They tell how to determine the truth of any sentence of **L** at a world *w* in a Kripke structure M from the truth assignment to the primitive propositions at that world. $\models$ means "supports the truth of" and $\not\models$ means "does not support the truth of". p is a primitive proposition, and $\alpha$ and $\beta$ are formulas of **L**.

1. M,*w* $\models$ p iff p is true at *w* under truth assignment $\pi$.
2. M,*w* $\models \sim\alpha$ iff M,*w* $\not\models \alpha$
3. M,*w* $\models \alpha \wedge \beta$ iff M,*w* $\models \alpha$ and M,*w* $\models \beta$.
4. M,*w* $\models$ B$\alpha$ iff M,*v* $\models \alpha$ for all *v* such that *wRv*.

A sentence $\alpha \in$ **L** is *satisfied* at a world *w* in Kripke structure M if M,*w* $\models \alpha$, and $\alpha$ is *valid* in M (written "$\models \alpha$") if $\alpha$ is satisfied at every $w \in G$.

The support relations for logics modeled by modal structures is now given.[5] [Fagin, Halpern, and Vardi 84] proves that the satisfiability of a sentence of depth k is confirmed at level k in a modal structure. p is a primitive proposition, and $\alpha$ and $\beta$ are formulas of **L**.

1. $\langle f_0,\ldots,f_k\rangle \models$ p iff p is true under truth assignment $f_0$.
2. $\langle f_0,\ldots,f_k\rangle \models \sim\alpha$ iff $\langle f_0,\ldots,f_k\rangle \not\models \alpha$.
3. $\langle f_0,\ldots,f_k\rangle \models \alpha \wedge \beta$ iff $\langle f_0,\ldots,f_k\rangle \models \alpha$ and $\langle f_0,\ldots,f_k\rangle \models \beta$.
4. $\langle f_0,\ldots,f_k\rangle \models$ B$\alpha$ iff $\langle g_0,\ldots,g_{k-1}\rangle \models \alpha$ for every $\langle g_0,\ldots,g_{k-1}\rangle \in f_k(\mathbf{A})$, where $\alpha$ is of depth k-1.

The truth of all formulas that contain no modal operators is confirmed at level 0 of the modal structure, while the truth of formulas with k nested modal operators is confirmed at level k. A depth-k sentence $\alpha$ is *satisfied* at a modal structure f (written "f $\models \alpha$") if $\langle f_0,\ldots,f_k\rangle \models \alpha$, and $\alpha$ is *valid* if it is satisfied at every modal structure.

Each modal structure corresponds to a single state of a Kripke structure, whose truth assignment is restricted to a finite set of propositions, together with its accessibility information. Moreover, each Kripke structure corresponds to a collection of modal structures, such that exactly the same set of sentences is satisfiable, and thus valid, in each model. The following theorem makes the equivalence explicit.

> **Theorem 5:** [Fagin and Vardi 85] To every Kripke structure M and state *s* in M, there corresponds a modal structure $f_{M,s}$ such that M,*s* $\models \alpha$ iff $f_{M,s} \models \alpha$, for every formula $\alpha$. Conversely, there is a Kripke structure M such that

[5] The symbols $\models$ and $\not\models$ are used in this paper to define the support relations for several logics; the symbols have distinct definitions for each logic. Because the meaning is clear from the context, however, no confusion arises.

for every modal structure $f$ there is a state $s_f$ in M such that $f \models \alpha$ iff $M, s_f \models \alpha$, for every formula $\alpha$.

*Belief structures* [Fagin and Vardi 85] are simply modal structures constrained to model the logic weak S5. Figure 2-2 shows the first three levels of a sample belief structure and the Kripke structure that it corresponds to.

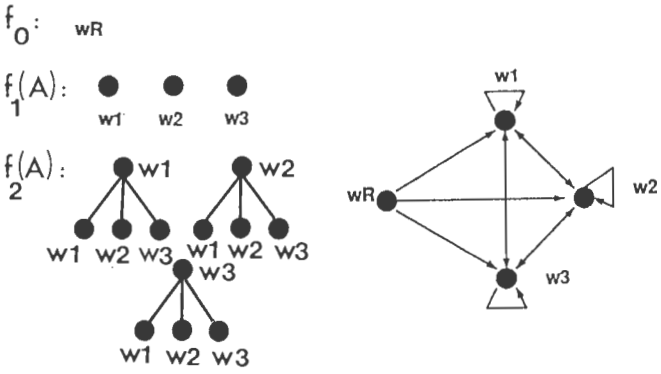



**Figure 2-2:** A Belief Structure and a Corresponding Kripke Structure

The world $w_R$ at level 0 is the "real world". At level 1, worlds $w_1$, $w_2$, and $w_3$ are the worlds compatible with the agent's beliefs, and assigned to it by $f_1$. At level 2, $f_2$ assigns to the agent a set of three 2-ary worlds that are compatible with its beliefs about its beliefs about the world.

There are three semantic restrictions on belief structures, the first of which applies to all modal structures.

T1) **Basic Restriction:** $\langle g_0,\ldots,g_{k-2}\rangle \in f_{k-1}(\mathbf{A})$ iff there is a $g_{k-1}$ such that $\langle g_0,\ldots,g_{k-2}, g_{k-1}\rangle \in f_k(\mathbf{A})$, for $k \geq 2$.

That is, each (k-1)-ary world forms the *prefix* of some k-ary world at level k, and each k-ary world at level k has as its prefix some (k-1)-ary world from the previous level. In terms of trees, this restriction says that the leaves of subtrees at level k-1 are the parents of leaves at level k, and conversely, that the leaves at level k have as parents the leaves of subtrees at level k-1. Intuitively, each level extends the agent's lower-level beliefs. In Figure 2-2, each world at level 1 is the prefix of some 2-ary world at level 2, and the leaves at level 2 are the children of worlds at level 1.

T2) **Full Introspection:** if $\langle g_0, \ldots, g_{k-1}\rangle \in f_k(\mathbf{A})$, then $g_{k-1}(\mathbf{A}) = f_{k-1}(\mathbf{A})$, for $k \geq 2$.

Agents under weak S5 are fully introspective, so the worlds accessible from those at the previous level are exactly all the worlds at the previous level. That is, an agent knows exactly what it believes and doesn't believe at the previous level. T2 corresponds to the transitive and Euclidian restrictions on Kripke structures.[6] In Figure 2-2, the suffix of each 2-ary world (the set of leaves of each subtree) at level 2 contains all the worlds at level 1. Because weak S5 does not require beliefs to correspond to "reality", $w_R$ is not required to appear at level 1.

T3) **Consistency:** $f_k(\mathbf{A})$ is nonempty for $k \geq 1$.

An agent's beliefs must be consistent. Since worlds are consistent, if there is some world at every level, then the agent's beliefs are consistent. T3 corresponds to the serial restriction on Kripke structures.

A belief structure has an infinite number of levels. After some level, however, an agent will have no new information to believe, and the higher levels will contain only the worlds compatible with the beliefs the agent gains by introspecting about its beliefs (and lack of beliefs) at the previous levels.[7] A level which contains no information not in the previous level is called a *no-information extension* of the previous level [Fagin, Halpern, and Vardi 84].

## 3. BLK-Structures and BL4-Structures

[Levesque 84a] describes a single-agent logic of implicit and explicit belief (called **BL** in this paper), where explicit beliefs are those "actively held" by the agent, and implicit beliefs are the logical consequences of the explicit beliefs. Explicit beliefs are not required to be consistent. As well, an agent is not required to believe explicitly all tautologies, the consequences of its explicit beliefs, or certain statements that are logically equivalent to its explicit beliefs. The language of **BL** consists of a set of primitive propositions **P**, the connectives $\wedge$ and $\neg$, and the modal operators B and L. B$\alpha$ is read "the agent explicitly believes $\alpha$" and L$\alpha$ is read "the agent implicitly believes $\alpha$". The connectives $\vee$, $\supset$ and $\equiv$ are introduced by definition. No nesting of modal operators is allowed, so the agent possesses no meta-beliefs.

The semantics of **BL** is given in terms of *situations* as well as worlds: A situation assigns *true*, *false*, both or neither to propositions. A world then is a complete and consistent situation, or alternatively, a situation is a partial, possibly inconsistent, world. The semantics is given in terms of a *model-structure*, called a ***BL-model*** in this paper to avoid confusion with modal structures. A **BL**-model M is a 4-tuple $\langle \mathbf{S},\mathbf{B},\mathbf{T},\mathbf{F}\rangle$, where **S** is the set of all situations, **B** is the set of situations that correspond to the agent's explicit beliefs, and **T** and **F** map primitive propositions to sets of situations in which they are respectively true and false. The agent's implicit beliefs are modeled by the set of worlds **W(B)**, which is *compatible* with the belief set **B**. A world $w$ is *compatible* with a situation $s$ if all propositions that are true (false) at $s$ are true (false) at $w$. Every $w \in \mathbf{W(B)}$ is compatible with some $s \in \mathbf{B}$, and all explicit beliefs are also implicit beliefs by the definition of compatibility. The support relations for a situation in a **BL**-model M are shown below. $\models_T$ means "supports the truth of", $\models_F$ means "supports the falsity of", and $\not\models_T$ means "does not support the truth of".

[6] The transitive and Euclidean restrictions may be specified individually by replacing the equality in T2 with a subset ($\subseteq$) relation for the transitive restriction and a superset ($\supseteq$) relation for the Euclidean restriction [Vardi 85].

[7] If there are several agents, the situation is more complex; see [Fagin, Halpern, and Vardi 84] for details.

1. $M,s \models_T p$ iff $s \in \mathbf{T}(p)$.
   $M,s \models_F p$ iff $s \in \mathbf{F}(p)$.
2. $M,s \models_T (\alpha \vee \beta)$ iff $M,s \models_T \alpha$ or $M,s \models_T \beta$.
   $M,s \models_F (\alpha \vee \beta)$ iff $M,s \models_F \alpha$ and $M,s \models_F \beta$.
3. $M,s \models_T (\alpha \wedge \beta)$ iff $M,s \models_T \alpha$ and $M,s \models_T \beta$.
   $M,s \models_F (\alpha \wedge \beta)$ iff $M,s \models_F \alpha$ or $M,s \models_F \beta$.
4. $M,s \models_T \sim\alpha$ iff $M,s \models_F \alpha$.
   $M,s \models_F \sim\alpha$ iff $M,s \models_T \alpha$.
5. $M,s \models_T B\alpha$ iff for every situation $t \in \mathbf{B}$, $M,t \models_T \alpha$.
   $M,s \models_F B\alpha$ iff $M,s \not\models_T B\alpha$.
6. $M,s \models_T L\alpha$ iff for every world $w \in \mathbf{W(B)}$, $M,w \models_T \alpha$.
   $M,s \models_F L\alpha$ iff $M,s \not\models_T L\alpha$.

The truth of a sentence $\alpha$ is verified only at worlds in **W(B)**. In particular, $\alpha$ is *valid* (written "$\models\alpha$") if it is satisfied at all worlds in all **BL**-models. $\alpha$ is *satisfied* at a world $w \in \mathbf{W(S)}$ in a **BL**-model M (written "$M,w \models \alpha$") if $M,w \models_T \alpha$.

The semantics of **BL** can be given equivalently in terms of a collection of modified modal structures called **BL**-*structures*, where each **BL**-structure models one situation (or world) in a **BL**-model. Since **BL** permits no meta-knowledge, **BL**-structures contain only two levels. Level 0 contains an assignment $f_0$ of *true*, *false*, both or neither to the propositions in **P**; it represents the "actual situation", where agent **A** is located. At level 1, **A** is assigned both a set of situations $s_1(\mathbf{A})$ and a set of worlds $w_1(\mathbf{A})$, which correspond to **B** and **W(B)**, respectively. Each world in $w_1(\mathbf{A})$ is compatible with some situation in $s_1(\mathbf{A})$, just as **W(B)** and **B** are compatible in **BL**-models. **BL**-structures are described in detail in [Hamilton 87]. The logics **BLK** and **BL4**, which are described next, extend **BL** to allow meta-beliefs. The modal structures which provide their semantics are described in detail.

[Lakemeyer 87] extends **BL** to allow meta-beliefs in his logic **BLK**. The language used is that of **BL**, except that nested beliefs are allowed; the only restriction on nesting is that no L may appear in the scope of a B, so the agent cannot hold explicit beliefs about its implicit beliefs. This is a syntactic restriction on the language which does not affect the semantic model.

Lakemeyer introduces the constraint that an agent cannot introspect explicitly about its beliefs. To achieve this, he replaces the belief set **B** in the **BL**-model with two accessibility relations **R** and $\overline{\mathbf{R}}$, for positive and negative explicit beliefs, respectively. Positive explicit beliefs are sentences with an initial unnegated B operator (*e.g.*, $B\alpha$ and $B(B\alpha \vee \sim B\beta)$). Negative explicit beliefs are sentences with an initial negated B operator (*e.g.*, $\sim B\alpha$ and $\sim B(B\alpha \vee B\beta)$). The intuition is that an agent **A**'s beliefs are confirmed by looking in one set of situations (those accessible through **R**), and disconfirmed by looking in another (those accessible through $\overline{\mathbf{R}}$). The two accessibility relations coincide at worlds, which model **A**'s implicit beliefs. Thus, $w\mathbf{R}s$ iff $w\overline{\mathbf{R}}s$, for a world $w$ and a situation $s$. This ensures that and agent's beliefs are consistent at a world, although they may not be at a situation. Unlike in **BL**-models, both worlds and situations are reachable through **R** and $\overline{\mathbf{R}}$. Implicit beliefs are fully introspective, as in weak S5, but are not required to be consistent. A ***BLK-model*** is then a 5-tuple $\langle \mathbf{S,T,F,R,}\overline{\mathbf{R}} \rangle$, where **S**, **T**, and **F** are as in **BL**-models, and **R** and $\overline{\mathbf{R}}$ are the accessibility relations.

Transitive and Euclidean restrictions on worlds in *BLK*-models give the agent full implicit introspective powers over its implicit and explicit beliefs. For all worlds $w$ and $v$ and situations $s \in \mathbf{S}$,

if $w\mathbf{R}v$ and $v\mathbf{R}s$, then $w\mathbf{R}s$ (transitive), and

if $w\mathbf{R}v$ and $w\mathbf{R}s$, then $v\mathbf{R}s$ (Euclidean).

The support relations for a **BLK**-model are the same as those for a **BL**-model except for those that interpret sentences of the form $B\alpha$ and $L\alpha$. Intuitively, $B\alpha$ is supported at a situation if $\alpha$ is supported at all situations accessible through **R**; $B\alpha$ is not supported at a situation if $\alpha$ is not supported at some situation accessible through $\overline{\mathbf{R}}$. $L\alpha$ is true if $\alpha$ is true in all worlds accessible through **R**. $\alpha$, of course, can itself contain modal operators. The two new support relations follow.

5. $M,s \models_T B\alpha$ iff for all $t$, if $s\mathbf{R}t$ then $M,t \models_T \alpha$.
   $M,s \models_F B\alpha$ iff for some $t$, $s\overline{\mathbf{R}}t$ and $M,t \not\models_T \alpha$.
6. $M,s \models_T L\alpha$ iff for all worlds $w$, if $s\mathbf{R}w$ then $M,w \models_T \alpha$.
   $M,s \models_F L\alpha$ iff $M,s \not\models_T \alpha$.

Satisfiability and validity are defined as for **BL**-models.

The semantics of **BLK** can be described equivalently in terms of a collection of extended **BL**-structures called ***BLK-structures***; each **BLK**-structure is equivalent to a situation in a **BLK**-model, along with its accessibility information. Level 0 of a **BLK**-structure contains an assignment of truth values that describes the "actual situation", as in **BL**-structures. Each ensuing level associates with agent **A** two sets of situations, corresponding to the situations accessible to **A** through **R** and $\overline{\mathbf{R}}$. The formal definition is shown below. As before, we assume a fixed, finite set of primitive propositions **P**, and a single agent **A**.

**Definition 6:** $s_0$: $\mathbf{P} \rightarrow$ Powerset({*true*,*false*}) is a *0th-order situation truth assignment*. $w_0$: $\mathbf{P} \rightarrow$ {*true*,*false*} is a *0th-order world truth assignment*.

Intuitively, $s_0$ assigns *true*, *false*, both or neither to the propositions in **P**, while $w_0$ assigns either *true* or *false* to them. $f_0$ is the $s_0$ that describes the "real situation" at level 0 of a BL-structure.

**Definition 7:** $\langle s_0 \rangle$ is a *1-ary situation* (abbreviated *situation*), and $\langle w_0 \rangle$ is a *1-ary world* (abbreviated *world*).

A world is a situation. **T**(p) is the set of all situations at which p is true, and **F**(p) is the set of all situations at

which p is false, for all $p \in \mathbf{P}$. Every world appears in exactly one of $\mathbf{T}(p)$ or $\mathbf{F}(p)$, but each situation can appear in one, both, or neither.

Let $\mathbf{S_1}$ be the set of all 1-ary situations.

**Definition 8:** $f_1: \{\mathbf{A}\} \rightarrow 2^{\mathbf{S_1}}$ is a *1st-order positive situation assignment.* $\bar{f}_1: \{\mathbf{A}\} \rightarrow 2^{\mathbf{S_1}}$ is a *1st-order negative situation assignment.* $[f_1, \bar{f}_1]$ is a *1st-order BLK-assignment.*

Intuitively, $f_1$ associates with the agent a set of 1-ary situations compatible with its explicit positive beliefs about the world. Intuitively, $\bar{f}_1$ associates with the agent a set of situations compatible with its explicit negative beliefs about the world.

**Definition 9:** $\langle f_0,[f_1,\bar{f}_1],\ldots,[f_{k-1},\bar{f}_{k-1}]\rangle$ is called a *k-ary BLK-situation.* If $\langle f_0\rangle$ is a world, $\langle f_0,[f_1,\bar{f}_1],\ldots,[f_{k-1},\bar{f}_{k-1}]\rangle$ is also called a *k-ary BLK-world.*

Let $\mathbf{S_k}$ be the set of all k-ary *BLK*-situations. Then $k^{th}$-order positive and negative situation assignments are specified as follows.

**Definition 10:** $f_k: \{\mathbf{A}\} \rightarrow \text{Powerset}(\mathbf{S_k})$ is a *$k^{th}$-order positive situation assignment.* $\bar{f}_k: \{\mathbf{A}\} \rightarrow 2^{\mathbf{S_k}}$ is a *$k^{th}$-order negative situation assignment.*

Intuitively, $f_k$ assigns to the agent a set of k-ary *BLK*-situations which are compatible with its positive explicit depth-k beliefs, and $\bar{f}_k$ assigns to each agent a set of k-ary *BLK*-situations which are compatible with its negative explicit depth-k beliefs. $f_k(\mathbf{A})$ is the set of k-ary *BLK*-situations assigned to the agent by $f_k$, and $\bar{f}_k(\mathbf{A})$ is the set of k-ary *BLK*-situations assigned to the agent by $\bar{f}_k$, both at level k. $W(f_k(\mathbf{A}))$ is the set of k-ary *BLK*-situations $\langle g_0,[g_1,\bar{g}_1],\ldots,[g_{k-1},\bar{g}_{k-1}]\rangle \in f_k(\mathbf{A})$ in which $\langle g_0\rangle$ is a world. The *suffix* of a k-ary *BLK*-situation $\langle f_0,[f_1,\bar{f}_1],\ldots,[f_{k-1},\bar{f}_{k-1}]\rangle$ is the set of depth-(k-2) subtrees assigned to the agent by the $(k-1)^{th}$-order *BLK*-assignment $[f_{k-1},\bar{f}_{k-1}]$; $f_{k-1}(\mathbf{A})$ is called the *positive suffix*, and $\bar{f}_{k-1}(\mathbf{A})$ is called the *negative suffix.*

**Definition 11:** The infinite sequence $\langle f_0, [f_1,\bar{f}_1], [f_2,\bar{f}_2],\ldots\rangle$ is a *BLK-structure* if every prefix $\langle f_0, [f_1,\bar{f}_1],\ldots,[f_{k-1},\bar{f}_{k-1}]\rangle$ is a k-ary *BLK*-situation for every $k > 0$, and the structure satisfies the semantic restrictions S1, S2, and S3 given below.

Figure 3-1 shows the first three levels of a sample **BLK**-structure and a corresponding **BLK**-model. The hollow circles represent situations, while the dark circles represent worlds. The solid and dashed lines represent **R** and $\overline{\mathbf{R}}$, respectively. The **BLK**-structure models world w1 of the **BLK**-model. At level 1 of the **BLK**-structure, $f_1(\mathbf{A})$ contains the situations compatible with the agent's positive beliefs about the world (those accessible through **R** from w1), while $\bar{f}_1(\mathbf{A})$ contains the situations compatible with the agent's negative beliefs about the world (those accessible through $\overline{\mathbf{R}}$ from w1).[8] At level 2, $f_2(\mathbf{A})$ contains three subtrees, each one rooted at one of the situations in $f_1(\mathbf{A})$. The leaves of each subtree are the situations accessible through **R** and $\overline{\mathbf{R}}$ from the root, so that $f_1(\mathbf{A})$ represents beliefs of the form $BB\alpha$ and $B\sim B\alpha$. The same is true of $\bar{f}_2(\mathbf{A})$, which models beliefs of the form $\sim BB\alpha$ and $\sim B\sim B\alpha$. It is possible for $f_2(\mathbf{A})$ ($\bar{f}_2(\mathbf{A})$) to contain more than one subtree rooted at a situation in $f_1(\mathbf{A})$ or $\bar{f}_1(\mathbf{A})$; this could happen if the *BLK*-model contained two situations with identical truth assignments but different accessible situations.

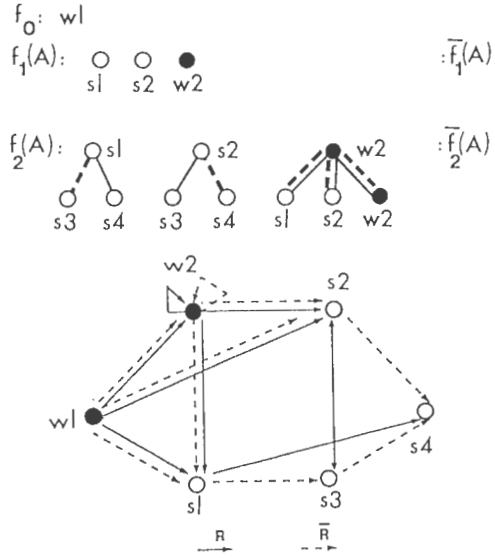



**Figure 3-1:** A **BLK**-Structure and a Corresponding **BLK**-Model

Three semantic restrictions are needed to enforce the properties of **BLK**. S1 and S3 are closely related to T1 and T2 of belief structures, and S2 enforces the coincidence of **R** and $\overline{\mathbf{R}}$ at worlds.

**S1) Basic Restriction:** $\langle g_0, [g_1,\bar{g}_1],\ldots, [g_{k-2},\bar{g}_{k-2}]\rangle \in f_{k-1}(\mathbf{A})$ ($\bar{f}_{k-1}(\mathbf{A})$) iff there exists a $[g_{k-1},\bar{g}_{k-1}]$ such that $\langle g_0, [g_1,\bar{g}_1],\ldots, [g_{k-2},\bar{g}_{k-2}], [g_{k-1},\bar{g}_{k-1}]\rangle \in f_k(\mathbf{A})$ ($\bar{f}_k(\mathbf{A})$) for $k \geq 2$.

S1 says simply that each (k-1)-ary **BLK**-situation at level k-1 forms the prefix for at least one k-ary **BLK**-situation at level k, and the prefix of each k-ary **BLK**-situation at level k is equivalent to some (k-1)-ary **BLK**-situation at level k-1. Intuitively, as in belief structures, each level extends the agent's lower-level beliefs. In Figure 3-1, for example, each of the three elements of $f_1(\mathbf{A})$ forms the prefix (the root in this example) of some 2-ary

[8] In this example, $\bar{f}_2(\mathbf{A})$ contains the same subtrees as $f_2(\mathbf{A})$, because $\langle f_0\rangle$ is a world. This is not true in general.

**BLK**-situation in $f_2(\mathbf{A})$, and the prefix of each 2-ary **BLK**-situation in $f_2(\mathbf{A})$ is some element of $f_1(\mathbf{A})$. The same is true of $\bar{f}_1(\mathbf{A})$ and $\bar{f}_2(\mathbf{A})$.

S2) ***w*R*s* iff *w*$\bar{\mathbf{R}}$*s*:** If $<f_0>$ is a world in the **BLK**-structure $<f_0, [f_1,\bar{f}_1],...>$ then $f_k(\mathbf{A}) = \bar{f}_k(\mathbf{A})$ for all $k \geq 1$.

S2 corresponds to the **BLK**-model restriction *w*R*s* iff *w*$\bar{\mathbf{R}}$*s*, which ensures that the agent's positive and negative beliefs are consistent *at a world*. This restriction must therefore be enforced whenever the *BLK*-structure models a world. In a **BLK**-structure, a positive depth-k sentence is modeled by the subtrees in $f_k(\mathbf{A})$, while the corresponding negative sentence is modeled by the subtrees in $\bar{f}_k(\mathbf{A})$. So if $<f_0>$ is a world, the positive and negative sets at each level must be the same. Restriction S2 is illustrated in Figure 3-1; notice that it holds not only in the main **BLK**-structure, but also within the 2-ary **BLK**-world in $f_2(\mathbf{A})$ and $\bar{f}_2(\mathbf{A})$.

S3) **Full Implicit Introspection:** For $<g_0>$ a world, if $<g_0,[g_1,\bar{g}_1],...,[g_{k-1},\bar{g}_{k-1}]> \in f_k(\mathbf{A})$ then $g_{k-1}(\mathbf{A}) = f_{k-1}(\mathbf{A})$ for $k \geq 2$.

S3 enforces the BLK introspection axiom, which gives full implicit introspection over implicit and explicit beliefs. It corresponds to the transitive and Euclidean restrictions on **R** in *BLK*-models, and says that the positive suffixes ($g_k(\mathbf{A})$) of k-ary BLK-worlds at level k must contain all the (k-1)-ary BLK-situations at level k-1. By restriction S2, the negative suffixes ($g_k(\mathbf{A})$) contain the same worlds, so the agent's implicit beliefs are fully introspective.

The support relations for **BLK**-structures are shown below. $p$ is a primitive proposition, and $\alpha$ is a sentence. As in **BLK**-models, $|=_T$ means "supports the truth of", $|=_F$ means "supports the falsity of", and $|\neq_T$ means "does not support the truth of".

1. $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T p$ iff $<f_0> \in \mathbf{T}(p)$.
   $<f_0,[f_1,f_1],...,[f_k,f_k]> |=_F p$ iff $<f_0> \in \mathbf{F}(p)$.
2. $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \sim\alpha$ iff
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \alpha$.

   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \sim\alpha$ iff
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \alpha$.
3. $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \alpha \wedge \beta$ iff
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \alpha$ and
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \beta$.

   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \alpha \wedge \beta$ iff
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \alpha$ or
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \beta$.
4. $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \alpha \vee \beta$ iff
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \alpha$ or
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \beta$.

   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \alpha \vee \beta$ iff
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \alpha$ and
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F \beta$.
5. $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T B\alpha$ iff
   $<g_0,[g_1,\bar{g}_1],...,[g_{k-1},\bar{g}_{k-1}]> |=_T \alpha$ for all
   $<g_0,[g_1,\bar{g}_1],...,[g_{k-1},\bar{g}_{k-1}]> \in f_k(\mathbf{A})$.

   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F B\alpha$ iff
   $<g_0,[g_1,\bar{g}_1],...,[g_{k-1},\bar{g}_{k-1}]> |\neq_T \alpha$ for some
   $<g_0,[g_1,\bar{g}_1],...,[g_{k-1},\bar{g}_{k-1}]> \in \bar{f}_k(\mathbf{A})$.
6. $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T L\alpha$ iff
   $<w_0,[g_1,\bar{g}_1],...,[g_{k-1},\bar{g}_{k-1}]> |=_T \alpha$ for all
   $<w_0,[g_1,\bar{g}_1],...,[g_{k-1},\bar{g}_{k-1}]> \in W(f_k(\mathbf{A}))$.

   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_F L\alpha$ iff
   $<f_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |\neq_T \alpha$.

A sentence $\alpha$ of depth k is modeled at level k of the **BLK**-structure. Specifically, the *BLK*-structure $f = <w_0,[f_1,\bar{f}_1],...>$ is said to *satisfy* sentence $\alpha$ of depth k (written "$f |= \alpha$") if $<w_0,[f_1,\bar{f}_1],...,[f_k,\bar{f}_k]> |=_T \alpha$. $\alpha$ is *valid* (written "$|=\alpha$") if it is satisfied in every **BLK**-structure $<w_0, [f_1,\bar{f}_1],...>$. Validity thus corresponds to satisfiability at every world in every model of Kripke structures.

The equivalence between **BLK**-models and **BLK**-structures is similar to the equivalence between Kripke structures and belief structures: **BLK**-structures model a single situation, while Lakemeyer's **BLK**-models model collections of situations, such that the same set of sentences is satisfiable in each model.

> **Theorem 12:** [Hamilton 87] To every **BLK**-model $M = <\mathbf{S,T,F,R,\bar{R}}>$ and world $w$ in M, there corresponds a **BLK**-structure $f_{M,w}$ such that $M,w |= \alpha$ iff $f_{M,w} |= \alpha$, for every formula $\alpha$. Conversely, there is a **BLK**-model M such that for every **BLK**-structure $f$ there is a world $w_f$ in M such that $f |= \alpha$ iff $M,w_f |= \alpha$ for every formula $\alpha$.

Lakemeyer presents an axiomatization of **BLK** that is both sound and complete with respect to **BLK**-models. Since **BLK**-models and **BLK**-structures are equivalent, the axiomatization is also sound and complete with respect to **BLK**-structures.

**BLK**-structures, like belief structures, are infinite in height. However, after some finite number of levels, no further information is introduced, and the agent's higher-level beliefs result from its implicit introspection. These implicit beliefs can be propagated upwards through the

levels on the positive side by using a *no-information* **BLK** *extension* similar to the no-information extension of belief structures. Intuitively, the no-information **BLK**-extension $[f_{k+1}, \bar{f}_{k+1}]$ describes the agent's depth-(k+1) beliefs, given that it has no information other than that expressed in its depth-k beliefs. The explicit and implicit no-information extensions begin at the same level, since explicit beliefs are implicit beliefs. The suffixes of positive (k+1)-ary **BLK**-*worlds* in the no-information **BLK**-extension contain the entire previous level, to model the fact that the agent's implicit beliefs at that level are a result of introspecting about its beliefs at the previous level. The suffixes of positive (k+1)-ary **BLK**-*situations* contain all possible (k+1)-ary **BLK**-worlds in order to model the fact that the agent has no positive beliefs about either its beliefs or its non-beliefs at the previous level. The suffixes of all negative (k+1)-ary **BLK**-situations and **BLK**-worlds contain all possible (k+1)-ary **BLK**-worlds in order to model the fact that the agent does not believe anything at that level. The formal definition of the no-information **BLK**-extension is in [Hamilton 87].

[Lakemeyer 87] transforms **BLK** into a system called **BL4**, which is like **BLK** except that the agent's explicit positive beliefs are subject to positive introspection. This is done by making **R** transitive for situations as well as worlds (if $s\mathbf{R}t$ and $t\mathbf{R}u$, then $s\mathbf{R}u$), and adding the balancing restriction that if $s\bar{\mathbf{R}}t$ and $t\mathbf{R}u$, then $s\bar{\mathbf{R}}u$, for all situations $s$, $t$, and $u$. BL4 can be modeled in a ***BL4-structure***, which is a **BLK**-structure with one additional semantic restriction, S4, to enforce explicit positive introspection. The formal definition of S4 is given below. The support relations for **BL4**-structures are exactly those of **BLK**-structures.

S4) **Explicit Positive Introspection:** If $\langle g_0,[g_1,\bar{g}_1],\ldots,[g_{k-1},\bar{g}_{k-1}]\rangle \in f_k(\mathbf{A})$ $(\bar{f}_k(\mathbf{A}))$, then $g_{k-1}(\mathbf{A}) \subseteq f_{k-1}(\mathbf{A})$ $(\bar{f}_{k-1}(\mathbf{A}))$ for all $k \geq 2$.

S4 says that at every level k above level 1, the positive suffixes of k-ary **BL4**-situations in $f_k(\mathbf{A})$ and $\bar{f}_k(\mathbf{A})$ are subsets of $f_{k-1}(\mathbf{A})$ and $\bar{f}_{k-1}(\mathbf{A})$, respectively. This is illustrated in Figure 3-2, which shows the first three levels of a **BL4**-structure and the associated **BL4**-model. As before, the hollow and dark circles represent situations and worlds, respectively. In every subtree of $f_2(\mathbf{A})$, the set $g_1(\mathbf{A})$ (the set of leaves reachable through **R**) is a subset of $f_1(\mathbf{A})$, and similarly, every set $g_1(\mathbf{A})$ in $\bar{f}_2(\mathbf{A})$ is a subset of $\bar{f}_1(\mathbf{A})$. In the subtrees that model worlds $w$ and $v$, the same is true of $\bar{g}_1(\mathbf{A})$ (the set of leaves reachable through $\bar{\mathbf{R}}$). In the subtrees that model the situations $r$ and $t$, $\bar{g}_1(\mathbf{A})$ is not a subset of the sets in the previous level.

Explicit beliefs are extended through a *no-information* **BL4**-*extension* which incorporates restriction S4. In **BL4**-structures, the positive **BL4**-situations that model non-worlds at the upper levels model the beliefs the agent acquires by introspecting explicitly about its positive beliefs, while those that model worlds continue to model the agent's implicit, fully introspective beliefs. Details are in [Hamilton 87].

## 4. KB-Modal-Structures

While the previous sections extended modal structures to allow partial possible worlds and a double accessibility relation, this section extends modal structures to a first-order quantificational setting. [Levesque 84b] and [Levesque 81a] describe a first-order language, KL, that can be used as the representation and query language for a knowledge base (KB), and enables the KB to answer questions about both its domain and its knowledge.[9] The semantics of the language is given in terms of a possible worlds model which gives the KB the power of weak S5, but is extended to allow quantification over variables. This section briefly describes the language KL and its semantic theory, and presents an equivalent representation using an adaptation of belief structures.

KL includes countably infinite sets of predicate and function symbols including the equality predicate "=", constants, individual variables, and parameters, which act as rigid designators in the domain. It includes a single modal operator K, where $K\alpha$ is read "the KB believes that $\alpha$", and the standard first-order logical operators. *Terms* of KL include variables, parameters, and function applications, and *sentences* are defined recursively in terms of predicate applications combined by the logical operators.

A function $v$ assigns every primitive term of KL to a parameter, enabling the KB to know when two terms refer to the same domain entity, or *co-refer*. If $s$ is the set of all primitive sentences that the KB believes to be true, then $[s,v]$ is a ***world-structure***, which models a possible world. A ***KB-model*** $m$ is any non-empty set of world-structures, intuitively, those that are compatible with the world-knowledge of the KB. The accessibility relation on the worlds in $m$ is transitive, Euclidean, and serial, and so the logic has the properties of weak S5.

The semantics of KL can thus be defined in terms of a modified belief structure called a ***KB-structure***, where each **KB**-structure models a single world-structure in a **KB**-*model*. **KB**-structures are exactly the belief structures of Section 2, except that level 0 is redefined to describe first-order worlds instead of propositional ones. The three semantic restrictions on belief structures apply directly to **KB**-structures, and a new semantic restriction, described below, is added. The no-information extension of belief structures can also be applied directly.

Level 0 of each **KB**-structure contains an assignment $v$ of terms to parameters, and a set of primitive sentences $s$ that are true given $v$ at that world-structure. It also contains a domain-mapping function (DMF) $d$ which maps the parameters to the domain entities. In each **KB**-model there is a single implicit DMF which applies over all world-structures, but each **KB**-structure models only a single world-structure, so the mapping must be

[9]The KB actually has *beliefs* about the domain, since its knowledge is not required to be accurate, but it has accurate knowledge of its own knowledge. The terms knowledge and belief will be used interchangeably in this section.

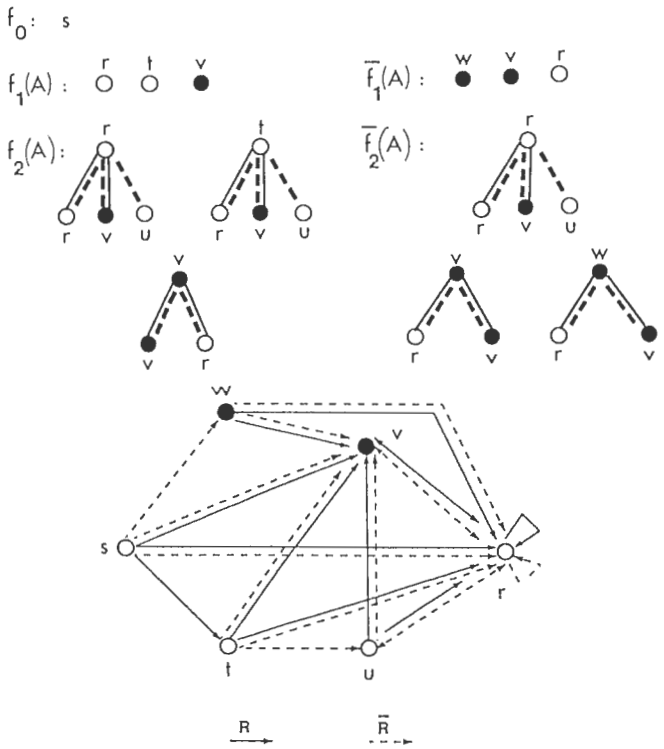



**Figure 3-2:** A Sample **BL4**-Structure and a Corresponding **BL4**-Model

done in every **KB**-structure. The universality of the mapping across the world-structures modeled by **KB**-structures must therefore be enforced; the function *d* will be used in the formulation of a semantic restriction for this purpose on **KB**-structures. Level 0 then consists of a triple $f_0 = [s,v,d]$.

Semantic restriction UD (Universal Domain) ensures that the DMF *d* is the same across all world-structures in the **KB**-structure. Since levels 0 and 1 together contain all the possible worlds in the corresponding **KB**-model, it suffices to compare the DMF of every world-structure at level 1 with that of level 0. Let $\langle f_0 \rangle.d$ represent the domain mapping function of the world-structure $\langle f_0 \rangle = \langle [s,v,d] \rangle$.

**UD)** $\langle g_0 \rangle.d = \langle f_0 \rangle.d$ for all $\langle g_0 \rangle \in f_1(\mathbf{A})$ in **KB**-structure $\langle f_0, f_1, \ldots \rangle$.

The support relations of the **KB**-structures bear a very close resemblance to the support relations of the **KB**-model, given in [Levesque 84a]. In fact, all but the support relation for sentences of the form Kα are identical, with $\langle [s,v,d], f_1,\ldots,f_{k-1} \rangle$ substituted for $m,[s,v,d]$. The one that differs is shown below.

7. $\langle [s,v,d], f_1,\ldots,f_k \rangle \models K\alpha$ if $\langle [s',v',d], g_1,\ldots,g_{k-1} \rangle \models \alpha$ for all $\langle [s',v',d], g_1,\ldots,g_{k-1} \rangle \in f_k(\mathrm{KB})$.

**KB**-structures are equivalent to Levesque's **KB**-models in the same way that belief structures are equivalent to Kripke structures, with the additional restriction that the DMF of the **KB**-model be equivalent to the DMF in the **KB**-structures. The equivalence theorem and its proof are in [Hamilton 87].

## 5. Discussion

This paper has shown how the modal structures of [Fagin, Halpern, and Vardi 84] and [Fagin and Vardi 85] can be adapted to model the non-standard epistemic logics **BL** of [Levesque 84a] and **BLK** and **BL4** of [Lakemeyer 87], and a first-order system, the **KB**-model of [Levesque 81b] and [Levesque 84b]. As well, a three-step transformation from Kripke to modal structures was described to clarify their relationship.

Modal structures do appear to be generally extensible to logics that differ from the standard epistemic logics for which they were designed. But are they, as their inventors claim, more intuitive than the Kripke structures they replace? In belief structures this does seem to be the case; the transitive and Euclidean restrictions on an accessibility relation, for example, seem less obviously related to full introspection than having the same set of accessible worlds at every successive level. Certainly as well, the ability to look at a particular level k of the modal structure to determine the depth-k beliefs of an agent is appealing, compared to the necessity of tracing all paths of length k in a Kripke structure to find the appropriate set of worlds. But the relationship between the worlds, which is easily seen in a Kripke structure, can be obscured in a modal structure by the redundancy at each level. In Figure 2-2, for example, it is clear that in the Kripke structure the worlds accessible from $w_R$ are in an equivalence class. But to draw the same conclusion from the corresponding modal structure, it is necessary to verify at every level that the suffix at each level contains all the worlds of the previous level. The intricate notation of modal structures can also be unwieldy, especially in the more complicated models.

[Fagin and Vardi 85] demonstrates that the finite number of worlds at a level and the ability to verify a depth-k sentence at level k lead to simpler proofs for soundness and completeness, as well as decidability, with modal structures than with Kripke structures. This should be true also for the extensions described in this paper.

Modal structures assume a fixed, finite number of primitive propositions, presumably to model the beliefs of a finite agent. This restriction undoubtedly makes it easier to represent particular states of belief, since the number of unique possible worlds is now finite. The restriction may not always be reasonable, however, since it effectively bars the agent from ever holding beliefs that are not represented in the fixed, finite set. There is no structural reason why modal structures could not be defined for an infinite number of propositions, but the number of possible worlds at a level would not then be guaranteed to be finite, and determining validity in modal structures would no longer be decidable.

**Acknowledgements:** We are grateful to Gerhard Lakemeyer for helpful correspondence during the formulation of BLK-structures, to Alan Mekler for pointing out the relationship between modal structures and trees, to one of the reviewers for detailed, helpful comments, and to Howard Hamilton for technical assistance in the preparation of this paper.

## References


[Fagin and Vardi 85]
Fagin, R. and Vardi, M.Y. "An Internal Semantics for Modal Logic: Preliminary Report", in *Proc. 17th ACM SIGACT Symp. on the Theory of Computing*, pages 305-315. Providence, RI, 1985.

[Fagin, Halpern, and Vardi 84]
Fagin, R., Halpern, J.Y., and Vardi, M.Y. "A Model-Theoretic Analysis of Knowledge: Preliminary Report", in *Proc. 25th IEEE Symp on Foundations of Computer Science*, pages 268-278. 1984.

[Halpern and Moses 85]
Halpern, J. and Moses, Y. "A Guide to the Modal Logics of Knowledge and Belief", in *Proc. of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 480-490. 1985.

[Hamilton 87]
Hamilton, S.J. *Using Modal Structures to Represent Extensions to Epistemic Logics*, Master's thesis, Simon Fraser University, December, 1987.

[Kripke 63]
Kripke, S. "Semantic Analysis of Modal Logic", *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*,9:67-96, 1963.

[Lakemeyer 87]
Lakemeyer, G. "Tractable Meta-Reasoning in Propositional Logics of Belief", In *Proc. of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*. Milan, Italy, 1987.

[Levesque 81a]
Levesque, H.J. *A Formal Treatment of Incomplete Knowledge Bases*, PhD thesis, University of Toronto, 1981.

[Levesque 81b]
Levesque, H.J. "The Interaction with Incomplete Knowledge Bases: A Formal Treatment", in *Proc. of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 240-245. Vancouver, B.C., 1981.

[Levesque 84a]
Levesque, H.J. "A Logic of Implicit and Explicit Belief", Technical Report 32, FLAIR, August, 1984.

[Levesque 84b]
Levesque, H.J. "Foundations of a Functional Approach to Knowledge Representation", *Artificial Intelligence* 23:155-212, 1984.

[Vardi 85]
Vardi, M. "A Model-Theoretical Analysis of Monotonic Knowledge", in *Proc. of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 509-512. Los Angeles, 1985.


## Appendix

Theorem 12 and its proof are shown below.

**Theorem 12:** To every **BLK**-model M = <**S**,**T**,**F**,**R**,$\bar{\mathbf{R}}$> and world $w$ in M, there corresponds a **BLK**-structure $f_{M,w}$ such that M,$w$ |= $\alpha$ iff $f_{M,w}$ |= $\alpha$, for every formula $\alpha$. Conversely, there is a **BLK**-model M such that for every **BLK**-structure $f$ there is a world $w_f$ in M such that $f$ |= $\alpha$ iff M,$w_f$ |= $\alpha$ for every formula $\alpha$.

**Proof:** To show the first part of the theorem, suppose M = <**S**,**T**,**F**,**R**,$\bar{\mathbf{R}}$> is a **BLK**-model. For every situation $s \in$ **S** in M, we construct a **BLK**-structure $f_{M,s}$ = <$f_0$, [$f_1$,$\bar{f}_1$],...>, where $f_0$ is the assignment at situation $s$. Suppose we have constructed a (k-1)-ary **BLK**-situation <$f_0$, [$f_1$,$\bar{f}_1$],...,[$f_{k-2}$,$\bar{f}_{k-2}$]> for each situation $s \in$ **S**. Then $f_{k-1}$(a) = {<$g_0$, [$g_1$,$\bar{g}_1$],...,[$g_{k-2}$,$\bar{g}_{k-2}$]> | $s$**R**$g$}, and $\bar{f}_{k-1}$(a) = {<$h_0$, [$h_1$,$\bar{h}_1$],...,[$h_{k-2}$,$\bar{h}_{k-2}$]> | $s\bar{\mathbf{R}}h$}, where <$g_0$, [$g_1$,$\bar{g}_1$],...,[$g_{k-2}$,$\bar{g}_{k-2}$]> is the k-ary **BLK**-situation constructed for $g$, and <$h_0$, [$h_1$,$\bar{h}_1$],...,[$h_{k-2}$,$\bar{h}_{k-2}$]> is the k-ary **BLK**-situation constructed for $h$. As well, W($f_{k-1}$(a)) = {<$w_0$, [$g_1$,$\bar{g}_1$],...,[$g_{k-2}$,$\bar{g}_{k-2}$]> | $s$**R**$w$}, where $w$ is a situation $g \in$ **S** that is a world. It remains to be shown that M,$w$ |= $\alpha$ iff $f_{M,w}$ |= $\alpha$. The proof is mechanical and is left to the reader.

To show the converse of the theorem, let M = <**S**,**T**,**F**,**R**,$\bar{\mathbf{R}}$> be a **BLK**-model where S = {$s_f$ | $s_f$ = $f_0$ for every **BLK**-structure $f$ = <$f_0$, [$f_1$,$\bar{f}_1$],...>}. Then $s_f$ **R**$g$ iff <$g_0$, [$g_1$,$\bar{g}_1$],...,[$g_{k-2}$,$\bar{g}_{k-2}$]> $\in$ $f_{k-1}$(a), and $s_f$ $\bar{\mathbf{R}}h$ iff <$h_0$, [$h_1$,$\bar{h}_1$],...,[$h_{k-2}$,$\bar{h}_{k-2}$]> $\in$ $\bar{f}_{k-1}$(a) for every k $\geq$ 1, where <$g_0$, [$g_1$,$\bar{g}_1$],...,[$g_{k-2}$,$\bar{g}_{k-2}$]> is the (k-1)-ary **BLK**-situation that corresponds to $g$, and <$h_0$, [$h_1$,$\bar{h}_1$],...,[$h_{k-2}$,$\bar{h}_{k-2}$]> is the (k-1)-ary **BLK**-situation that corresponds to $h$. w($f_{k-1}$)(a)) is defined as above. $f$ |= $\alpha$ iff M,$w_f$|=$\alpha$ by the same reasoning as in part 1.

# A Solution to the Paradoxes of Confirmation

James P. Delgrande

School of Computing Science,
Simon Fraser University,
Burnaby, B.C.,
Canada V5A 1S6

## Abstract

A recalcitrant problem in machine learning concerns the issue of what hypotheses are supported by what evidence. If one accepts, first, that some ground instances describing a domain do indeed support some general hypotheses while others do not and, second, that if an instance supports a hypothesis, it also supports all logical consequences of the hypothesis, then unintuitive and problematic results are immediately forthcoming. For example, a flying bird would support not only the hypothesis that birds fly, but also the contrapositive that all non-flying things are non-birds. This paper argues that, while an instance that supports a hypothesis does also support all consequences of the hypothesis, the notion of classical consequence is too strong to apply to the problem at hand and that instead a weaker notion of consequence should be adopted. A formal system which addresses the problems of evidential support is developed, and it is shown that in this framework the problems do not arise.

## Résumé

Cet article adresse un problème relié au domaine de l'apprentissage automatique concernant les hypothèses qui sont soutenues par un ensemble d'évidence. Si on accepte, premièrement, qu'il y ait des fait qui puissent soutenir des hypothèses générales alors que d'autre ne le font pas, et que, deuxièment, un fait soutenant une hypothèse implique qu'il soutient également toutes les conséquences logiques de cette hypothèse, alors on se retrouve devant une situation problèmatique. Par example, un oiseau volant ne soutiendrait non seulement l'hypothèse "les oiseaux volent" mais aussi l'hypothèse opposée, soit "les choses qui ne volent pas ne sont pas des oiseaux". Nous prétendons dans cette communication qu'un fait soutenant une hypothèse soutient aussi toutes les conséquences logiques de cette hypothèse qu'il soutient. Nous prétendons aussi que la notion de conséquence de la logique du premier ordre est trop forte. On a besoin donc d'une notion de conséquence plus faible. Pour achever ce but, nous présentons dans ce papier un système formel qui adresse les problèmes de soutien d'évidence et nous démontrons que ces problèmes n'arrivent pas avec cette méthode.

## 1. Introduction

The area of inductive learning addresses the problem of forming general hypotheses on the basis of a set of specific facts. For this approach, a learning system receives information concerning a domain of application in the form of facts, or ground atomic formulae. On the basis of this information the learning system induces general statements characterising the domain, and hence hypothesises relations among the known relations in the domain. These hypotheses are phrased independently of any particular individuals. Clearly then, in this approach one must determine when a fact is to count as confirming a hypothesis. In the case of universal statements though, the problem appears to be trivial: a general statement is confirmed by any of its instances. Hence, for example, the statement "all ravens are black" is confirmed by a black raven[1] and falsified by a non-black raven. This observation appears to be so straightforward, that every (to my knowledge) Artificial Intelligence (AI) system that generalises from examples simply accepts this without comment.

Yet matters are not quite as straightforward as they at first seem. If one accepts the condition that whatever evidence confirms a hypothesis also confirms a statement that is logically equivalent to it, then difficulties arise immediately. For example, consider the hypothesis "all ravens are black" and its contrapositive "all non-black things are non-ravens". Clearly a non-black non-raven supports the latter hypothesis. However it also, by the above criterion, supports the former hypothesis. Thus we are led to the unappealing conclusion that the desk that I am writing on, or my pen, supports the hypothesis "all ravens are black". To quote Nelson Goodman, "the prospect of being able to investigate ornithological theories without going out in the rain is so attractive that we know there must be a catch in it" [Goodman 79 p. 70]. This problem is known variously as the "paradoxes of confirmation" or "Hempel's paradox".[2]

[1] More accurately, is confirmed by an instance representing a black raven. Although confirmation is a relation between statements, I will, for simplicity, on occasion talk more loosely about objects confirming statements.

[2] Note though that these problems do not constitute a genuine paradox per se. There is, for example, nothing inconsistent arising from the problem. Rather what we have is a clash with commonsense intuitions. A more appropriate term then is perhaps "the problems of confirmation".

There are various proposals in the philosophical literature for a solution to this problem; none however appears to be unproblematic. In AI, as mentioned, this problem appears to have not been addressed. Clearly though there has been extensive research carried out in machine learning and, in particular in a restriction of inductive learning, learning from examples [Carbonell et al 83]. If a goal of AI is to understand principles underlying intelligent behaviour, then it is essential that such difficulties be addressed. In particular, for the situation at hand, it seems important that some account of the relation between evidence and hypotheses be given.

This paper attempts to provide a starting point for such an account. We argue that the difficulty arises from the use of the notion of classical consequence as a metric for determining which sets of sentences are to be counted as confirmed by a particular piece of evidence. Rather, the approach taken here is to address the problem of evidential support from first principles and, from a set of arguably minimal assumptions, give a formal account of this problem. To do so, we first review a system that has been developed from first principles as a means of analysing formal issues arising from learning from examples. Subsequently we show how it may be applied to the problem at hand. This system proves to be weaker than classical logic; in particular negation is weaker than classical negation. As a means of further analysing this approach to confirmation, an alternative, three-valued semantics is also developed for the system Lastly, we show that the paradoxes of confirmation do not arise in this framework. The approach is argued to be largely independent of the notions of "evidence" and "confirmation" used by a learning system; thus these considerations arguably apply to all systems that learn from examples.

In this paper however we address only the issue of what ground instances are to count as positive instances of hypotheses. Broader issues, such as how hypotheses are to be updated in the presence of new, conflicting instances, are irrelevant to the problem at hand, and are ignored. For a possible account of these further issues, as well as more details on the formal systems presented here, see [Delgrande 87c].

## 2. The Paradoxes of Confirmation

Consider again the hypothesis "all ravens are black". We can represent this in first-order logic as

$$(x)(Raven(x) \supset Black(x)). \qquad 1.$$

This hypothesis is logically equivalent to the following statements:

$$(x)(\neg Black(x) \supset \neg Raven(x)), \qquad 2.$$

$$(x)(\neg Raven(x) \vee Black(x)). \qquad 3.$$

Intuitively an individual that is both a raven and black (say, $Raven(a) \wedge Black(a)$) confirms this hypothesis. Intuitively also some things, say my right shoe or the Eiffel Tower, do not confirm this hypothesis. However, my right shoe and the Eiffel Tower do appear to confirm statements 2 and 3. If we accept that logically equivalent statements are supported by the same evidence, then we are presented with a dilemma. It appears that we must either allow that all three individuals – individual $a$, my right shoe, and the Eiffel Tower – uniformly confirm statements 1-3, or else we must allow that logically equivalent statements need not necessarily be confirmed by the same evidence. This issue was originally discussed by C.G. Hempel in [Hempel 45]. [Black 66] and [Scheffler 81] contain highly readable and thorough surveys; neither however proposes an unproblematic solution.

One proposed approach to the problem is to claim that hypotheses need to be placed in some "context". Thus, to use Black's example, "all vertebrates are warm-blooded" is a statement about animals, or about vertebrates. On the other hand, the argument goes, the statement "all non-warm-blooded things are non-vertebrates" is not. Hence we might formulate the first statement instead by "relativising" the context to animals:

$$(x)((An(x) \wedge Vert(x)) \supset (An(x) \wedge WB(x))). \qquad 4.$$

The corresponding restricted contrapositive would be:

$$(x)((An(x) \wedge \neg WB(x)) \supset (An(x) \wedge \neg Vert(x))). \qquad 5.$$

In this case an animal that is cold-blooded and invertebrate may well be seen as confirming the original hypothesis. However, 4 is also logically equivalent to

$$(x)((An(x) \vee \neg An(x)) \supset (\neg An(x) \vee \neg Vert(x) \vee (An(x) \wedge WB(x)))). \qquad 6.$$

A positive instance of 6 is:

$$(An(a) \vee \neg An(a)) \wedge (\neg An(a) \vee \neg Vert(a) \vee (An(a) \wedge WB(a))). \qquad 7.$$

Hence any non-animal will satisfy 7 and thus be evidence for 6 and so, by implication, 4. Thus any individual falling outside the "context" provides evidence for the hypothesis. This clearly is as bad as the original dilemma.

An alternative is to attempt to provide some account of how evidence changes one's confidence in a given hypothesis. Such approaches generally begin with the observation that, for example, the class of ravens is (presumably) much smaller that the class of non-black things. That being the case, finding a particular raven to be black should increase one's confidence in the hypothesis much more than finding a particular non-black entity to be non-raven. Thus both instances confirm "all ravens are black"; it's just that one does so much more than the other. However this approach, which boils down to quantifying the notion of "degrees of confirmation", has a number of problems. To begin with, using degrees of confirmation may be quite simply irrelevant to the problem: the problem after all is simply to determine what is to count as a confirming instance. It appears also that a successful characterisation of degrees of confirmation will not lend itself to the problem at hand. In particular, the paradoxes arise even when the cardinality of the antecedent is obviously larger than that of the complement of the consequent. Consider the examples:

All atoms are inanimate, or

All invertebrates are cold-blooded.

In neither case does it seem reasonable to take the pen in front of me (say) as a positive instance. Note also there is good reason to doubt that any coherent account of degrees of confirmation may be obtained [Putnam 79].

A third possible solution, proposed by Black, begins with the claim that the error arises in treating the conditional in "all ravens are black" as the material conditional: arguably it isn't the material conditional that one wants, but rather some other

weaker operator. This seems plausible, since it does seem clear that whatever the connective is in "all ravens are black", it is not material implication. For one thing, it certainly isn't the case that *every* raven is black – albinos, for example, generally are white. [Lewis 73], [Chellas 75] and others have considered the problem of conditionals other than material implication, such as those dealing with counterfactual or other subjunctive statements. [Delgrande 87a] and [Delgrande 88] contain an analysis of the conditional used in naive scientific statements, such as "ravens are black" or "birds fly". In all these cases, the truth of a conditional depends not on the present state of affairs being modelled, but on other states of affairs. So, for example, in Lewis' approach, the truth of the counterfactual $A \Rightarrow B$ depends on the states of affairs most similar to the present one in which $A$ is true.

Applying these approaches to the present problem is not straightforward because the semantics of the underlying systems rests on the notion of "alternative states of affairs" or "possible worlds". Thus for example, in the approach of [Delgrande 87a], while the truth of $Raven(a) \wedge Black(a)$ is determined with respect to the present state of affairs, the truth of "ravens are black" is determined with respect to other "less exceptional" states of affairs that may be quite different from the one presently being modelled. Now this may be fine for expressing the meaning of, say, "ravens are black", or other such conditionals, but it does not lend itself to the problems of evidence and hypotheses. On the one hand, it seems somewhat unusual to say that a particular black raven in *this* state of affairs provides evidence for a statement whose truth is determined relative to *other* states of affairs. On the other hand, if we require that the present state of affairs be one of those used in determining the truth of "ravens are black", then we are forced to have our conditional operator correspond to material implication. This in turn requires that in the present state of affairs *all* ravens must be black, which clearly is unrealistic. Hence to the extent that all extant systems for dealing with variable conditionals rely on a possible worlds semantics, this last suggestion appears inapplicable for our purposes. Note also that this last proposal supposes that the difficulty lies with the conditional. However, the claim of this paper is that the difficulty in fact arises with negation. Assuming that this claim is indeed correct, then it is quite conceivable that the paradoxes of confirmation will also arise with conditionals weaker than the material conditional.

## 3. Approach

The framework within which I wish to address the paradoxes of confirmation is as follows. A hypothesis will be a general, universally quantified statement that is not known to be either true or false, but for which one has justified belief in its truth.[3] A hypothesis then may be of the form $(x)(\alpha(x) \supset \beta(x))$ or simply $\alpha(x) \supset \beta(x)$, where $\alpha(x)$ and $\beta(x)$ will be used to stand for arbitrary wffs of first-order logic, with free variable $x$. We will accept *a priori* that such a hypothesis is confirmed by an instance of the form $\alpha(a) \wedge \beta(a)$. The issue then is one of determining, given that ground instance $e$ confirms the general hypothesis $h$, what other hypotheses $e$ confirms.

Consider for example where we have a hypothesis $\alpha(x)$ that is supported by some set of instances. We know that $\alpha(x)$ is equivalent to $(\alpha(x) \wedge (\beta(x) \vee \neg\beta(x)))$. The instances supporting $\alpha(x)$ then should also provide evidence for $\alpha(x) \wedge (\beta(x) \vee \neg\beta(x))$. However if we assume that the set of instances that support the conjunction of two hypotheses is the intersection of the sets that support the conjuncts, then we may run into difficulties. Conceivably the known instances of $\beta$ and $\neg\beta$ could be disjoint from those of $\alpha$. So even though we know of instances satisfying $\alpha$, we may not know of instances that satisfy $\alpha(x) \wedge (\beta(x) \vee \neg\beta(x))$ and so by most any criteria, would not have evidence for it.[4] This is in contrast to statements such as $\alpha(x) \wedge \beta(x)$ and $\beta(x) \wedge \alpha(x)$ where any instance known to satisfy one conjunction must satisfy the other; hence evidence for one *must* constitute evidence for the other.

So the question of interest is to formally characterise this support relation among hypotheses. Fortunately we can make use of an existing system to address this question. [Delgrande 87b] and [Delgrande 87c] develop a foundational approach to "learning from examples" and "learning by being told" [Carbonell et al 83]; the results presented there are applicable to the present case. In broad outline, the approach is as follows. The domain of application is assumed to be described by a presumably infinite set of ground atomic formulae, formed from presumably infinite sets of individuals and predicates. The truth value of some of these instances are known and constitute the *evidence* that may support hypotheses. For example, we may know that $Raven(opus)$, $Black(opus)$, and $\neg Bird(snoopy)$, but we may know neither $Black(snoopy)$ nor $\neg Black(snoopy)$. Hypotheses will express relations among relations in the domain. For example, "the set of (or extension of) ravens is contained in that of black things" may be one such hypothesis.

By *term* I will mean an entity that denotes a relation in the domain. To each term in a hypothesis we can associate two subsets of the ground instances, consisting of those known to satisfy the term and those known to not. Thus with the term "black raven" we can associate the set of individuals known to be both ravens and black, and the set known to be either non-black or non-raven. These sets then contain all the individuals known to satisfy or to not satisfy the term. Hence for term $t$ in hypothesis $h$ these sets give the individuals that $t$ may contribute as evidence in support (or not) of hypothesis $h$. Note then that we do not restrict the notion of evidence that could be used by a learning system, except to require that a hypothesis have known confirming instances. However we do require that the evidence for a hypothesis be a function of the known satisfying (and known non-satisfying) instances in the terms of the hypothesis.

[3] In this paper however I will not be concerned with what constitutes "justified belief", except for some very general considerations.

[4] Note that if we were to take individuals known to be true of $\alpha$ and determine whether or not they were true of $\beta$, then we could locate individuals known to satisfy $\alpha(x) \wedge (\beta(x) \vee \neg\beta(x))$ and hence locate evidence for $\alpha(x) \wedge (\beta(x) \vee \neg\beta(x))$. However this process is quite independent of the simple determination of which consequences of $h$ are supported by $e$, given that $e$ supports $h$. See [Delgrande 87c] for an examination of how such a determination of instances may be carried out.

These (pairs of) sets in a term interrelate in various ways. In the following section, the algebra of these pairs is derived, and from this algebra the corresponding propositional logic is derived. These systems provide a precise specification of what hypotheses are supported by the same evidence. Given the propositional system, it is a trivial matter to translate sentences from this system into a first-order one.

To summarise then, for the approach I assume only that:

1. The task is to determine what ground instances support what general statements.
2. Some instances support a hypothesis, some do not, and some are irrelevant to the hypothesis.
3. For a hypothesis to be supported, there must be a known satisfying instance of the hypothesis.
4. The evidence for a hypothesis is a function of the known satisfying (and known non-satisfying) instances in the terms of the hypothesis.

## 4. Formal Systems for Evidential Support

### 4.1. An Algebra for Evidential Support

Hypotheses are expressed in a language HL that is analogous to that of elementary set theory, except that operators and relations are subscripted with the character "$h$". In [Delgrande 87b] and [Delgrande 87c] operators for the converse, composition, image, domain, and others are specified. These are not directly relevant to our concerns with the paradoxes of confirmation, and so are not included here. A "ply" operator $\supset_h$ is also introduced [Curry 63]. After we consider the algebra of terms of HL, we will want to also consider the corresponding propositional logic; the ply operator serves as the analogue in the algebra of the material conditional in the logic.

**Definition:** If **P** is the set of known predicate names, then the *terms* of HL are exactly those given by:

1) If $\alpha \in \mathbf{P}$ then $\alpha$ is a term of HL.
2) If $\alpha$ and $\beta$ are terms of HL, then so are $\alpha \cap_h \beta$, $\alpha \cup_h \beta$, $\neg_h \alpha$, and $\alpha \supset_h \beta$.

**Definition:** The *sentences* of HL are exactly given by:

If $\alpha$, $\beta$ are terms of HL, then $\alpha =_h \beta$, $\alpha \subset_h \beta$, $\alpha \subseteq_h \beta$, $\alpha \neq \beta$, $\alpha \not\subset \beta$, $\alpha \not\subseteq \beta \in$ HL.

So, for example,

$$Raven \cup_h Penguin \cup_h Robin \subseteq_h Bird$$

has the reading "the set of ravens, penguins, and robins is hypothesised to be contained in the set of birds".

While this is a somewhat impoverished language compared to that of first-order logic, it is, arguably, the appropriate language for expressing relations among relations in the domain. In addition the systems developed appear to be general and expressive enough to sufficiently illustrate and argue our approach to the paradoxes of confirmation. Hence the arguments given here apply also to more expressive systems. Lastly, there is a clear and easy translation of the systems developed into a full first-order framework.

Given a particular predicate there is a set of individuals (or tuples) which are known to satisfy it and a set of individuals which are known to not. Hence:

**Definition:** For each known predicate symbol P define sets $P_+$ and $P_-$ by:

$P_+ = \{<a_1, \ldots, a_n> \mid P(a_1, \ldots, a_n)$ is known to be true$\}$,

$P_- = \{<a_1, \ldots, a_n> \mid \neg P(a_1, \ldots, a_n)$ is known to be true$\}$.

The sets $P_+$, $P_-$ are called the *known extension* and *known antiextension (respectively) of* $P$.

The known extension and antiextension corresponding to terms in HL can easily be determined. Thus, for example, the *hypothetical intersection* of $P$ and $Q$ is known to contain just those elements that both $P$ and $Q$ are true of, and is known to not contain just those elements that either $P$ or $Q$ is known to not be true of. For the hypothesised operations we obtain the following known extension/antiextension pairs.

**Definition:**

| | |
|---|---|
| Complement: | $\neg_h \alpha$ is $<\alpha_-, \alpha_+>$ |
| Union: | $\alpha \cup_h \beta$ is $<\alpha_+ \cup \beta_+, \alpha_- \cap \beta_->$ |
| Intersection: | $\alpha \cap_h \beta$ is $<\alpha_+ \cap \beta_+, \alpha_- \cup \beta_->$ |
| Ply: | $\alpha \supset_h \beta$ is $<(\neg\alpha_+ \cup \beta_+) \cap (\neg\beta_- \cup \alpha_-), \beta_- \cap \neg\alpha_->$[5] |

The expression $\neg\alpha_+$ is defined as $\mathbf{I} - \alpha_+$ where $\mathbf{I}$ is the set of all known individuals (i.e. individuals represented by constants in the set of ground instances).

Now, the relations that will be of interest to us are not the hypothesised relations between terms, but rather will be known relations among extension/antiextension pairs of terms. Two terms of HL are defined to be (strictly) equal when their known extensions and antiextensions coincide. Containment ($\leq$) is introduced by the usual definition. In [Delgrande 87c], the known relations among terms (e.g. $\alpha = \beta$) are used in part to enforce hypothesised relations among terms (e.g. $\alpha =_h \beta$). However here, where we do not consider the dynamics of hypothesis maintenance, we will consider only the known relations among terms.

**Definition:** For $\alpha$, $\beta$ terms of HL,

$\alpha = \beta$ iff $\alpha_+ = \beta_+$ and $\alpha_- = \beta_-$,

$\alpha \leq \beta$ iff $\alpha \cap_h \beta = \alpha$,

$\alpha < \beta$ iff $\alpha \leq \beta$ and $\alpha \neq \beta$.

These relations then give the evidential relations that are known to hold among terms in a hypothesis. Thus for example, it follows easily from the above expressions that $\alpha \cap_h \beta = \beta \cap_h \alpha$; hence any individuals known to satisfy (or not) $\alpha \cap_h \beta$ will do likewise for $\beta \cap_h \alpha$. The algebra of these relations on the corresponding pairs of sets then provides a formal specification of these relations. This algebra, HLA, is given by

[5] The expression for the ply is actually getting a little ahead of the game: while the expressions for the complement, union, and intersection are derived using clear intuitions, the expression for the ply is derived once postulates for it have been given (below).

HLA = [**H**; $\neg_h$, $\cap_h$, $\cup_h$, $\supset_h$], where, for a set of known individuals **I**, the carrier **H** is given by:

$$\mathbf{H} = \{<a,b> \mid a,b \subseteq \mathbf{I} \text{ and } a \cap b = \varnothing\}.$$

The pair of elements in a member of **H** corresponds to possible individuals known to be in a relation or to not be in the relation. Upper and lower bounds of **H** are defined by:

**Definition:**

$$\mathbf{1} = <\mathbf{I},\varnothing>; \quad \mathbf{0} = <\varnothing,\mathbf{I}>.$$

We obtain the following postulates and rule of inference:

**Postulates:**

P1 $\alpha \cap_h \beta = \beta \cap_h \alpha$ $\quad\quad$ $\alpha \cup_h \beta = \beta \cup_h \alpha$

P2 $\alpha \cap_h (\beta \cap_h \gamma) = (\alpha \cap_h \beta) \cap_h \gamma$
$\alpha \cup_h (\beta \cup_h \gamma) = (\alpha \cup_h \beta) \cup_h \gamma$

P3 $\alpha \cap_h (\alpha \cup_h \beta) = \alpha$ $\quad\quad$ $\alpha \cup_h (\alpha \cap_h \beta) = \alpha$

P4 $\alpha \cap_h (\beta \cup_h \gamma) = (\alpha \cap_h \beta) \cup_h (\alpha \cap_h \gamma)$
$\alpha \cup_h (\beta \cap_h \gamma) = (\alpha \cup_h \beta) \cap_h (\alpha \cup_h \gamma)$

P5 $\alpha \cap_h \alpha = \alpha$ $\quad\quad$ $\alpha \cup_h \alpha = \alpha$

P6 $\alpha \cap_h (\beta \cup_h (\alpha \cap_h \gamma)) = (\alpha \cap_h \beta) \cup_h (\alpha \cap_h \gamma)$
$\alpha \cup_h (\beta \cap_h (\alpha \cup_h \gamma)) = (\alpha \cup_h \beta) \cap_h (\alpha \cup_h \gamma)$

P7 $\alpha \cap_h \mathbf{0} = \mathbf{0}$ $\quad$ $\alpha \cup_h \mathbf{0} = \alpha$ $\quad\quad$ $\alpha \cap_h \mathbf{1} = \alpha$ $\quad$ $\alpha \cup_h \mathbf{1} = \mathbf{1}$

P8 $\alpha = \neg_h \neg_h \alpha$

P9 $\neg_h (\alpha \cup_h \beta) = \neg_h \alpha \cap_h \neg_h \beta$ $\quad\quad$ $\neg_h (\alpha \cap_h \beta) = \neg_h \alpha \cup_h \neg_h \beta$

P10 $\alpha \cap_h \neg_h \alpha \leq \beta \cup_h \neg_h \beta$

P11 $\alpha \cap_h (\alpha \supset_h \beta) \leq \beta$

**Rule of Inference:**

R1 If $\alpha \cap_h \gamma \leq \beta$ then $\gamma \leq (\alpha \supset_h \beta)$.

P1-P10 then characterise $\cap_h$, $\cup_h$, and $\neg_h$. These postulates very nearly, but don't quite, characterise Boolean algebras. Instead of a postulate for a universal complement,

$$\alpha \cap_h \neg_h \alpha = \mathbf{0}, \qquad \alpha \cup_h \neg_h \alpha = \mathbf{1}$$

we obtain the weaker postulate P10. However we retain postulates governing universal bounds (P7) and involution (P8) as well as De Morgan's laws (P9). The weakened complement arises from the fact that the known extension and antiextension of a predicate typically do not together constitute the set of known individuals **I**. The subalgebra [ **H**; $\neg_h$, $\cap_h$, $\cup_h$] has been investigated under the names of *normal involution lattices* [Kalman 58] and *Kleene algebras* [Kleene 52].

Postulate P11 and rule R1 govern the ply operator. P11 effectively specifies in the algebra that *modus ponens* applies to the ply, while R1 says that the ply is maximal among potential operators satisfying P11. The postulate and rule are important because they guarantee that, when we come to consider the corresponding propositional logic, implication in the logic will correspond to standard material implication. On the other hand, it is readily shown that the ply is not reducible to other operators and, in particular, the equivalence $(\alpha \supset \beta) \equiv (\neg\alpha \vee \beta)$ of propositional logic (PC) will not hold in the derived logic. The reason that this equivalence is not obtained however is because the complement in HLA is weaker than in Boolean algebra and so, not surprisingly, negation is weaker in our system than in PC.

### 4.2. A Propositional Logic for HLA

For the algebra HLA we also derive the corresponding propositional logic, HLL. While the correspondence is interesting from a technical standpoint, the logic also serves to specify evidential relations in a perhaps more familiar fashion. The operations of hypothesised intersection, union, and complement are clearly analogous to the logical operations of conjunction, disjunction, and negation; the ply corresponds to the material conditional. This logic is specified as follows:

**Axiom Schemata**

A1 $\alpha \supset (\beta \supset \alpha)$

A2 $(\alpha \supset (\beta \supset \gamma)) \supset ((\alpha \supset \beta) \supset (\alpha \supset \gamma))$

A3 $\alpha \wedge \beta \supset \alpha$

A4 $\alpha \wedge \beta \supset \beta$

A5 $\alpha \supset (\beta \supset (\alpha \wedge \beta))$

A6 $\alpha \supset (\alpha \vee \beta)$

A7 $\beta \supset (\alpha \vee \beta)$

A8 $(\alpha \supset \gamma) \supset ((\beta \supset \gamma) \supset (\alpha \vee \beta \supset \gamma))$

A9 $\alpha \equiv \neg\neg\alpha$

A10 $(\alpha \supset \neg\alpha) \vee \neg(\alpha \supset \neg\alpha)$

**Rules of Inference**

MP From $\vdash \alpha$ and $\vdash \alpha \supset \beta$ infer $\vdash \beta$.

HN $\vdash \alpha \supset \beta$ iff $\vdash (\alpha \supset \neg\alpha) \vee \beta$ and $\vdash (\neg\beta \supset \beta) \vee \neg\alpha$.

A semantic account for the formulae of HLL follows easily from the algebra HLA:

**Definition:** $\models \alpha'$ in HLL iff $\alpha = \mathbf{1}$ in HLA.

The formula $\alpha'$ is obtained from $\alpha$ by replacing intersection with conjunction, etc. in the obvious way. The material conditional is linked to containment, via entailment, by the following.

**Proposition:** $\models \alpha' \supset \beta'$ in HLL iff $\alpha \leq \beta$ in HLA.

We obtain:

**Theorem:** HLL is sound and complete with respect to HLA.

The intended interpretation of derivability in the logic corresponds to an evidential support relation between formulae. That is $\Gamma \vdash_{HLL} \alpha$ can be interpreted as "if we have evidence for the elements of $\Gamma$, then we are guaranteed to have evidence for $\alpha$". That this interpretation is indeed reasonable can be seen from the correspondence $\vdash_{HLL} \alpha' \supset \beta'$ iff $\alpha \leq \beta$ in HLA. Thus $\vdash_{HLL} \alpha' \supset \beta'$ if and only if the instances supporting $\alpha$ also support $\beta$.

Not surprisingly, negation in HLL is weaker than in PC: we lose *reductio ad absurdum* as a method of proof; also we lose the law of the excluded middle. PC is obtained however if A10 and HN are replaced by $(\alpha \supset \beta) \supset ((\alpha \supset \neg\beta) \supset \neg\alpha)$. The logic resembles the system of first-degree entailment, $\mathbf{E}_{fde}$, of [Anderson and Belnap 75]. The principal difference is that axiom A1 and the theorem $\alpha \supset (\beta \supset \beta)$ of HLL is rejected by Anderson and Belnap, while their axiom $(\alpha \supset \beta) \supset (\neg\beta \supset \neg\alpha)$ is not a theorem of HLL. We also obtain:

**Theorem:**

$\vdash \alpha \supset \beta$ iff $\vdash \neg\beta \supset \neg\alpha$.

If $\vdash \neg\alpha \vee \beta$ then $\vdash \alpha \supset \beta$.

If $\vdash \beta$ then $\vdash \neg\beta \supset \alpha$.

If $\vdash \beta$ and $\vdash \alpha \supset \neg\beta$ then $\vdash \neg\alpha$.

If $\vdash \neg(\alpha \supset \beta)$ then $\vdash \neg\beta$.

Note however that none of the formulae obtained by replacing the meta-theoretic "if $\cdots$ then $\cdots$" in the above theorem, with the material conditional, is a theorem of HLL. Also, neither $\alpha \vee \neg\alpha$ nor $(\alpha \wedge \neg\alpha) \supset \beta$ nor $(\neg\alpha \vee \beta) \equiv (\beta \supset \alpha)$ are theorems of HLL.

Thus it appears that the introduction of a single (i.e. uniterated) negation operator can generally take place only between valid statements. This is in contrast with PC where the introduction of negation is a relation among formulae. Since converting between a disjunction and a conditional in PC involves the use of negation, we can only relate valid instances of disjunction (and hence conjunction) with valid instances of the conditional in HLL. This point is worth restating for emphasis. In HLL, conjunction, disjunction, and the conditional behave *exactly* the same as in PC. The difference between HLL and PC lies in their respective treatment of negation: the negation operator of HLL is weaker than that of PC. Thus although the equivalence $(\neg\alpha \vee \beta) \equiv (\alpha \supset \beta)$ is lost in HLL, this loss cannot be attributed to a differing interpretation of implication; rather the loss is an artifact of the weaker negation.

This last point proves to be the key in resolving the paradoxes of confirmation. However before considering the paradoxes in light of this system, we first further investigate the properties of the negation operator by considering an alternative, 3-valued, semantics for HLL.

### 4.3. A Three-Valued Semantics

This section continues the analysis by considering a three-valued semantics for HLL. In addition to the truth values "true" and "false", a third is introduced standing for, perhaps, "indeterminate" or "undetermined".[6] "True" then, from our reading in the logic of the last section, can be interpreted as "has supporting evidence". Truth tables are given for the logic, and it is shown that the set of theorems corresponds to the set of tautologies. This result is interesting then not only because it provides further insight into the logic, but also because it yields an easy decision procedure via the truth tables. The consistency proof is quite easy; the completeness proof follows the method presented in [Mendelson 64] for the completeness of PC, but extends it to deal with three truth values.

The expression for the hypothesised intersection $\alpha \cap_h \beta$ was given by $\langle \alpha_+ \cap \beta_+, \alpha_- \cup \beta_- \rangle$. Given our previous interpretation, this can be read as "something is evidence for $\alpha$ and $\beta$ iff it is evidence for both $\alpha$ and $\beta$; something is not evidence for $\alpha$ and $\beta$ iff it is not evidence for either $\alpha$ or $\beta$; otherwise the evidential relation cannot be determined". By similar reasoning with the definitions of the other connectives, we get the following truth tables:

| $\alpha$ | $\neg\alpha$ |
|---|---|
| T | F |
| ? | ? |
| F | T |

| | $\alpha \wedge \beta$ | | | $\alpha \vee \beta$ | | | $\alpha \supset \beta$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $\alpha/\beta$ | T | ? | F | T | ? | F | T | ? | F |
| T | T | ? | F | T | T | T | T | ? | F |
| ? | ? | ? | F | T | ? | ? | T | T | F |
| F | F | F | F | T | ? | F | T | T | T |

[6] "Epistemic value" is undoubtedly a more appropriate term than "truth value" in this context; however, for simplicity I will retain the more familiar "truth value".

These values are determined entirely by the underlying algebra, HLA, and thus by our original assumptions concerning evidential relations. Hence for example the fact that if $\alpha$ and $\beta$ have value **?** then $\alpha \supset \beta$ has value **T** is dictated by the assumption that the ply is maximal among possible operators satisfying modus ponens. The truth tables are truth-functional and, moreover, reduce to PC when only the truth values **T** and **F** are considered. A truth value assignment $V$ for formulae of HLL is easily defined, based on these truth tables.[7] Given such a definition then, as usual, $V$ *verifies* a formula $\alpha$ if $V(\alpha)=\mathbf{T}$. A formula is a *tautology* if $V(\alpha)=\mathbf{T}$ for all truth value assignments for $\alpha$. We obtain:

**Theorem:** $\alpha$ is a tautology iff $\alpha$ is a theorem of HLL.

Since we now have three truth values, a point of interest is to further examine the role of negation in HLL with respect to these values. Our approach is to consider three "negation operators", one for each truth value, where each operator is true just when the formula that it is applied to has a specified truth value, and is false otherwise. Thus, for falsity we would want the operator to be **T** when the formula is **F**, and **F** otherwise. If we call these operators $J_1$, $J_2$ and $J_3$, we require the following truth tables.[8]

| $\alpha$ | $J_1(\alpha)$ | $J_2(\alpha)$ | $J_3(\alpha)$ |
|---|---|---|---|
| **T** | **T** | **F** | **F** |
| **?** | **F** | **T** | **F** |
| **F** | **F** | **F** | **T** |

Interestingly, these operators are definable in terms of negation and the conditional:

**Definition:**

$J_1(\alpha)$ is $\neg(\alpha \supset \neg\alpha)$.

$J_2(\alpha)$ is $(\alpha \supset \neg\alpha) \wedge (\neg\alpha \supset \alpha)$ or $\neg J_1(\alpha) \wedge \neg J_3(\alpha)$.

$J_3(\alpha)$ is $\neg(\neg\alpha \supset \alpha)$ or $J_1(\neg\alpha)$.

**Theorem:**

$\vdash_{HLL} (\beta \supset J_3(\alpha)) \supset ((\beta \supset J_2(\alpha)) \supset ((\beta \supset J_1(\alpha)) \supset \neg\beta))$.

$\vdash_{HLL} J_1(\alpha) \vee J_2(\alpha) \vee J_3(\alpha)$.

The first formula of the theorem provides an analogue to the usual axiom of PC for introducing negation, $(\beta \supset \neg\alpha) \supset ((\beta \supset \alpha) \supset \neg\beta)$. That is, one way to prove $\neg\beta$ is to show that assuming $\beta$ leads to a contradiction. This didn't work for us with $\neg$ since we don't have $\vdash_{HLL} \alpha \vee \neg\alpha$. However the "extended" negation operators fill the gap: a valuation must be one of true, false, or indeterminate. This is given explicitly in the second formula of the theorem, and provides a law of the excluded middle for HLL. It also proves to be the case that either of the formulae of the theorem could be used in place of A10 in the proof theory of HLL.

Appositives – words that follow a noun or pronoun and identify it – are usually set off by commas if they are nonrestrictive. Such words add parenthetical information, for example: Her cat, Mount Diablo Base Line as Seen from Highway 395 Just North of Bridgeport, knew how to give an admirable back rub with his well-tempered and discreet claws.

This treatment of HLL as a three-valued logic can be equally well viewed as specifying a correspondence between HLL and the (specific) system of HLA, $[\{\mathbf{T},?,\mathbf{F}\}; \neg_h, \cap_h, \cup_h, \supset_h]$. In this case **T** corresponds to **1**, **F** corresponds to **0** and the truth tables give the results of applying an operation to any two elements of the set $\{\mathbf{T},?,\mathbf{F}\}$.

## 5. A Solution to the Paradoxes of Confirmation

The rule of inference for negation, NH, provides the resolution for the paradoxes of confirmation. Recall that the paradoxes arose from the assumption that whatever supports a universal generalisation also supports all statements that are logically equivalent to the generalisation. Thus, for example, non-black non-ravens apparently support the hypothesis that all ravens are black. The paradox is resolved here not by rejecting the assumption that evidence supports all logical consequences of a generalisation; rather it is resolved by asserting that the notion of classical consequence is too strong for the problem at hand. What we have effectively done to this point then is develop a logic for specifying what hypotheses are supported by the same evidence.

Consider then the hypothesis $Raven \supset Black$. In PC, we have:

$$\vdash (Raven \supset Black) \equiv (\neg Black \supset \neg Raven).$$

Hence, if we were using PC to determine which hypotheses are supported by what evidence, then a black raven would also support the statement that non-black things are non-ravens. However the above is certainly not a theorem of HLL. Rather what we have is the meta-theorem:

$$\vdash_{HLL} Raven \supset Black \quad \text{iff} \quad \vdash_{HLL} \neg Black \supset \neg Raven,$$

or in the obvious translation into a first-order system,

[7] Thus for example, for conjunction, $V(\alpha \wedge \beta)=\mathbf{T}$ iff $V(\alpha)=\mathbf{T}$ and $V(\beta)=\mathbf{T}$; $V(\alpha \wedge \beta)=\mathbf{F}$ iff $V(\alpha)=\mathbf{F}$ or $V(\beta)=\mathbf{F}$; otherwise $V(\alpha \wedge \beta)=?$.

[8] This notation is taken from [Rosser and Turquette 52], as is the notion of the extended negation operators.

$\vdash(x)(Raven(x) \supset Black(x))$ iff $\vdash(x)(\neg Black(x) \supset \neg Raven(x))$.

Thus a black raven provides no support for non-black things being non-raven. However, if it is the case that all ravens are indeed black, then it also is the case that all non-black things are non-ravens. In a similar fashion, since we no longer have the equivalence $(Raven \supset Black) \equiv (\neg Raven \vee Black)$, a confirming instance for one side of the equivalence does not constitute evidence for the other side.

This logic also is the strongest logic in which the paradoxes do not arise, in the following sense. Recall that in the algebra HLA terms were identified with extension/antiextension pairs; these pairs specify *all* that can be known about a particular term. Thus, for $\alpha \cap_h \beta$, the only individuals that are known to be in the intersection are those in $\alpha_+ \cap \beta_+$, and the only individuals that are known to not be in the intersection are those in $\alpha_- \cup \beta_-$. Furthermore, no other individuals may justifiably be claimed to be in the extension or antiextension. In the case of the ply, for example, this leads to some not-entirely intuitive properties – for example the value of $\alpha \supset \beta$ where both $\alpha$ and $\beta$ are **?** is **T**. However, one clearly could consider a yet weaker logic (were the preceding formula would have value **?** for example) with arguably more plausible properties, and it should be clear that the problems of confirmation would not arise in that framework. Rather, the logic HLL specifies, according to our assumptions, the strongest such logic in which the problems do not arise.

It is interesting to note that in the surveys of proposed solutions to this problem ([Black 65], [Scheffler 81]), little consideration was given to altering the underlying logic. The unstated assumption (presumably) was that altering the logic was too large a sacrifice if the logic was to also be used for reasoning about the conjectures. Here though the approach is to explicitly distinguish the formal systems used for determining relations of evidential support among hypotheses, from the systems for deductively reasoning with hypotheses. In this case, in the system for evidential support, HLL, the paradoxes of confirmation don't arise. This weaker logic then is appropriate for deciding what confirms what. For deductively reasoning with hypotheses, first-order logic is presumably appropriate.

## 6. Conclusions

This paper has proposed and developed a solution to the "paradoxes of confirmation". The approach is to reject the claim that evidence supporting a hypothesis also supports all consequences of the hypothesis in classical first-order logic. Instead, beginning from first principles we examine a particular, arguably general, notion of "evidence" and from this develop formal systems that characterise a relation of evidential support. An algebra HLA is derived based on intuitions concerning elementary notions of support. From this system a logic HLL is derived; equivalent sentences in this logic correspond to statements that are (or would be) confirmed by the same evidence. It proves to be the case that in HLL conjunction, disjunction, and the conditional have the same properties as in classical propositional logic. Negation however is weaker; in particular we do not have the standard law of the excluded middle. In order to further characterise and examine this notion of negation, we also consider a three-valued semantics for HLL. Analogues to the properties of classical negation (for example, the excluded middle) are also presented.

It was shown that within the resultant framework the paradoxes do not arise. Informally, the problem does not occur because, with the weaker negation, instances known to satisfy or to not satisfy a particular predicate do not constitute the set of all known individuals. However, we still retain a useful notion of negation. For example, while an instance that supports the hypothesis $\alpha(x) \supset \beta(x)$ does not support $\neg\beta(x) \supset \neg\alpha(x)$, it is the case that any instance supporting $\alpha(x)$ also supports $\neg\neg\alpha(x)$ and that any instance supporting $\alpha(x) \wedge \beta(x)$ also supports $\neg(\neg\alpha(x) \vee \neg\beta(x))$.

The notion of evidence for a hypothesis was left undefined except for two, arguably fundamental, restrictions. First we require that a supported hypothesis have at least a single known confirming instance. Second, we require that evidence for a hypothesis be a function of the known satisfying (and known non-satisfying) instances in the terms of the hypothesis. These restrictions appear to be quite general, and presumably any AI system that learns from examples satisfies these conditions.


### Acknowledgement

This paper is taken in part from my doctoral dissertation in the Department of Computer Science at the University of Toronto. Robert Hadley, Sharon Hamilton, and Howard Hamilton provided helpful comments to an earlier draft; Pierre Massicotte helped with the translation of the asbtract. I also wish to thank the two anonymous referees for their comments. Financial assistance from the Province of Ontario and the Department of Computer Science, University of Toronto, is gratefully acknowledged, as well as support from the Natural Sciences and Engineering Research Council of Canada grant A0884.

## References


[1] A.R. Anderson and N.D. Belnap Jr., *Entailment: The Logic of Relevance and Necessity, Vol. I*, Princeton University Press, 1975

[2] M. Black, "Notes on the 'Paradoxes of Confirmation'", in *Aspects of Inductive Logic*, J. Hintikka and P. Suppes eds., North-Holland, 1966

[3] J.G. Carbonell, R.S. Michalski, and T.M. Mitchell, "An Overview of Machine Learning", in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), pp. 3-23, 1983.

[4] B.F. Chellas, "Basic Conditional Logic", *Journal of Philosophical Logic 4*, 1975, pp 133-153.

[5] H.B. Curry, *Foundations of Mathematical Logic*, McGraw-Hill, 1963

[6] J.P. Delgrande, "A First-Order Logic for Prototypical Properties", *Artificial Intelligence*, 33, 1, 1987a.

[7] J.P. Delgrande, "A Formal Approach to Learning from Examples", *10th International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987b.

[8] J.P. Delgrande, "A Foundational Approach to Autonomous Knowledge Acquisition", *Computational Intelligence*, 4, 3, 1987c.

[9] J.P. Delgrande, "An Approach to Default Reasoning Based on First-Order Conditional Logic: Revised Report", *Artificial Intelligence* (to appear), 1988.

[10] N. Goodman, *Fact, Fiction and Forecast*, 3rd ed., Hackett Publishing Co., 1979

[11] C.G. Hempel, "Studies in the Logic of Confirmation", *Mind*, Vol. 54, pp. 1-26, 91-121, 1945.

[12] J.A. Kalman, "Lattices with Involution", *Transactions of the American Mathematical Society*, Vol. 87, 1958, pp 485-491

[13] S.C. Kleene, *Introduction to Metamathematics*, North Holland Pub. Co., 1952

[14] D. Lewis, *Counterfactuals*, Harvard University Press, 1973.

[15] E. Mendelson, *Introduction to Mathematical Logic*, D. Van Nostrand Co., 1964

[16] H. Putnam, "'Degree of Confirmation' and Inductive Logic", in *Mathematics, Matter and Method: Philosophical Papers Volume I*, 2nd ed., Cambridge University Press, 1979, pp 270-292

[17] J.B. Rosser and A.R.Turquette, *Many-Valued Logics*, North-Holland Pub. Co., 1952

[18] I. Scheffler, *The Anatomy of Inquiry: Philosophical Studies in the Theory of Science*, Hackett Publishing Co., 1981

# Concepts, Analogies, and Creativity

Douglas R. Hofstadter and Melanie Mitchell
Department of Psychology and Department of Computer Science
University of Michigan
Ann Arbor, Michigan 48104

**Abstract**

Analogy-making requires the ability to perceive situations very flexibly, to use concepts fluidly rather than rigidly, to decide which aspects of a situation are relevant, to determine the "best" level of abstraction at which to perceive a given situation, and to flexibly translate ideas in one situation into a different situation. These abilities are central to every facet of human intelligence, from perception and learning to recognition of concrete and abstract objects (faces, letters, artistic and musical styles), and even to creativity. We believe that the mechanisms for analogy-making in humans are centered on the overlapping, associative, and flexible structure of human concepts, and the goal of our research is to understand the nature of this structure and how creative analogy-making arises from it. In this paper we describe a theory of how this conceptual structure interacts with perception in the process of analogy-making, and describe a computer model of our theory that is able to create analogies in a "microworld". We believe that this microworld captures much of the essence of analogy-making in general and that mechanisms successful in this small world will form the core of models of creativity in more complex domains.

Keywords: Analogy, Cognitive Modeling, Concepts, Creativity

## 1. Introduction

This paper describes a theory of human concepts and analogy-making, and a computer model of this theory that is able to create analogies in a small domain. The goals of this research are: (1) to understand the structure of human concepts: their overlapping and associative nature, and their flexibility as a function of context -- in short, their "fluidity"; (2) to understand how fluid concepts are used in the creation of analogies; (3) to understand the role played by such analogies in creative thought.

## 2. The Microworld of Our Model

For our research we have designed a "microworld" small enough to be manageable, but large and rich enough to contain what we believe to be the recurrent issues in analogy-making (Hofstadter, 1984a). The microworld consists of the 26 letters of the alphabet and other associated concepts; in it we construct analogy problems involving strings of letters. Despite their apparent simplicity, many of these problems involve abstract notions such as *grouping*, *symmetry*, and *sequence*. Seven sample problems are given below. (Notational note: In what follows, lowercase boldface letters designate *instances* of letter categories, and uppercase boldface letters designate the categories themselves. For example, **a** is an instance of the category **A**.)

1. *If* **abc** ---> **abd**, *then* **pqrs** ---> ? (That is, if the string **abc** changes to **abd**, then what is the analogous change to **pqrs**?)

A literal answer is **pqrd**, using the rule "Replace the rightmost letter by a **D**". A more abstract way of perceiving the change is "Replace the rightmost letter by its successor", yielding **pqrt**. Other possible answers include **pqss** ("Replace the third letter by its successor"), **pqds** ("Replace the third letter by a **D**"), **pqrs** ("Replace all **C**'s by **D**'s"), and **abd** ("Replace any string by **abd**"). Most people prefer **pqrt**.

2. *If* **abc** ---> **abd**, *then* **srqp** ---> ?

Here there are two answers that view the situation abstractly: **trqp** and **srqo**. The original rule is still "Replace the rightmost letter by its successor", but this rule must be "translated" to apply to **srqp**. The answer **trqp** results from the translation "Replace the leftmost letter by its successor", involving a "slippage" from *rightmost* to *leftmost* (i.e., the *leftmost* letter in **srqp** plays the role of the *rightmost* letter in **abc**) based on seeing **abc** as a left-to-right string and **srqp** as a right-to-left string (where each string increases alphabetically). The answer **srqo** results from the translation "Replace the rightmost letter by its predecessor", a result of seeing **abc** as increasing and **srqp** as decreasing (both viewed as moving rightwards), yielding a slippage from *successor* to *predecessor*.

3. *If* **abc** ---> **abd**, *then* **aaabbbccc** ---> ?

A rigid answer would be **aaabbbccd**. A more insightful perceiver would see **aaabbbccc** as consisting of three *groups* (**aaa-bbb-ccc**). The answer **aaabbbddd** involves a slippage from the concept *letter* to the concept *group* ("Replace the rightmost group by its 'successor'").

4. *If* **abc** ---> **abd**, *then* **mrrjjj** ---> ?

One answer is **mrrkkk**, but this doesn't take into account the abstract similarity between **abc** and **mrrjjj**: **abc** increases alphabetically while **mrrjjj** consists of groups that increase numerically. If this similarity is perceived, then the answer is **mrrjjjj**, fluidly extending the concept *successor*. People occasionally give the answer **mrrkkkk**, replacing the rightmost group by both its numerical and alphabetic successor. This answer is strange in that it confounds aspects of the two situations.

Once the abstract similarity between **abc** and **mrrjjj** has been perceived, and the slippage from *alphabetic successor* to *numeric successor* has been made, then to also apply the notion of alphabetic successor to the numeric-successor situation shows a strange combination of flexibility and rigidity. It is as if a translator decided to tell the story of War and Peace in the context of the American Civil War, but gave the (now American) characters the names "Natasha" and "Alexey". An even stranger translation would leave the names in the original Cyrillic letters, which might correspond to the answer **mrrdddd**.

5. *If* **abc** ---> **abd**, *then* **aababc** ---> ?

Another creative extension of the concept "successor": if **aababc** is parsed as **a-ab-abc**, then the "successor" of the rightmost "letter" (actually the *group* **abc**) is **abcd**, yielding the answer **a-ab-abcd**, which seems elegant compared to **aababd**, in which no grouping was perceived, and the **C** was simply replaced by a **D**.

6. *If* **abc** ---> **abd**, *then* **ace** ---> ?

A fairly literal answer is **acf**, which doesn't take into account the "double successor" structure of **ace** (**C** is the double successor of **A**, etc.). If **ace** is seen as similar to **abc** because **abc** is a "successor group" and **ace** is a "double-successor group", then the best answer is **acg**.

7. *If* **abc** ---> **abd**, *then* **xyz** ---> ?

We exclude the "circular-alphabet" answer **xya** in order to force the analogy-solver to deal with the "snag" (that **Z** has no successor) and to look at the problem more deeply. A very abstract but compelling view (once seen) is that the **A** in **abc** corresponds to the **Z** in **xyz**, since **A** is the *first* letter of the alphabet, and **Z** is the *last* letter. This renders **abc** and **xyz** abstract "mirror images" of each other, making possible a strong answer: **wyz**.

As is evident, many analogy problems in this microworld require creativity; they require the solver to view situations flexibly, to perceive abstract kinds of similarity, to decide which aspects of a situation are relevant, to determine the "best" level of abstraction at which to perceive a given situation (e.g., "Replace the rightmost letter by its successor" versus "Replace the rightmost letter by a **D**"), and to flexibly *translate* ideas in one situation into a different situation. The letter strings can be thought of as representing abstract idealized situations; in this sense, the concepts and situations of our microworld become metaphors for concepts and situations in other domains. The abilities listed above are by no means unique to solving letter-string analogy problems, but are central to the creation of an analogy between *any* two situations. In fact, these abilities are central to every facet of human intelligence, from perception and learning to recognition of concrete and abstract objects (faces, letters, artistic and musical styles), and even to creativity. We believe our microworld captures much of the essence of analogy-making in general and we are creating a computer program that can make fluid and creative analogies in this small world. Our belief is that mechanisms successful in this small world will form the core of models of creativity in more complex domains.

## 3. Competing Pressures and Deep Analogies; Related Work on Analogy

A fundamental tenet of our theory is that analogies arise from the interaction and competition of many pressures (Hofstadter, 1984b). Pressures can be aspects of a given situation that push the perceiver to view it in a certain way, or they can be *a priori* biases of the perceiver. For example, in problem 6, there is a perceptual pressure to map the **C** in **abc** onto the **C** in **ace**, which competes with a pressure to see the two rightmost letters as playing the same (more abstract) role. An essential question is how "syntactic" pressures (emphasizing more superficial features) compete with "semantic" pressures (emphasizing more abstract features). In general, in good analogies, the more semantic pressures win out, but this is not always the case (Hofstadter, 1985a).

Much of the work on analogy by other researchers has focused on what pressures analogy-making involves. Gentner's work on structure-mapping and systematicity is perhaps the best-known (Gentner, 1983). We agree that systematicity plays an important role in many analogies, but we see it as one pressure among many, not always producing the strongest answer. As we show below, in our system, pressures toward systematicity are emergent rather than explicit driving principles, and must compete with other pressures. Holyoak and Thagard (1987) have proposed five constraints on analogical mapping that are very similar to pressures in our system, and they use a relaxation algorithm on a connectionist network to allow these constraints to compete and cooperate. Other researchers (Burstein, 1986; Carbonell, 1983; Darden, 1983; Evans, 1968; Greiner, 1985; Kedar-Cabelli, 1985; Winston, 1980) have proposed pressures similar to systematicity and to the constraints of Holyoak and Thagard, and have built models for deciding how to use these pressures in different situations. More detailed comparisons of our work with other research is given by Hofstadter (1984b) and by Hofstadter, Mitchell, and French (1987).

Some researchers propose that mapping one situation onto another can be done only if one knows the purpose of the analogy (Kedar-Cabelli, 1985), and that the goodness of the mapping is determined largely by the "usefulness" of the analogy in solving problems or making predictions (Greiner, 1985). In our view, many analogies do have a purpose (e.g., to help understand a given problem), and such a purpose creates pressures affecting the analogy. But we disagree on the centrality of purpose or usefulness in analogy. Many analogy problems in our microworld (and in real life) have compelling answers despite the absence of any purpose they might serve. This is because humans have a strong sense of context-independent similarity; that is, certain mappings seem compelling in and of themselves. (Consider the geometric analogies of Evans, 1968, and the pattern-recognition problems of Bongard, 1970, which helped inspire our microworld.) We are studying how differences in pressures affect the preference for one mapping over another. These pressures can of course include ones related to a specific purpose, but this is by no means necessary. This difference of opinion brings up another difference between our focus and that of many other researchers. Analogy is often characterized as something used in problem-solving -- i.e., as a "tool" to be used by a reasoning procedure. We believe the opposite: in our view the analogical facility pervades cognition at every level, and high-level cognitive functions (reasoning, problem-solving) emerge from analogy-making capacities, rather than the reverse.

## 4. Architecture of the Model

"Copycat" is the name of the computer program that serves as a model of our theory. It has been described previously by Hofstadter (1984b, 1985a) and by Hofstadter, Mitchell, and French (1987). (Meredith, 1986, has used a very similar architecture to model pattern perception in a different but closely related domain.) The program produces answers to letter-string analogy problems like the ones given above. The philosophy behind the model is that fluid, human-like thought should be an emergent property of a system composed of many independent processes running in parallel, influencing each other by creating and destroying temporary structures and by raising and lowering the activation levels of various permanent concepts stored in an associative network. We rely on statistics to bring coherent and focused behavior out of a "seething broth" of small parallel actions. A guiding metaphor for this model is the way in which a living cell's coherent global activity is created by the joint actions of millions of independently-acting enzymes. As in a living cell, Copycat has no top-level executive; rather its processing is carried out in parallel by cooperating and competing "chains" of "enzymes" (small pieces of code) that build and destroy organizing structures while interacting probabilistically with each other and with a network of permanent concepts (a "Slipnet").

### 4.1 The Slipnet

The model is centered on a structure we call the "Slipnet", which embodies in an approximate way our notion of what concepts are, how they overlap, and how, under pressures, one concept can "slip" into another related one, producing a shift of viewpoint and perhaps a new insight.

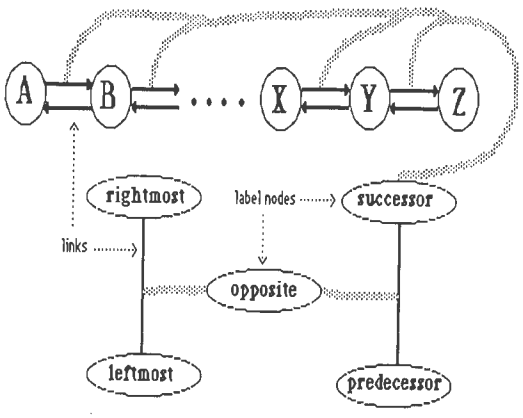



*Figure 1: A small part of Copycat's Slipnet*

A small part of Copycat's Slipnet is shown in Figure 1 above. A node in the Slipnet represents the "core" of a concept, and a link simultaneously represents a resemblance or relationship between two nodes and a potential substitution of one node for the other in descriptions containing that node (e.g., given certain situations, the description "rightmost letter" could become "leftmost letter", as it did in problem 2 above). We call such a substitution a slippage from one node to another. The shorter the link, the stronger the resemblance, and the easier it is for a slippage to be made. Slippages are not restricted to nodes connected by a single link; a slippage can be made between any two sufficiently close nodes. Thus the Slipnet represents all potential slippages.

Each node has a fixed degree of semanticity: a number representing the node's degree of conceptual depth. We use the word "semantic" to describe relatively general or deep concepts in order to contrast with the notion of "syntactic", which implies easily seen surface qualities. Letters have a low degree of semanticity; concepts like "predecessor" and "rightmost" are more semantic; then come concepts like "repetitive group"; finally, domain-independent concepts like "opposite" are the most semantic of all. Each node also has a time-varying activation -- a number representing its perceived relevance to the problem at hand. (Similar notions of activation have also been used by Anderson, 1983, Collins and Loftus, 1975, Norman and Rumelhart, 1982, Quillian, 1968, Reitman, 1965, and in connectionist networks such as those of Feldman and Ballard, 1982, and McClelland and Rumelhart, 1986). Nodes become activated when the program perceives instances of the concept involved, and stay activated as long as such instances are perceived. The activation of a node pushes the system to look for more instances of the given concept. Nodes spread activation to their neighbors, and also decay. Low-semanticity nodes lose activation quickly through decay, hence never get very activated, whereas very semantic nodes, which decay slowly, can reach high activations. This ensures that general and abstract concepts have deeper influences on the course of the analogy, producing an effect that is similar to Gentner's "abstraction" and "systematicity" principles (Gentner, 1983).

Links have two attributes: length and label. The length of a link corresponds to the intuitive notion of conceptual distance. The label of a link roughly classifies the resemblance or relationship the link encodes. Each label is itself represented by a node, and the length of a link changes dynamically in a manner determined by the activation of its label node. For example, the nodes *leftmost* and *rightmost* are connected by a link with label *opposite* (see Figure 1). A high activation of *opposite* will cause the link between *leftmost* and *rightmost* to shrink. The distance between any two nodes is a function of the lengths of the links on all the paths between the two nodes. This distance function determines the perceived resemblance and the likelihood of slippage between any two nodes.

The blurriness and context-sensitivity of human concepts has been investigated by many cognitive scientists, including Hofstadter (1985a), Holland et al. (1986), Kahneman and Miller (1986), Lakoff and Johnson (1980), Rosch et al. (1976), and Smith and Medin (1981). In our model, a concept is a region in the Slipnet, centered on a particular node (its core), having blurry rather than sharp boundaries: any other node is included in the concept with probability inversely related to its distance from the core node. For example, the concept **B** is almost always part of the concept **A**, but **C** is less likely to be included, and **Q** is almost never included. The boundaries of a concept are also dynamic: since the lengths of links in the Slipnet are variable, the distance from the core node to nearby nodes changes as different aspects of the problem are perceived. Thus concepts are not explicit structures that can be manipulated by the program, but rather they are emergent aspects of the Slipnet arising from many low-level interactions between dynamic link-lengths and environmental inputs. Their flexibility and context sensitivity are due to this emergent nature. This notion of emergent concepts (Hofstadter, 1984b) gives Copycat's Slipnet a flavor similar to many connectionist

models (McClelland and Rumelhart, 1986), and to classifier systems (Holland et al., 1986).

### 4.2 Perceptual Structures

In this section we will use the following problem as an example: If **abc** ---> **abd**, then **ssrrqqpp** ---> ? This is a combination of problems 2 and 3 above. There are two strong answers: **ttrrqqpp** and **ssrrqqoo**.

Copycat's input is a "raw" analogy problem: the program is given only the three strings of letters (**abc**, **abd**, and **ssrrqqpp**), and is told only the category membership of each of the letters (e.g., **a** is an instance of **A**), and the left-to-right order of the letters in each string. In order to formulate a solution, the program must "perceive" at a high level what is going on in the problem. To accomplish this, the program builds (on top of the letter strings) various kinds of structures that represent its high-level perception of the problem. (This is similar to the way the Hearsay II speech-recognizing program built perceptual structures on top of raw representations of sounds; see Erman et al., 1980.) These structures represent concepts of various degrees of generality being brought to bear on the problem, and accordingly, each of these structures is built out of copied parts of the Slipnet. The flexibility of the program rests on the fact that concepts from the Slipnet can be "borrowed" for use in perceiving situations, and that those concepts are not rigid but rather fluid, adjusting themselves to fit the situations at hand. An essential part of our model is this interaction of top-down and bottom-up processes: as the program's perception of a given problem adjusts itself to the shapes of concepts in the Slipnet, the shapes of those concepts themselves are determined by what the program perceives.

The program builds five kinds of perceptual structures: descriptions, relations, groups, correspondences, and rules. Descriptions borrow nodes from the Slipnet in order to describe objects in each string. For example, the first **S** in **ssrrqqpp** might have the following descriptions: "leftmost letter", "leftmost letter in leftmost repetitive group", "an instance of **S**", and so on. The group **ss** might have the descriptions "leftmost repetitive group", "repetitive group of **S**'s with length 2", and so on. Each word in the description is actually a node in the Slipnet (except for words like "in" and "of", which we insert for readability). Relations are descriptions of objects relative to other objects in the same string. Relations are built on top of descriptions, and also consist of borrowed parts of the Slipnet. For example, Figure 2 shows a relation between the **A** and the **B** in **abc**.

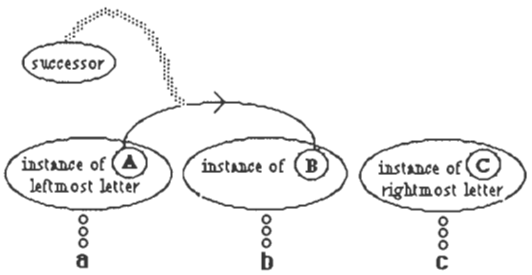



*Figure 2: A successor relation*

When the program notices relations, it begins looking for related groups. At present Copycat has concepts for three types of groups: successor groups (e.g., **abc** viewed from left to right, or **srqp** viewed from right to left), predecessor groups (e.g., **abc** viewed from right to left, or **srqp** viewed from left to right), and repetitive groups (e.g., **aaaa** and **abababab**, both being repetitive groups of length four). There are other types of groups the program should eventually be able to recognize, including symmetry groups (e.g., **elqle** and **abcdcba**) and generic tuples (e.g., **mblmblmbl** is a repetitive group containing three copies of the tuple **mbl**). Like letters, groups have descriptions and relations to other objects. For example, the group **bbb** can be represented by a parameter-letter ***B*** and thus the group **bbb** can be conceived of as the "successor" of the group **aaa.** Decomposing a string into groups can require considerable subtlety, as in perceiving that the string **aababc** contains three successor groups (**a**, **ab**, and **abc**) that themselves form a successor group at the parameter-letter level: ***ABC***).

The heart of what Copycat does in creating an analogy is to construct correspondences between objects in different strings. A correspondence is a mapping between two objects that are found to resemble each other in ways relevant to the situation at hand. Each correspondence is justified by a set of slippages (or substitution rules) between two concept nodes; those slippages give reasons for the correspondence. For example, the slippage *rightmost --> leftmost* says that the role of *rightmost* in the original string is being played by *leftmost* in the target string. A slippage consists of the names of two nodes, one from the first object's descriptions and the other from the second object's descriptions, which are (at the time the slippage is created) sufficiently close in the Slipnet. (Whether the two nodes are "sufficiently close" to each other is decided probabilistically). Figure 3 below shows two diagonal correspondences for the sample problem, along with the slippages that underlie them.

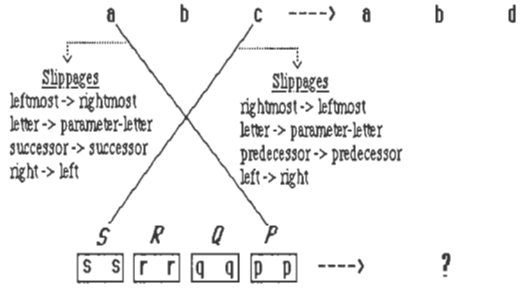



*Figure 3: A set of correspondences*

(For the sake of clarity, Figure 3 shows only the correspondences between **abc** and **ssrrqqpp**. The program would also build "horizontal" correspondences between **abc** and **abd**, most likely mapping the **A** to the **A**, the **B** to the **B**, and the **C** to the **D**.)

In Figure 3 above, the uppercase italic letters above the repetitive groups in **ssrrqqpp** are parameter-letters representing the groups. The program creates these parameter-letters when it perceives the repetitive groups. The correspondence between the **A** and the parameter-letter ***P*** is supported by the slippages *leftmost --> rightmost*; *letter --> parameter-letter*; *successor --> successor*; and *right --> left* (the last two slippages come from the fact that the **A** has a successor relation to its right neighbor, and the parameter-letter ***P*** has a successor relation to its left neighbor). The

correspondence between the **C** and the parameter-letter ***S*** is supported by the slippages *rightmost --> leftmost*; *letter --> parameter-letter*; *predecessor --> predecessor*; and *left --> right*. The use of the word "slippage" to describe a pair like *successor --> successor* may seem strange, since nothing was slipped! But if slippages are seen as reasons for correspondences between objects, then such an identity slippage can be a very strong reason for making a mapping between two objects.

Notice that there is no correspondence between the **B** in **abc** and anything in **ssrrqqpp**. This reflects the fact that in any analogy, many aspects of the source situation have no counterparts in the target situation. For example, when making an analogy between the Iran-Contra affair and Watergate, you probably wouldn't find a Watergate counterpart of the cake that was taken along on the secret mission to Iran (although one might exist). Mappings tend to be made between pairs of *salient* objects (such as Presidents Reagan and Nixon) and between pairs of very *similar* objects (such as the daytime TV broadcasts of the U.S. Senate hearings in both situations). Such biases for salience and similarity guide Copycat's construction of correspondences.

Each correspondence has a strength associated with it, representing in principle the goodness of the mapping it encodes, and in practice how hard the correspondence is for the program to destroy. The idea of assigning strengths to correspondences reflects the philosophy that good analogies are created by perceiving deep similarities between situations. This requires a strength function that values (1) slippages involving nodes with a high degree of semanticity (e.g., *letter --> parameter-letter*), suggesting that abstract aspects of the situations are being taken into account; (2) slippages with small distances in the Slipnet (e.g., the identity-slippage ***A --> A***), suggesting that very similar aspects about the situations are being taken into account; (3) correspondences supported by many slippages, suggesting that many aspects of the situations are being taken into account; and (4) correspondences that are compatible with already-existing correspondences. Roughly speaking, two correspondences are compatible if the supporting slippages of one are conceptually parallel to the supporting slippages of the other, expressing the same "world view"; for instance, the slippages *leftmost --> leftmost* and *rightmost --> rightmost* are parallel, whereas *leftmost --> leftmost* and *left --> right* are not. (The notion of valuing conceptually parallel slippages, along with the bias towards more semantic descriptions of objects and relationships, tends to give rise to mappings that satisfy Gentner's systematicity principle, but not deterministically.) A different set of mutually compatible correspondences (i.e., different from the set given in figure 3) would map the **A** in **abc** onto the parameter-letter ***S*** and the **C** onto the paramter-letter ***P***. In a run of the program, these two sets of correspondences would probably compete with each other; the choice of a winner would emerge from the outcome of many probabilistic choices based on the strengths of the correspondences (see figure 4).

In each of Copycat's analogy problems there is some kind of transformation carrying the first string (here, **abc**) into the second string (here, **abd**), and the analogy solver has to transform the third string (here, **ssrrqqpp**) in "the same way". To do this, Copycat builds a rule -- a structure summarizing the program's perception of the first transformation -- and uses the slippages relating the first and third strings to *translate* this rule for use on the third string. Some possible rules for **abc --> abd** are: "Replace all **C**'s by **D**'s", "Replace the rightmost letter by a **D**", "Replace the third letter by a **D**", and "Replace the rightmost letter by its successor". At present, the program uses the simplifying assumption that all rules are of the form "Replace _____ by _____", where only one object (i.e., letter or group) has been replaced. The blanks are to be filled in with descriptions of objects. This rule template can be used for a large number of problems, but certainly not all (e.g., "If **eqe** --> **qeq**, then **aaabaaa** --> ?", where the rule is to "turn the string 'inside-out'"), so this capability will have to be expanded.

The rule "Replace the rightmost letter by its successor" is a good description of the transformation between **abc** and **abd**, but not general enough to create good analogies in sufficiently distant situations. For example, if applied directly to **ssrrqqpp**, this rule would give **ssrrqqpq**, which virtually all people consider to be too literal and rigid. To apply to the new situation, the rule has to be *translated*. The translation is implicitly given in the slippages underlying the correspondences between **abc** and **ssrrqqpp**. For example, the set of correspondences shown in figure 3 includes the following slippages: *letter --> parameter-letter*, *rightmost --> leftmost*, and *successor --> successor*. The program simply applies these translation instructions to the original rule. This produces "Replace the leftmost parameter-letter by its successor", yielding answer **ttrrqqpp**.

Some researchers have proposed that people make analogies not by translation, but by creating a sufficiently abstract "schema" that applies to both the initial and target situations (Genesereth, 1980; Gick and Holyoak, 1983; Greiner, 1985). We agree that in some cases one can create an schema, but in general, no rule can anticipate how all other situations will differ from the original one. For example, all of the sample problems given at the beginning of this paper are in an abstract sense the "same" problem (in each, the idea is to replace an "extreme" element by its "successor"), but it would be very hard to create one single abstract schema that could be used to find the answer in each case, and impossible to create a schema that could be used on these and on the infinitely many other problems with this same theme. We believe that our scheme of translation via slippages borrowed from a concept network (i.e., a Slipnet) is a better model of the mechanisms behind analogy-making.

### 4.3 How Copycat Works

The actual building (and destroying) of perceptual structures is carried out by codelets -- small pieces of code that play the role of enzymes mentioned above. As was said before, Copycat has no top-level executive; all processing is carried out by codelets, many of which are independent and run in parallel. A certain set of codelets is present at the onset of processing. New codelets are sometimes created by old codelets to continue working on a task in progress, and these codelets may in turn create other codelets, and so on. New codelets can also be created by certain highly activated nodes in the Slipnet. When a codelet is created, it is assigned an urgency: a number representing how important it seems to its creator at the time of its creation, and it is placed in a pool of waiting codelets. At each time-step of the program, a certain number of codelets are selected from the pool probabilistically according to urgency, and these codelets run in parallel.

The operation of Copycat can be roughly divided into three stages:
1) Constructing one or more descriptions for each object (letter or group) in each string, perceiving relations between objects within a string, and noticing groups within a string (such as the four groups in **ssrrqqpp**) -- in short, building a high-level perception of each string.

2) Finding correspondences between objects in the initial string (**abc**) and the modified string (**abd**), and between objects in the initial string and the target string (**ssrrqqpp**).

3) Constructing a rule to describe the transformation between the initial string (**abc**) and the modified string (**abd**), and using the mapping between the initial and target strings to fluidly adapt (i.e., translate) the rule to the target string (**ssrrqqpp**).

It is important to point out that these stages are not preprogrammed; rather, they emerge from the lower-level probabilistic architecture of the system. This division of the processing into stages only roughly describes the program's behavior. The construction of a perceptual structure is divided into small steps, each step setting the stage for the next. Each step corresponds to a codelet, and since many codelets run in parallel, efforts towards building different structures are interleaved, sometimes cooperating and sometimes competing. In other words, these high-level tasks are carried out by interleaved chains of codelets, with each codelet being responsible for some very small action. Many chains -- some cooperating, some competing -- progress in parallel at different rates, the rate of each being set by the urgencies of its component codelets. Thus Copycat follows what we call the principle of the parallel terraced scan: simultaneously exploring promising avenues at high speeds and others at lower speeds (Hofstadter, 1983). Almost all codelets make one or more probabilistic decisions, and the high-level behavior of the program arises from the combination of thousands of these very small choices. In this way, Copycat's "style" arises from its low-level stochastic substrate.

Figure 4 shows nine screen printouts from an actual run of the program on the sample problem. The program was interrupted at various points during its run and these printouts of its graphics were made. They show how perceptual structures are formed in an interleaved manner, how competition between correspondences occurs, and how a compatible set of correspondences is settled upon and used to create an answer.

## 5. Present Status of Copycat

At present, the Copycat program can solve problems 1, 2, and 3 from the sample problems given at the beginning of this report, as well as several variants (including **abc** ---> **abd**, **ssrrqqpp** ---> ?), and should soon be able to solve problems 4 and 5. By "solve", we mean that the program is able to (on different runs) come up with all of the various answers that people give to each problem, including the very rigid answers, although it is much more likely to produce answers that we see as being more flexible and abstract. (Since the program's low-level workings are permeated by probabilistic choices, it often produces different answers on different runs.) It is possible to modify Copycat's behavior by varying parameters associated with the program; we can, for example, cause it to "prefer" more rigid answers (i.e., to produce them more often).

Problems 6 and 7 are beyond the program's present capabilities, and will necessitate some additions to the architecture beyond what has been described above (see Hofstadter, Mitchell, and French, 1987).

## 6. Conclusion

In this paper we have described the Copycat program as it is currently implemented. We are now extending its capabilities and also testing its generality by using the same architecture in different domains (Hofstadter, Mitchell, and French, 1987). Our long-term goal is, of course, to create mechanisms for doing analogies in their full generality and in any domain. We are not at present dealing with the important question of how people retrieve memories in order to construct an analogy with a given situation (Carbonell, 1983; Gick and Holyoak, 1983; Schank,1983); in Copycat the two situations are merely presented. We are also not trying to make our programs create new concepts (i.e., to add new nodes to the Slipnet). In that sense, we are not modeling learning. On the other hand, our work does involve learning, if that term is taken to include the automatic generalization from experience that humans perform in novel contexts (Holland, 1986; Holland et al., 1986). Our focus is, above all, to understand the nature and interaction of human concepts. We believe that the mechanisms we are developing are psychologically realistic; the ultimate test is, of course, how well they mirror human performance over a broad range of problems in many domains.


### Acknowledgements

Our special thanks go to Robert French for many important contributions to this project. David Rogers and Gray Clossman have also given us invaluable assistance. In additon, we thank the following people for helpful discussions: Daniel Defays, Dedre Gentner, Robert Hofstadter, John Holland, Greg Huber, John Laird, Roy Leban, Wayne Loofbourrow, Alejandro López, Larry Maloney, David Moser, Ed Smith, and Henry Velick. This research has been supported by a grant from the University of Michigan, a grant from Mitchell Kapor, Ellen Poss, and the Lotus Development Corporation, a grant from Apple Computer, Inc., and grant DCR 8410409 from the National Science Foundation.


### References


[1] Anderson, J.R. (1983). The architecture of cognition. Cambridge, MA: Harvard University Press.

[2] Bongard, M. (1970). Pattern recognition. Hayden Book Co. (Spartan Books).

[3] Burstein, Mark H. (1986). Conceptual formation by incremental analogical reasoning and debugging. In R. Michalski et al. (Eds.), Machine learning: An artificial intelligence approach: Vol. 2. Los Altos, CA: Morgan Kaufmann.

[4] Carbonell, Jaime G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. Michalski et al. (Eds.), Machine learning: An artificial intelligence approach. Palo Alto, CA: Tioga.

[5] Collins, A. M. and E. F. Loftus (1975). A spreading-activation theory of semantic memory. Psychological Review, 82, 407-428.

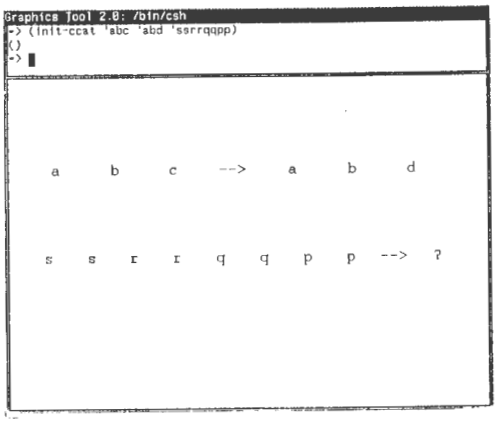
[6] Darden, Lindley (1983). Reasoning by analogy in scientific theory construction. Proceedings of the International Machine Learning Workshop. Monticello, Ill.

[7] Erman, L.D., F. Hayes-Roth, V. R. Lesser, and D. Raj Reddy (1980). The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. Computing Surveys, 12(2), 213-253.

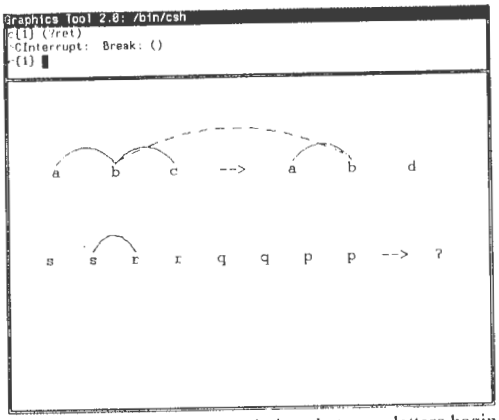
[8] Evans, Thomas G. (1968). A program for the solution of a class of geometric-analogy intelligence-test questions. In M. Minsky (Ed.), Semantic information processing. Cambridge, MA: MIT Press.

[9] Feldman, J. and D. Ballard (1982). Connectionist models and their properties. Cognitive Science, 6(3), 205-254.
[10] Genesereth, Michael R. (1980). Metaphors and models. Proceedings of the First Annual National Conference on Artificial Intelligence , 208-211. Menlo Park, CA: American Association of Artificial Intelligence.
[11] Gentner, Dedre (1983). Structure-mapping: A theoretical framework for analogy. Cognitive Science, 7(2).
[12] Gick, Mary L. and Keith J. Holyoak (1983). Schema induction and analogical transfer. Cognitive Psychology, 15, 1-38.
[13] Greiner, Russell (1985). Learning by understanding analogies (Technical Report STAN-CS-85-1071). Stanford, CA: Stanford University, Computer Science Department.
[14] Hofstadter, Douglas R. (1983). The architecture of Jumbo. Proceedings of the International Machine Learning Workshop. Monticello, Ill.
[15] Hofstadter, Douglas R. (1984a). Simple and not-so-simple analogies in the Copycat domain. Unpublished FARG Document, University of Michigan, Ann Arbor, MI.
[16] Hofstadter, Douglas R. (1984b). The Copycat project: An experiment in nondeterminism and creative analogies (AI Memo #755). Cambridge, MA: MIT AI Laboratory.
[17] Hofstadter, Douglas R. (1985a). Analogies and roles in human and machine thinking. In Metamagical Themas (pp. 547-603). New York: Basic Books.
[18] Hofstadter, Douglas R., Melanie Mitchell, and Robert French (1987). Fluid concepts and creative analogies: A theory and its computer implementation. Technical Report 10, Cognitive Science and Machine Intelligence Laboratory, University of Michigan, Ann Arbor, MI.
[19] Holland, John, Keith J. Hoyoak, Richard E. Nisbett, and Paul R. Thagard (1986). Induction. Cambridge, MA: Bradford/MIT Press.
[20] Holland, John (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski et al. (Eds.), Machine learning: An artificial intelligence approach: Vol. 2. Los Altos, CA: Morgan Kaufmann.
[21] Holyoak, Keith J. and Paul Thagard (1987). Analogical mapping by constraint satisfaction. Manuscript submitted for publication.
[22] Kahneman, Daniel and Dale T. Miller (1986). Norm theory: Comparing reality to its alternatives. Psychological Review, 93(2), 136-153.
[23] Kedar-Cabelli, Smadar (1985). Purpose-directed analogy. In Proceedings of the Cognitive Science Society. Irvine, CA.
[24] Lakoff, G. and M. Johnson (1980). The metaphorical structure of the human conceptual system. Cognitive Science, 4 (2), 195-208.
[25] McClelland, J. and D. Rumelhart (1986) (Eds.). Parallel distributed processing. Cambridge, MA: Bradford/MIT Press.
[26] Meredith, Marsha J. (1986). Seek-Whence: A model of pattern perception. Unpublished doctoral dissertation, Indiana University, Computer Science Department.
[27] Norman, D. and D. Rumelhart (1982). Simulating a skilled typist: A study of skilled cognitive-motor performance. Cognitive Science, 6(1).
[28] Quillian, Ross (1968). Semantic Memory. In M. Minsky (Ed.), Semantic information processing. Cambridge, MA: MIT Press.
[29] Reitman, Walter (1965). Cognition and thought. New York: Wiley.
[30] Rosch, E., C.B. Mervis, W.D. Gray, D.M. Johnson, and P. Boyes-Braem (1976). Basic objects in natural categories. Cognitive Psychology,8, 382-439.
[31] Schank, Roger (1983). Dynamic memory. Oxford University Press.
[32] Smith, E. E. and D. L. Medin (1981). Categories and concepts. Cambridge, MA: Harvard University Press.
[33] Winston, Patrick Henry (1980). Learning and reasoning by analogy. Communications of the ACM, 23(12), 689-703.

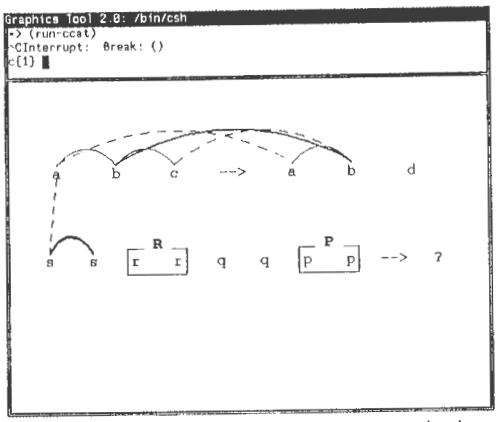
*Figure 4: A run of the program on* **abc** --> **abd**, ***ssrrqqpp*** --> *?* This run produced the answer **ttrrqqpp**, but note that since the program is non-deterministic, different runs can produce different answers. In particular, the answer **ssrrqqoo** is produced quite often, and on rare occasions, rigid answers such as **ssrrqqpq** and **ssrrqqpd** are produced.



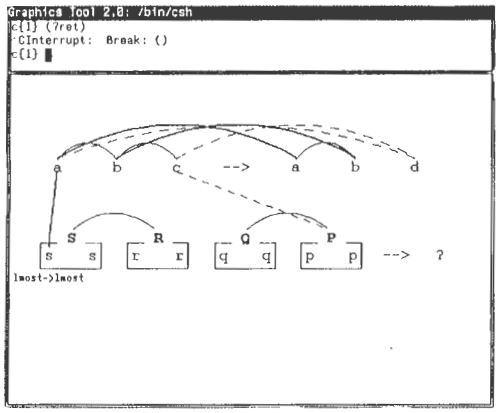

1. The program is presented with the three strings.



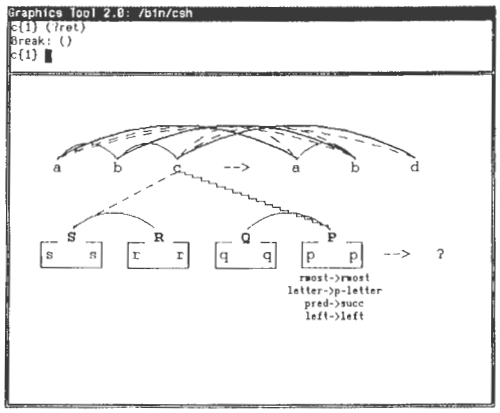

2. Successor-predecessor relations between letters begin to be noticed (represented by small arcs between letters) and the program tentatively considers a correspondence between the two **B**'s (dashed arc).



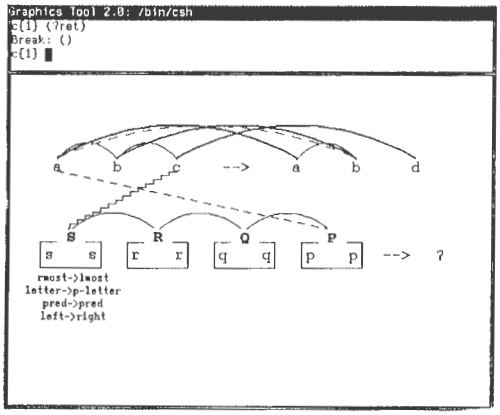

3. Sameness relations and groups in **ssrrqqpp** begin to be noticed and the correspondence between the two **B**'s has been built (solid arc). Other tentative correspondences are being considered, including a correspondence between the **A** and the **S** (dashed line).



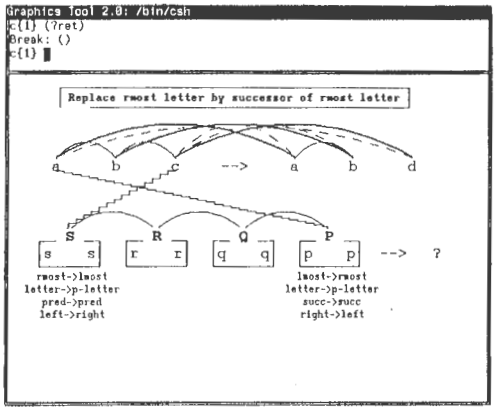

4. Successor-predecessor relations have been noticed between some of the parameter-letters. A correspondence has been built between the **A** and the leftmost **S** (represented by a jagged line; its single slippage, "leftmost --> leftmost", appears below it). A competing tentative correspondence between the **C** and the parameter-letter ***P*** is being considered (dashed line).



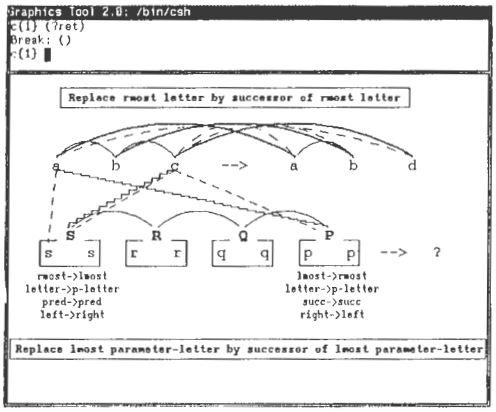

5. A fight has occurred between the **A-S** correspondence and the **C-*P*** correspondence, and the latter has won and destroyed the former. A competing correspondence between the **C** and the parameter-letter ***S*** is being considered. The mapping between **abc** and **abd** is now complete, although alternative "horizontal" correspondences continue to be considered.



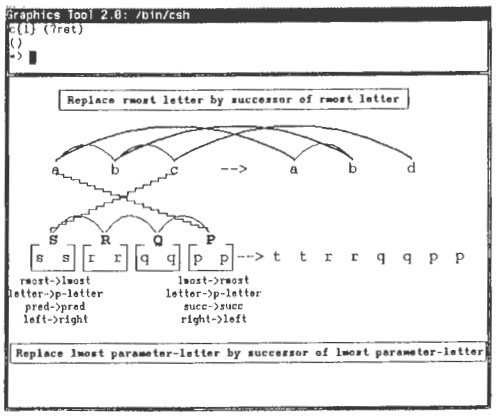

6. The **C-*S*** correspondence has been built, destroying the **C-*P*** correspondence. The compatible **A-*P*** correspondence is being considered.




7. The **A-*P*** correspondence has been built. In addition, a rule describing the change from **abc** to **abd** has been created (in box at top of screen).




8. The rule has been translated (the translated rule appears at the bottom of the screen). Competing tentative correspondences continue to be considered.




9. An answer has been created from the translated rule.

# The Complexity of Model-Preference Default Theories

**Bart Selman**
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
bart@ai.toronto.edu

**Henry Kautz**
AT&T Bell Laboratories
AI Principles Research Department
Murray Hill, NJ 07974
kautz@allegra.att.com

## Abstract

Most formal theories of default inference have very poor computational properties, and are easily shown to be intractable, or worse, undecidable. We are therefore investigating limited but efficiently computable theories of default reasoning. This paper defines systems of Propositional Model Preference Defaults, which provide a true model-theoretic account of default inference with exceptions. Model Preference theories are not identical to Default Logic or Circumscriptive theories, but some of our complexity results extend to those formalisms.

The most general system of Model Preference Defaults is decidable but still intractable. Inspired by the very good (linear) complexity of propositional Horn theories, we consider systems of Horn Defaults. Surprisingly, finding a most-preferred model in even this very limited system is shown to be NP-Hard. Tractability can be achieved in two ways: by eliminating the "specificity ordering" among default rules, thus limiting the system's expressive power; and by restricting our attention to systems of Acyclic Horn Defaults. These acyclic theories can encode inheritance hierarchies of the form examined by Touretzky, but are strictly more general.

This analysis suggests several directions for future research: finding other syntactic restrictions which permit efficient computation; or more daringly, investigation of default systems whose implementations do not require checking global consistency – that is, fast "approximate" inference.

Keywords: nonmonotonic reasoning, knowledge representation.

# I. Introduction

It is commonly acknowledged that an agent need not, indeed cannot, have absolute justification for all of his beliefs. An agent often assumes, for example, that a certain member of a particular kind (*e.g.*, Tweety the bird) has a certain property (*e.g.*, the ability to fly) simply because it is typically true that entities of that kind have that property. When formulating a plan of action, an agent often assumes that certain acts will lead to certain consequences, when in fact those consequences are not guaranteed because the world may be in some unusual state. In order to assimilate information about its environment, an agent will often use a strategy of "hypothesize and test", and adopt a *particular* model of those inputs, rather than maintaining a representation of *all logically possible* interpretations.

Such default reasoning seems to offer several advantages. It allows an agent to come to a decision and act in the face of incomplete information. It provides a way of cutting off the possibly endless amount of reasoning and observation that the agent might perform in order to gain perfect confidence in its beliefs. And, as Levesque (1986) argues, default reasoning may greatly reduce the complexity of regular deduction. Defaults can be used to "flesh out" an incomplete knowledge base to a *vivid* one; that is, a set of atomic formulas which completely characterize a domain. Once a vivid knowledge base is obtained, deduction reduces to standard database lookup.

A satisfactory *formal* theory of default reasoning should therefore both *model* what an agent could come to believe on the basis of given facts and default assumptions, *and* precisely characterize the very real *efficiency* of default reasoning over pure deduction (or classical probability theory, for that matter). While there is some dispute (Hanks and McDermott 1986) as to the representational adequacy of such proposed formal systems as Default Logic (Reiter 1980) or Circumscription (McCarthy 1980), no

one is prepared to defend their abysmal *computational* properties. All are easily shown to be undecidable in the first-order case, and badly intractable in the propositional case.

We are therefore investigating limited but efficiently computable theories of default reasoning. Such results are of interest even if one intends to implement the default reasoning system on a massively parallel machine. As Levesque (1986) points out, the processing requirements of an exponentially-hard problem can quickly overwhelm even enormous arrays of processors, equal in size to the number of neurons of the brain.

Our interest in using defaults to generate vivid models is a particular reason for our concern with complexity results. It is hardly of interest to eliminate the exponential component of deductive reasoning by introducing an even more costly process of transforming the representation into a vivid form.

The number and variety of formal default systems presents an immediate obstacle to the problem of determining the complexity of the *task* of default inference itself. Who is to say, for example, that a problem which is intractable when formulated in theory A is not tractable when formulated in theory B? Etherington (1986) has demonstrated that one should not simply lump all default theories together, as they differ significantly in both their expressive power and the kinds of conclusions they justify. Part of the problem in comparing default theories is their primarily syntactic characterization; indeed, even the semantic accounts provided in the literature retain a strong syntactic flavour (Etherington 1987).

This paper defines a straightforward way of encoding defaults by stating a *preference ordering* over the space of all possible models. This ordering is defined by statements of the form, "a model where $\alpha$ holds is to be preferred over one where $\beta$ holds." The details of this system of Model Preference Defaults are spelled out below. The task of the default inference process is to find a *most preferred* model.

This theory provides a true semantic characterization of default inference; it is important to note that it is *not* a "semantics" which simply mimics the sequential application of syntactic rules. One benefit of this model-theoretic foundation is the ease with which one can incorporate a general *specificity ordering* over defaults. As will be seen, the specificity ordering makes sense of *exceptional defaults*, and is at the heart of probability theory's notion of the *reference class* (Kyburg 1983). Various kinds of Model Preference theories have expressive power comparable (but not strictly identical) to special cases of Default Logic or Circumscription, and some of our *negative* complexity results carry over to those formalisms.

The propositional version of Model Preference Default theory is decidable but still intractable. Inspired by the very good (linear) complexity of propositional Horn theories, we next consider systems of specificity-ordered Horn Defaults over initially-empty knowledge bases. Surprisingly, finding a most-preferred model in even this very limited system is shown to be NP-Hard. Tractability is finally achieved by restricting our attention to systems of Acyclic Horn Defaults. These acyclic theories can encode inheritance hierarchies of the form examined by Touretzky, but are more general.

The final section of this paper considers the consequences of this complexity analysis. One reaction may be to search for other syntactic restrictions on default theories which permit efficient computation. A more daring venture would be to investigate default systems which do not require the existence of a single model of the entire theory. Such systems might be able to perform fast "approximate" inference.

# II. Model Preference Defaults

What is the meaning of a default rule? A common approach (*e.g.*, Reiter's Default Logic) is to take it to be similar to a deductive rule, but with the odd property of possessing a *global* (and perhaps non-computable) applicability condition. The *conclusions* of such a system can only be defined by examining the *syntactic structure* of particular proofs. There is a very different interpretation of default rules, however, with a natural and intuitive semantics, which is independent of the details of the proof theory. This approach is to use rules to define *constraints* on the set of *preferred* (or *most likely*) *models* of a situation. The *goal* of default inference is then to find a most preferred model (of which there may be many), but the details of the syntactic processes employed are separate from the model's semantic characterization.

Unlike previous approaches, the result of Model Preference Default inference is always a *complete* model; an appropriate result given our goal of obtaining a vivid representation as described above. By contrast, a Default Logic proof arrives at an *extension*, that is, a set of formulas which only *partially* characterizes a situation.

The model theory for Circumscription is similar to that for Model Preference Defaults, in that it involves considering models which are maximal in some partial order. They differ, however, in that the conclusions of a circumscriptive proof must hold in *all* maximal models, and in the fact that the partial order in a circumscriptive theory is defined *solely* in terms of minimizing predicates. The first difference makes circumscriptive theory (perhaps too) cautious, while the second leads, at times, to unnatural complexity in encoding default knowledge in terms of predicate minimization. The work of Shoham (1986) on default reasoning involving time and his unifying framework for nonmonotonic reasoning (Shoham 1987) appear to be quite similar to our own, in the emphasis on a semantic theory based on partially-ordered models; it remains to be seen how comparable our systems are in expressive power.

We define a series of default systems, beginning with a general but weak system $\mathcal{D}$, add a specificity-ordering over defaults to obtain $\mathcal{D}^+$, then restrict to Horn-form defaults to yield $\mathcal{DH}$ and $\mathcal{DH}^+$, and finally consider acyclic sets of default rules $\mathcal{DH}^+_{\mathrm{a}}$. This paper considers only purely propositional systems; a later paper will provide a straightforward extension to include propositional schemas.

## Definitions

Let $P = \{p_1, p_2, ....p_n\}$ be a set of propositional letters, and $\mathcal{L}$ be a propositional language built up in the usual way from the letters in $P$ and the connectives $\neg$ and $\wedge$ (the others will also be used freely as syntactic abbreviations). Also, let $q$ and $r$ be single literals (a literal is propositional letter $p \in P$ or its negation $\neg p$ written as $\overline{p}$), and $\alpha$ and $\beta$ be (possibly empty) sets of literals.

**Definition:** Model.
A model (or truth assignment) $M$ for $P$ is a function $t : P \rightarrow \{\mathrm{T}, \mathrm{F}\}$. $M$ satisfies a set $S$ of formulas of $\mathcal{L}$ (written as $M \models S$) iff $M$ assigns T to each formula in the set. Complex formulas are evaluated with respect to $M$ in the standard manner.

**Definition:** Default Rule.
A default rule $d$ is a rule of the form $\alpha \rightarrow q$. The rule $d$ is a Horn-form default rule iff $\alpha$ contains only positive literals.

**Definition:** Applicability.
A default rule $d$, of the form $\alpha \rightarrow q$, is applicable at a model $M$ iff

1. $M \models \alpha$, and
2. $d$ is not blocked at $M$. (For the definition of blocking see the description of the Specificity Condition given below.)

If $d$ is applicable at $M$, then the application of rule $d$ at $M$ leads to a model $d(M)$. This model is identical to $M$ with the possible exception of the truth assignment to the letter occuring in the literal $q$; this letter is assigned a truth value such that $d(M) \models q$.

**Definition:** Ordering on models.
Given a set of default rules $D$, the relation $\leq^+$ between pairs of models is defined as follows: $M \leq^+ M'$ iff $\exists d \in D$ s.t. $d(M) = M'$. The relation $\leq$ between pairs of models is defined as the reflexive transitive closure of $\leq^+$. Let $M > M'$ be defined as $[(M' \leq M) \wedge \neg(M \leq M')]$.

**Definition:** Maximal Model.
$M$ is a maximal model w.r.t. a set of defaults $D$ iff $\neg\exists M' \, (M' > M)$.

**Definition:** Default System $\mathcal{D}$.
In default system $\mathcal{D}$ we consider problems of the the following form: find an arbitrary model for a given set of propositional letters $P$ which is maximal according to a given set of defaults, while ignoring condition 2 of the definition of applicability.

For example, suppose that $P$ is {*student*, *adult*, *employed*}, with the intended interpretations "*this person* is a university student", "*this person* is an adult", and "*this person* is employed" (example from Reiter and Criscuolo 1983). Then the defaults "Typically university students are adults", "Typically adults are employed", and "Typically university students are not employed" can be captured as follows:[1]

1) *student* $\rightarrow$ *adult*
2) *adult* $\rightarrow$ *employed*
3) *student* $\rightarrow \overline{\textit{employed}}$

[1]We omit the set braces in the left-hand side of the default rules.

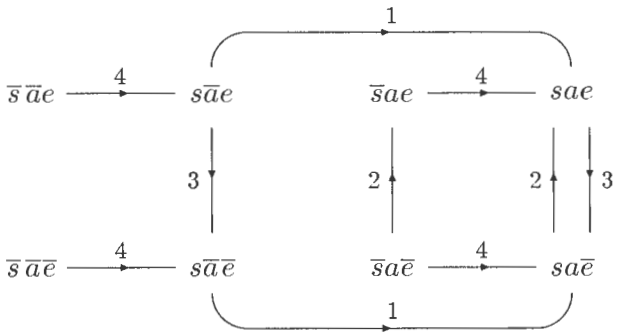



Figure 1: The preference ordering on models as given by the default rules 1) – 4).

So, for example, rule 1 says that when given two models that assign T to *student* and that differ only in the truth-assignment of *adult*, give preference to the model with *adult* assigned T. The default which says *this person* is a university student can be encoded by:[2]

4) $\emptyset \rightarrow$ *student*

Figure 1 gives the preference ordering on the models as defined by these defaults rules. We use the obvious abbreviations for the propositional letters in $P$. Thus, for example, $sa\overline{e}$ stands for the model in which both *student* and *adult* are assigned T and *employed* is assigned F. A directed edge from a model $M_a$ to a model $M_b$ indicates that $M_a \leq^+ M_b$. The numbers alongside the directed edges indicate the corresponding default rules.

We see that the model $sa\overline{e}$ is maximal, since there is no model that is *strictly* preferred over this model (as a matter of fact, for all other models $M^\star$, such as for example $s\overline{a}c$, we have $M^\star \leq sa\overline{e}$).

There is a maximal model in this system, however, that does not correspond to our intuitive understanding of the situation. This model is related to the "multiple extension" problem which has created much trouble in previous work on default reasoning (Hanks and McDermott 1986). Because $\mathcal{D}$ does not capture the notion that the third rule above should override the second, the model *sae* is also maximal.

Therefore we define a stronger default system which includes the notion that a more specific default overrides a less specific one.

**Definition:** Specificity Condition.
Given a set of defaults $D$, a default rule $d$ of the form $\alpha \rightarrow q$ is *blocked* at $M$ iff $\exists d' \in D$ of the form $(\beta \cup \alpha) \rightarrow \overline{q}$ and $M \models (\beta \cup \alpha)$.

From the above definitions, it follows that a model preference ordering is completely determined by the default rules; a theory has no effect on the ordering. (See also the definition of a maximal model w.r.t. a set of defaults and a non-empty theory at the end of this section.)

**Definition:** Default System $\mathcal{D}^+$.
In default system $\mathcal{D}^+$ we consider problems of the following form: find an arbitrary model for a given set of propositional letters $P$ which is maximal according to a given set of defaults, where rules may be blocked by the specificity condition (*i.e.*, both conditions of the definition of applicability are taken into account).

The first example is now more completely captured in $\mathcal{D}^+$ as follows.

$$\begin{array}{l} \textit{student} \rightarrow \textit{adult} \\ \textit{adult} \rightarrow \textit{employed} \\ \textit{student}, \textit{adult} \rightarrow \overline{\textit{employed}} \\ \emptyset \rightarrow \textit{student} \end{array}$$

The *only* maximal model is now $sa\overline{e}$.[3] (The graph representing the preference ordering is identical to the one in figure 1, without the arc labeled 2 from $sa\overline{e}$ to *sae* and the arc labeled 3 from $s\overline{a}e$ to $s\overline{a}\overline{e}$.)

While $\mathcal{D}^+$ appears to have adequate expressive power to handle the standard examples of default reasoning, we will see that it does not succumb to a tractable algorithm. Therefore we define the following restricted classes of default problems.

**Definition:** Default Systems $\mathcal{DH}$ and $\mathcal{DH}^+$.
In $\mathcal{DH}$ we are concerned with the set of problems in system $\mathcal{D}$ involving only Horn-form default rules; and likewise for $\mathcal{DH}^+$ w.r.t. $\mathcal{D}^+$.

**Definition:** Acyclic Defaults.
Define the directed graph $G(D) = (V, E)$ associ-

[2] Instead of adding default rule 4 to the set of defaults, one can express the fact that *this person* is a university student by having the propositional formula *student* in the theory. (The notion of a maximal model w.r.t. a set of defaults and a non-empty theory is defined below.)

[3] In this example a different solution to the the problem of multiple extensions that does not rely on specificity ordering would be to instead replace rule 2 by $\textit{adult}, \overline{\textit{student}} \rightarrow \textit{employed}$. However, specificity ordering captures nicely the intuition behind property inheritance, namely that properties inherited from more general concepts can be overridden by properties inherited from more specific concepts (more generally: more specific defaults should override less specific ones).

ated with a set of default rules $D$ as follows:[4] the $V$ contains a vertex labeled $p_i$ for each propositional letter $p_i$ in $P$, and $E = \{(p_i, p_j) \mid \exists d \in D$ of the form $\alpha \rightarrow q$ s.t. $\{[(p_i \in \alpha) \vee (\overline{p_i} \in \alpha)] \wedge [(p_j = q) \vee (\overline{p_j} = q)]\}$. A set of defaults $D$ is called acyclic iff the $G(D)$ is an acyclic directed graph.

The two sets of defaults discussed above are examples of sets of acyclic defaults. They encode basic examples of acyclic inheritance hierarchies (Touretzky 1986). Acyclic theories can encode such hierarchies, but are more general.

Note that there are also natural examples that do not fall into the class of acyclic default systems, such as those obtained by adding the default rule $adult \rightarrow \overline{student}$ to the sets of defaults given above.

**Definition:** Default System $\mathcal{DH}_a^+$.
In $\mathcal{DH}_a^+$ we are concerned with the set of problems in system $\mathcal{DH}^+$ involving only acyclic sets of defaults.

While problems of property inheritance fall within $\mathcal{DH}_a^+$, they do not completely circumscribe it. In Selman and Kautz (1988) we present an example of an acyclic preference default theory which cannot be easily represented in terms of property inheritance.

Finally, we consider the case in which we have apart from a set of defaults $D$ also a non-empty set of facts $T$.

**Definition:** Maximal model w.r.t. $D$ and $T$.
Let $D$ be a set of defaults and $T$ a set of propositional formulas. A model $M$ is maximal w.r.t. $D$ and $T$ iff $(M \models T) \wedge \neg\exists M' [(M' > M) \wedge (M' \models T)]$.[5]

# III. Computational Complexity

We defined a notion of default reasoning based on a model preference ordering. As stated above, the goal of default inference is to find a maximal model given a set of facts and a partial ordering on the models as defined by a set of default rules. Because of our interest in tractable forms of default reasoning, a central question is: what is the computational cost of finding such a model?

The problem of finding a maximal model is clearly decidable, since one can simply scan the directed graph representing the partial order on models for a maximal model w.r.t. the defaults and the set of facts. We proceed by analysing the computational complexity of finding such a model. First we consider the general system $\mathcal{D}$.

The following theorem shows that the problem of finding a maximal model[6] given an arbitrary set of defaults $D$ is computationally intractable (provided P $\neq$ NP):

**Theorem 1.** The search problem $\mathcal{D}$ is NP-hard.

The proofs of the theorems in this section are given in Selman and Kautz (1988), here we will only give a high-level description of them.

The proof of theorem 1 is based on a Turing reduction from 3-Satisfiability (Borgida 1986). In the proof it is shown how a propositional formula $\alpha$ in conjunctive normal form with three literals per clause can be translated in polynomial time into a set of default rules $D(\alpha)$ with the property that if $\alpha$ is satisfiable then all maximal models (ignoring condition 2 of the definition of applicability) of $D$ will satisfy $\alpha$. Now, consider an algorithm that takes as input a formula $\alpha$ (an instance of 3-Satisfiability) and constructs $D(\alpha)$, then calls an oracle that returns in constant time a maximal model $M$ of this set of defaults, and, finally, returns "yes" if $M$ satisfies $\alpha$ and "no" otherwise. The algorithm returns "yes" iff $\alpha$ is satisfiable; it runs in polynomial time. Therefore, finding a maximal model is NP-hard.

[4] This graph should not be confused with a graph like the one in figure 1 which makes explicit the ordering on the models.

[5] Given a model $M$ that satisfies a non-empty theory, there may exist a model $M'$ such that $M'$ does not satisfy the theory and $M' > M$. Therefore, we only require a maximal model w.r.t. a set of defaults and a non-empty theory to be maximal w.r.t. the set of models that satisfy the theory.

[6] We are interested in an algorithm that handles an arbitrary problem in $\mathcal{D}$. Therefore, we consider the search problem (Garey and Johnson 1979) associated with $\mathcal{D}$. A search problem $\Pi$ is defined as a set of finite objects $S_\Pi$ called instances, and for each instance $I \in S_\Pi$ a set of finite objects $S[I]$ called solutions of $I$. An algorithm is said to solve a search problem if it returns the answer "no" whenever $S[I]$ is empty and otherwise returns some arbitrary solution belonging to $S[I]$.

With each system of defaults $\mathcal{X}$ defined above one can associate in a straightforward manner a search problem $\mathcal{X}_s$. *E.g.*, an instance $I$ of the search problem $\mathcal{D}_s$ associated with Problem Class $\mathcal{D}$ is a set of propositional letters $P$ and a set of default rules $D$. $S[I]$ is the set of maximal models for $P$ w.r.t. $D$ (ignoring condition 2 in the definition of applicability). To keep our notation simple, we omit the subscript s. So, theorem 1 states that, provided P $\neq$ NP, there exists no polynomial algorithm that, given as input a set of defaults $D$, finds an arbitrary maximal model (ignoring the specificity ordering).

Given the very good complexity (linear, Dowling and Gallier 1984) of propositional Horn theories, we now turn our attention to default system $\mathcal{DH}$. According to the following theorem such defaults can indeed be handled efficiently:

**Theorem 2.** There exists a linear algorithm for the search problem $\mathcal{DH}$.

In Selman and Kautz (1988) we give a hill-climbing algorithm with linear time complexity and prove its correctness. We also show how the algorithm can be modified to take into account a non-empty theory consisting of a set of literals. The resulting algorithm is non-linear, but still polynomial.

We now consider the influence of the specificity condition (necessary to handle exceptions properly in default reasoning). This leads to the following surprising result:

**Theorem 3.** The search problem $\mathcal{DH}^+$ is NP-hard.[7]

The essence of the proof lies in transforming the set of default rules as used in the proof of theorem 1 into a set of Horn-form defaults. We therefore replace negative literals by new letters, *e.g.*, $\overline{p}$ is replaced by $p'$. We then add extra sets of Horn-form default rules that guarantee that when the original formula $\alpha$ is satisfiable, no maximal model will assign the same truth value to a pair of corresponding letters, such as $p$ and $p'$. The specificity condition is essential for this result.

Theorems 2 and 3 show how a relatively small change in expressive power of a tractable representation can lead to a computationally intractable system. Results like this show the importance of a detailed analysis of the computational properties of knowledge representation and reasoning systems (Levesque and Brachman 1985).

The following result is another illustration of the tradeoff between expressiveness and tractability:

**Theorem 4.** Given a set of Horn-form defaults *DH* (no specificity ordering) and a theory *TH* consisting of a set of Horn formulas, the problem of finding a maximal model w.r.t. *DH* and *TH* is NP-hard.

This result is of interest because of the fact that both propositional Horn-form defaults without specificity ordering (Theorem 2) and Horn theories by themselves are linear.

Finally, we can again obtain a tractable system by limiting our default systems to acyclic ones:

**Theorem 5.** There is a polynomial algorithm for the search problem $\mathcal{DH}_a^+$.

In Selman and Kautz (1988) we give a polynomial time algorithm and prove its correctness. The algorithm can be adapted to handle non-empty theories consisting of a set of literals.[8]

# IV. A Comparison to Default Logic

In this section we compare model preference defaults to Default Logic (Reiter 1980) and demonstrate how our complexity results may be used to analyze the complexity of default reasoning systems based on Default Logic. In Selman and Kautz (1988) we give a more detailed comparison, and also a similar analysis of the relation to circumscription (McCarthy 1980).

We first discuss an example of the translation of a set of preference default rules $D$ into a set of default rules $D_{dl}$ of Default Logic. The translation is done is such a way that there will be a one-to-one correspondence between maximal models of $D$ and extensions of $D_{dl}$ ($T = \emptyset$).

Consider the set of defaults $D$ used in the example illustrating specificity ordering (section II). The corresponding set $D_{dl}$ contains two groups of rules. The first group consists of rules that correspond to the model preference defaults:[9]

$$\left\{\frac{s:a}{a}, \frac{s \wedge a:\overline{e}}{\overline{e}}, \frac{a:e \wedge \overline{s}}{e}, \frac{:s}{s}\right\}.$$

(Note the use of a semi-normal default in the third rule to enforce a specificity ordering.) The second group of rules guarantees that the only extensions of $D_{dl}$ are complete models:

$$\left\{\frac{:a}{a}, \frac{:\overline{a} \wedge \overline{s}}{\overline{a}}, \frac{:e \wedge \overline{(s \wedge a)}}{e}, \frac{:\overline{e} \wedge \overline{a}}{\overline{e}}\right\}.$$

[7] As a direct consequence it follows that the search problem $\mathcal{D}^+$ is NP-hard.

[8] Although we expect that theories consisting of Horn formulas can also be handled efficiently, we have yet to find a polynomial-time algorithm for this case.

[9] Again, we use the obvious abbreviations.

These defaults can be viewed as a set of closed world assumptions (Reiter 1978) that "force" the system to decide on each letter. (Unlike other cases of closed world assumptions, no preference is given to negative information.) $D_{dl}$ has only one extension,[10] $Th\{s, a, \overline{e}\}$, corresponding to the only maximal model of $D$.

The correspondence between sets of model preference rules and default logic rules can be used to show the intractability of certain classes of semi-normal default rules by reductions from intractable model preference systems. The reduction process is somewhat complicated by the fact that the straightforward translation process used in the example above only applies to restricted sets of model preference rules.[11] However, it turns out that the process is applicable to the set of rules used in the proof of the intractability of $\mathcal{DH}^+$. Therefore, it follows that the problem of finding a complete set of literals that lies within an extension of a set of semi-normal defaults is NP-hard, even if we restrict the rules to the form $[\alpha : (q \wedge \beta)]/q$ where $\alpha$ is a conjunction of positive literals, $\beta$ is a formula in conjunctive normal form with at most two literals per clause, and and $q$ is a single literal.[12]

# V. Conclusions

We introduced a system for default reasoning based on a model preference relation. A maximal model in such a system is a complete description of a preferred or most likely state of affairs, based on incomplete information and a set of defaults. Unlike most other approaches to default reasoning, ours is purely semantic, and is defined independent of syntactic notions.

The goal of our work is to develop tractable methods of default reasoning, for use in fast reasoning systems which represent knowledge in a vivid form. Therefore, we only allow complete models as default conclusions. Model preference theories seem to be of interest, however, beyond this one application. The specificity ordering on defaults, a crucial component of any kind of default reasoning, is neatly captured in $\mathcal{D}^+$ and its subtheories. Another natural application for model preference theories is to encode a logic of choice, whereby an an agent chooses which of his *goal states* is most preferred.

We presented a detailed analysis of complexity properties of the various model preference default systems. The analysis indicates that only systems with quite limited expressive power lead to tractable reasoning (*e.g.*, $\mathcal{DH}$ and $\mathcal{DH}^+_{\mathrm{a}}$). We also gave an example of how a relatively small change in the expressive power of a tractable system can lead to intractability (from $\mathcal{DH}$ to the intractable $\mathcal{DH}^+$).

Acyclic inheritance hierarchies can be represented in the tractable system $\mathcal{DH}^+_{\mathrm{a}}$. Classes of acyclic rules have also been singled out by others (*e.g.*, Touretzky (1986) on acyclic inheritance hierarchies, and related work by Etherington (1986) on ordered default theories) for their relatively good computational properties. A direct comparison with our approach is complicated by the fact that we do not allow for partial models. It should be noted that the precise computational properties of most of these other system have not been analyzed in detail (*e.g.*, it is an open problem whether computing an extension based on the notion of inferential distance is tractable (Touretzky 1987)).

The nature of model preference defaults was further illustrated by a comparison with Default Logic (Reiter 1980). In this comparison we showed how a set of semi-normal rules, which can be viewed as representing a special form of closed world assumptions, can be added to a set of default logic rules in a way that guarantees the extensions to be complete models. As part of the comparison we showed how complexity results for model preference can be used to obtain complexity results of restricted classes of other default reasoning systems.

This work suggests several directions for future research. One is the development of a first-order version of model preference defaults. Another is to allow for more expressive power and introduce some form of "approximate reasoning" to keep the system tractable. A search for other tractable sub-classes would be in order. And, finally, to determine the usefulness of the tractable systems we have identified, a further study of the forms of defaults necessary in real world do-

[10] Here $Th$ denotes closure under logical consequence.

[11] The complication arises from the definition of maximal model as a model that has no model that is strictly larger. Therefore, the models on a cycle in the graph corresponding to the partial orderings relation on models can all be maximal (if not strictly dominated by another model). In that case, a translation like the one given above can lead to a set of semi-normal defaults with no extensions.

[12] Defaults of this form are similar to, but somewhat more general than, those used in network default theories (Etherington 1986).

mains, *e.g.*, conventions in cooperative conversation (Perrault 1987), is needed.

## Acknowledgments

This work originated from research meetings organized by Ron Brachman and financially supported by AT&T Bell Laboratories. It was supported in part by a Government of Canada Award to the first author. We would like to thank Hector Levesque for providing the definition of model preference defaults and for introducing us to the complexity questions associated with these systems, and Alex Borgida for the complexity result for $\mathcal{D}$. We also thank Ron Brachman, Jim des Rivières, Dave Etherington, and Armin Haken for useful discussions and comments.

## References

Borgida, A. (1986) Personal communication, September 1986.

Dowling, W.F. and Gallier, J.H. (1984). Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming*, **3**, 1984, 267–284.

Etherington, D.W. (1986). Reasoning With Incomplete Information: Investigations of Non-Monotonic Reasoning. Ph.D. Thesis, University of British Columbia, Department of Computer Science, Vancouver, BC, Canada, 1986. Revised version to appear as: *Reasoning With Incomplete Information.* London: Pitman / Los Altos, CA: Morgan Kaufmann.

Etherington, D.W. (1987). A Semantics for Default Logic. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987, 495 – 498.

Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability, A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman, 1979.

Hanks, S. and McDermott, D. (1986). Default Reasoning, Non-Monotonic Logics, and the Frame Problem. *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, 1986, 328–333.

Kyburg, H. (1983). The Reference Class. *Philosophy of Science*, **50**, 1983, 374–397.

Levesque, H.J. (1986). Making Believers out of Computers. *Artificial Intelligence*, **30**, 1986, 81–108.

Levesque, H.J. and Brachman, J.R. (1985). "A Fundamental Tradeoff in Knowledge Representation and Reasoning (Revised Version)." In *Readings in Knowledge Representation* by R.J. Brachman and H.J. Levesque (Eds.), Los Altos, CA: Morgan Kaufmann, 1985, 41–70.

McCarthy, J. (1980). Circumscription – A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, **13**, 1980, 27–38.

Perrault, C.R. (1987). An Application of Default Logic to Speech Act Theory. Technical Report, SRI International, Artificial Intelligence Center, Palo Alto, CA, 1987.

Reiter, R. (1980). On Closed World Data Bases. In *Logic and Data Bases*, Gallaire, H. and Minker, J. (eds.), New York: Plenum Press, 1978.

Reiter, R. (1980). A Logic for Default Reasoning. *Artificial Intelligence*, **13**, 1980, 81–132.

Reiter, R. and Criscuolo, G. (1983). Some Representational Issues in Default Reasoning, *Computers & Mathematics with Applications*, (Special Issue on Computational Linguistics), **9** (1), 1983, 1 – 13.

Selman, B. and Kautz, H. (1988). Model-Preference Default Theories. Technical Report, University of Toronto, Department of Computer Science, Toronto, Ont., Canada, 1988.

Shoham, Y. (1986). Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence. Ph.D. Thesis, Yale University, Computer Science Department, New Haven, CT, 1986.

Shoham, Y. (1987). Nonmonotonic Logics: Meaning and Utility. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987, 388– 393.

Touretzky, D.S. (1986). *The Mathematics of Inheritance Systems.* Research Notes in Artificial Intelligence. London: Pitman / Los Altos, CA: Morgan Kaufmann, 1986.

Touretzky, D.S. (1987) Personal communication, July, 1987.

# Instance-Based Prediction of Real-Valued Attributes

Dennis Kibler David W. Aha

*Department of Information & Computer Science*
*University of California, Irvine*
*Irvine, CA 92717 U.S.A.*
*kibler@ics.uci.edu aha@ics.uci.edu*

## Abstract

Instance-based representations have been applied to numerous classification tasks with a fair amount of success. These tasks predict a symbolic class based on observed attributes. This paper presents a method for predicting a *numeric* value based on observed attributes. We prove that if the numeric values are generated by continuous functions with bounded slope, then the predicted values are accurate approximations of the actual values. We demonstrate the utility of this approach by comparing it with standard approaches for value-prediction. The approach requires no background knowledge.

Keywords: incremental learning, prediction, instance-based, continuous functions

## 1 Introduction

Instance-based learning (IBL) strategies represent concepts using sets of instances and a similarity metric, where each instance is described in terms of a set of attribute-value pairs. IBL techniques have been applied to several learning problems, including speech recognition (Bradshaw, 1987), word pronunciation (Stanfill & Waltz, 1986; Stanfill, 1987), handwritten symbol recognition (Kurtzberg, 1987), thyroid disease diagnosis (Kibler & Aha, 1987), and the cart-pole balancing problem (Connell & Utgoff, 1987). In each case, IBL techniques were shown to be computationally inexpensive methods for solving classification tasks. Most IBL applications involved the prediction of symbolic values. Connell and Utgoff, however, applied their CART system to predict values for a numeric domain, namely the degree of desirability of instances (which represent states in their domain). In this paper we present a method for applying instance-based techniques for predicting numeric values. Theoretical and empirical arguments are supplied to support our claims.

Most of the published work on learning from examples tasks involved the prediction of symbolic values (Vere, 1980; Mitchell, 1982; Quinlan, Compton, Horn, & Lazurus, 1986; Michalski, Mozetic, Hong, & Lavrac, 1986). In many cases, numeric attribute domains were transformed to symbolic domains in order to simplify the learning task. This transformation often required the assistance of an expert. In many domains, however, experts do not exist or are sufficiently scarce to make the transformation task difficult. Some systems, such as STAGGER (Schlimmer, 1987), automatically perform this transformation process, but they incur computational costs and do not guarantee success. IBL techniques do not require these transformations.

Instance-based techniques are good methods to use for symbolic value-prediction tasks because they are extremely simple to apply, they allow for the incremental processing of training instances, they are highly tolerant of noise in the predictor attributes (Stanfill, 1987), and they tolerate irrelevant attributes (Kibler & Aha, 1987; Stanfill, 1987). They also are good methods for numeric value-prediction tasks. We can prove that instance-based prediction (IBP) techniques are correct when given noise-free instances (see Section 3) and they are applicable to a large set of value-prediction tasks. In a sense that we will make precise, any continuous function with bounded slope is learnable by these techniques.

The set of continuous functions contains a huge class of naturally occurring functions. Most physical laws are differential equations which have differentiable functions as their solutions. Our estimation of physical processes, and in particular, the movement and equilibrium of bodies, is required in such daily events as balancing, walking, eating, pouring, cooking, driving, throwing, catching, etc. Our ability at sports rests on the ability to predict, with some accuracy, the positions and velocities of moving bodies as well as the ability to coordinate and execute our own movements. Mispredictions, such as expecting another step while climbing stairs or lifting light objects which appear to be heavy, can yield calamitous results.

In Section 2 we illustrate our algorithms and compare IBP techniques to linear regression techniques for prediction. The latter is an alternative prediction method that is most applicable to those functions expressible as a linear combination of their attributes. IBP techniques are more generally applicable to the class of locally linear functions. One advantage of linear regression is that it tolerates noise. In Section 2 we introduce a simple technique (later applied in our experiments) that allows instance-based approaches to tolerate noise when predicting numeric values. (General techniques for tolerating noise in instance-based symbolic classification tasks are described by Aha & Kibler (1988).)

Section 3 summarizes our theoretical justifications for applying these techniques to numeric value-prediction tasks. We provide empirical justification in Section 4. We conclude in Section 5 with a discussion and evaluation of IBP.

## 2 Algorithms and Illustrations

Instance-based learning and prediction algorithms have predominantly been applied to symbolic classification tasks. In these tasks each instance is represented as a set of attribute-value pairs, where the values are either numeric or nominal. The value to be predicted is always nominal. A concept is a subspace of the instance space.

The problem to be solved by symbolic classification algorithms is to determine a function $\mathbf{F}$ that, given an instance, yields the concept of the instance. More precisely, let $C$ be a set of concepts, $I$ be an instance space with $n$ attributes, and $V_i$ be the set of values in attribute domain $i$ $(1 \leq i \leq n)$. Then

$$\mathbf{F}(\langle x_1, \ldots, x_n \rangle) = C_i$$
$$\text{where } \forall i\ (1 \leq i \leq n)\{x_i \in V_i\}, \text{ and}$$
$$C_i \in C.$$

Given a set of instances, learning algorithms typically generate a summary description or predicate for each symbolic concept. These predicates can be applied to a new instance to yield a classification. In contrast, instance-based approaches represent a concept using a set of instances and a similarity metric. A new instance is classified by some form of best-match with existing concepts.

For example, we (Kibler & Aha, 1987) have applied several IBL algorithms to two of Quinlan's thyroid disease databases (Quinlan et al, 1986). The instances were described in terms of 26 attributes. The concepts consisted of the set

{ hypothyroid, sick-euthyroid, negative }.

Each learning algorithm was given a *training set* of instances from which it derived a *concept set*, which is also a set of instances. The concept set was subsequently tested on a *test set* of disjoint instances. The nearest neighbor classification algorithm was used for all the learning algorithms. For each test instance $t$, it guessed that $\mathbf{F}(t) = \mathbf{F}(n)$, where $n$ is $t$'s nearest neighbor in the concept set. We measured each algorithm's performance by recording the percentage of test instances that were classified correctly by the concept set. In effect, a *concept description* (concept set plus similarity metric) described each algorithm's guess of $\mathbf{F}$ with respect to the instances on which it was trained.

In this paper we are concerned with instance-based methods for predicting numeric rather than symbolic values. More specifically, if we let $R$ be a numeric domain, then we can describe $\mathbf{F}$ for predicting numeric domain values as follows:

$$\mathbf{F}(\langle x_1, \ldots, x_n \rangle) = r_i$$
$$\text{where } \forall i\ (1 \leq i \leq n)\{x_i \in V_i\}, \text{ and}$$
$$r_i \in R.$$

---

$\forall t \in$ Training Set:

- $C \leftarrow C \cup \{\text{normalize}(t)\}$

$\forall t \leftarrow \text{normalize}(t'), t' \in$ Test Set:

- $\forall c \in C\{c \neq t\}$ : calculate Similarity$(t, c)$
- Let $Sim \subseteq C$ be the set of $N\%$ most similar instances of $C$ to $t$.
- Let $Sum = \sum_{c \in \text{Sim}} \text{Similarity}(t, c)$
- Then $\mathbf{F}$-estimate$(t) = \sum_{c \in \text{Sim}} \frac{\text{Similarity}(t, c)}{Sum} \times \mathbf{F}(c)$

---

Table 1: The proximity algorithm experiment (C = concept set).

---

$$\text{Similarity}(t, c) = \sum_{i=1}^{n} \text{Simil}(i(t), i(c))$$

where Simil$(x, y) = 1.0 - |x - y|$, and
$i(t)$ yields the attribute value of instance $t$ in dimension $i$

---

Table 2: Similarity of two normalized instances (n dimensions).

Connell and Utgoff (1987) recently applied IBP techniques to the cart-pole balancing problem. Their system predicted state desirability, which was continuous from $-1$ (undesirable) to 1 (most desirable). Saved instances had values of either $-1$ or 1. All other instances' degree of desirability were derived via a weighted-similarity function of the saved instances' desirability values. In our case, saved numeric domain values can have any range and any value within that range.

We chose to use the simplest instance-based learning algorithm, called the *proximity* algorithm, for the experiments in this paper. The proximity algorithm simply saves all training instances in the concept set. The IBP method employed in the experiments is detailed in Table 1. The normalization algorithm maps each attribute value into the continuous range 0–1, ensuring that all attributes are assigned equal classification salience. Assuming that each attribute counts equally is not necessarily correct, but is, at least, fair. The problem of finding the appropriate weights for attributes is another example of the unsolved credit assignment problem. The estimate $\mathbf{F}(t)$ for test instance $t$ is defined in terms of a weighted-similarity function of $t$'s nearest neighbors in the concept set. The similarity of two normalized instances is defined in terms of their Euclidean distance, as detailed in Table 2.

The classification algorithm employed in the experiments is a variant of Shepard's function (Connell & Utgoff, 1987) that, instead of using all concept instances to assist in classifying a new instance, uses only the subset of the neighboring concept instances for numeric value prediction. The underlying assumptions of the linear regression model is that $\mathbf{F}$ is approximately linear. In contrast, using only a few neighboring instances for classification assumes that the concept function is locally linear, a much weaker assumption. Unfortunately, our techniques become more sensitive to noise as fewer instances are used.

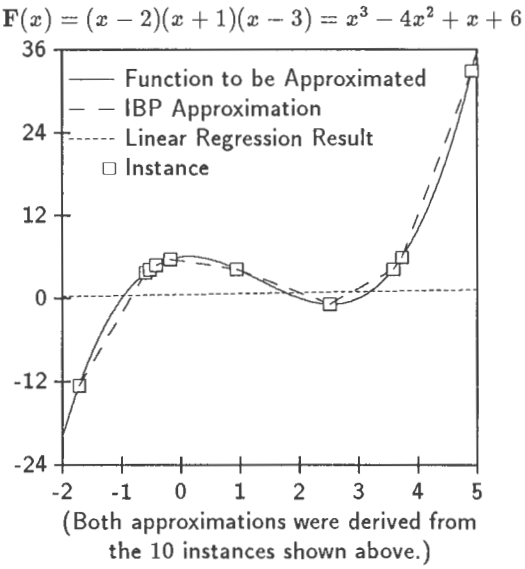



Figure 1: Approximating a typical non-linear function.

If the function is linear and without noise, then our function produces the same effect as interpolation. For example, if the domain is the real numbers, then the predicted value of a test instance is the weighted average of the **F** values of its two closest observed instances. Figure 1 illustrates an application of our techniques where **F** is a polynomial function.[1] Included in Figure 1 are two approximations of **F**, one by IBP using 10 training instances and the other by a linear regression model derived from the same training set. Linear regression techniques, of course, are not meant to be applied to nonlinear functions. This simply demonstrates a case where IBP methods are applicable and linear regression methods are not.

In the general case where there are $n$ attributes one might expect to use the $n+1$ nearest neighbors. Instead we choose to use $N\%$ of the nearest values. This allows us to tolerate some noise and yet does not demand that we assume the function is globally linear. We take an "average" hyperplane defined by these instances to predict the value of the unknown.

# 3 Theoretical Justification

The goal of this section is to demonstrate that instance-based techniques can learn to predict the value of any continuous function with bounded derivative. To do this we first establish that a sufficiently large random sample of values presents a good sampling of instances. Then we demonstrate that, given a good sample of instances and values, a piecewise linear function can be generated which will closely approximate the unknown function.

[1] We use a modified definition of IBP approximation for one-dimensional applications. Instead of making predictions from a set of nearest neighbors, the IBP approximation bases approximations on the two "surrounding" sample instances and uses the nearest neighbor approach for instances that are not surrounded. See Theorem 1 for details.

## 3.1 Coverage Lemma

Here we establish that a large-enough sample gives a good coverage of the domain.

We start by reviewing some basic material from advanced calculus. An *$\epsilon$-ball* about a point $x$ of the real line $R$ is the set of points $\{y : |y - x| < \epsilon\}$. (Note that the size of an $\epsilon$-ball in 1 dimensional space is approximately $2\epsilon$.) The same definition is valid in n-dimensional Euclidean space $R^n$, if we interpret $|x - y|$ as the distance function in $R^n$.

**Definition:** Let $X$ be a subset of an $m$-dimensional space. A subset $S$ of $X$ is an *$\epsilon$-net* for $X$ if, for all $x$ in $X$ there exists an $s$ in $S$ such that $|s - x| < \epsilon$.

We now prove that a sufficiently large random sample from the unit interval will probably be an $\epsilon$-net.

**Lemma 1.** *Let $\epsilon$ and $\delta$ be fixed positive numbers less than 1. A random sample $S$ containing $m > 1/\epsilon \times ln(1/\epsilon\delta)$ instances from $[0,1]$ will form an $\epsilon$-net with confidence greater than $1 - \delta$.*

**Proof:** We prove this lemma by partitioning the unit interval into $k$ equal-sized sub-intervals, each with size less than $\epsilon$. We also ensure that, with high probability, at least one of the $m$ sample instances lies in each sub-interval.

Let $k > 1/\epsilon$. The probability that arbitrary instance $i \in [0,1]$ will lie in some selected sub-interval is $1 - 1/k$. The probability that none of the $m$ sample instances will lie in a selected sub-interval is $(1 - 1/k)^m$. The probability that any sub-interval is excluded by all $m$ sample instances is $k \times (1 - 1/k)^m$. Since $(1 - 1/k)^m < e^{-m/k}$, then

$$k \times (1 - 1/k)^m < k \times e^{-m/k}.$$

We ensure that this probability is small by forcing it to be less than $\delta$. We solve for $m$ as follows:

$$\begin{aligned} k \times e^{-m/k} &< \delta \\ e^{-m/k} &< \delta/k \\ -m/k &< ln(\delta/k) \\ m &> -k \times ln(\delta/k) \\ m &> k \times ln(k/\delta) \end{aligned}$$

Consequently, with probability greater than $1-\delta$, each sub-interval contains some sample instance of $S$. Since each instance of $[0,1]$ is in some sub-interval, then, with this same probability, an arbitrary instance of $I$ is within $\epsilon$ of some instance of $S$. ■

The proof of Lemma 1 generalizes to any bounded region in $R^n$ (i.e. it guarantees that by picking enough random samples, we can ensure that we will probably get a good coverage of any nice domain).

## 3.2 Instance-based Prediction Theorems

Here we prove that, given a good sample of instances, IBP can generate a piecewise linear function that is a good approximation to an unknown continuous function.

**Definition:** A function $f$ is an *$\epsilon$-approximation* of a function $g$ if they have the same domain, and for all instances $x$ in their common domain $|f(x) - g(x)| < \epsilon$.

The following definition is motivated by Valiant's (1984) work on learnability theory.

**Definition:** A function $f$ is *learnable by IBP techniques* when, for $1 > \delta, \epsilon > 0$, the function $\tilde{f}$ generated by IBP is an $\epsilon$-approximation of $f$ with probability greater than $1-\delta$.

Note that our definition does not specify how long learning might take, only that it usually converges to approximately the right answer.

**Definition:** The slope of a function $f$ is *bounded on $X$ by $B$* if

$$\forall x, x' \in X \ (x \neq x') \left|\frac{f(x) - f(x')}{x - x'}\right| \leq B.$$

The following theorem demonstrates that continuous, real-valued functions with bounded slope in the unit interval [0,1] are learnable. Extensions to multi-valued functions of multiple arguments is standard, requiring only a working knowledge of advanced calculus.

**Theorem 1.** *Let $f$ be a continuous, real-valued function on $[0,1]$ with slope bounded by $B$. Then $f$ is learnable by IBP techniques.*

**Proof:** Let $f$ be a continuous function on $[0,1]$. Let $\delta$ and $\epsilon$ be arbitrary positive numbers less than 1. We will guarantee that $f$ does not vary much on a small interval by using the bound on the slope of $f$. In particular, we apply Lemma 1 with $\epsilon' = \epsilon/2B$. (We assume, without loss of generality, that $B \geq 1$.) The lemma guarantees that, if $m$ is large enough, then we will have an $\epsilon/2B$-net for $[0,1]$ with confidence greater than $1-\delta$. More specifically, we let $S$ be a random sample of $[0,1]$ with size $m > 2B/\epsilon \times ln(2B/\epsilon\delta)$.

We define the approximation function $\tilde{f}(x)$ of $f(x)$ as follows. For each of the $m$ sample points, let $\tilde{f}$ be defined as the sample value. On non-sample points, let $\tilde{f}$ be defined as the linear interpolation of its surrounding neighbors. If a non-sample point $p$ is not surrounded, then let $\tilde{f}(p) = f(x)$, where $x$ is the closest neighbor in the sample.

Now we must show that $\tilde{f}$ is an $\epsilon$-approximation of $f$. Let $x$ be an arbitrary point in $[0,1]$. We shall consider first the case where $x$ is surrounded by sample instances $x'$ and $x''$ in $S$, where $x'$ is the nearest neighbor of $x$ in $S$. Note that

$$f(x) = f(x') + s(x', x) \times (x - x')$$
$$\tilde{f}(x) = \tilde{f}(x') + \tilde{s}(x', x'') \times (x - x')$$

where $s(x', x)$ is the slope of $f$ between $x'$ and $x$ and $\tilde{s}(x', x'')$ is the slope of $\tilde{f}$ between $x'$ and $x''$.

Since $B$ is the upper bound on the slope between any two points for $f(x)$, it is also an upper bound on the slope between any two points for $\tilde{f}(x)$. Therefore,

$$|f(x) - f(x')| = |s(x', x) \times (x - x')| < B \times \epsilon/2B = \epsilon/2$$

and

$$|\tilde{f}(x) - \tilde{f}(x')| = |\tilde{s}(x', x'') \times (x - x')| < B \times \epsilon/2B = \epsilon/2.$$

Since $\tilde{f}(x') = f(x')$, we have, by triangular inequality,

$$|f(x) - \tilde{f}(x)| < \epsilon/2 + 0 + \epsilon/2 = \epsilon.$$

The second case, where $x$ is not surrounded by sample instances in $S$, can be handled similarly. This proves that the piecewise linear approximation $\tilde{f}$ is an $\epsilon$-approximation of $f$. ■

Using an extension of Lemma 1 and a proof similar to that used for Theorem 1, we can prove Theorem 2.

**Theorem 2.** *If $f$ is a continuous function on a closed and bounded $n$-dimensional space with derivative bounded by $B$, then $f$ is learnable by IBP techniques. In particular, for given positive values of $\epsilon$ and $\delta$, $m > 2B/\epsilon^n \times ln(2B/\epsilon^n\delta)$ samples will suffice.*

Note that any piecewise linear curve and any function with a continuous derivative has a bounded slope.

This result indicates that, given an open domain, one needs to require some constraints on the "wildness" of the function in order to ensure that the time to learn is polynomially bounded. In particular, we compensate for looser constraints on the domain of the function by tightening the constraints on how fast the function can change its value over a small interval. That is, we require that the derivative of the function exist and be uniformly bounded.

The requirement for the bound on the derivative is illustrated by the function $sin(1/x)$ on the open interval (0,1]. Note that as $x$ approaches 0, the derivative (slope) of the function is unbounded. As is easily seen, for any $\epsilon$ between 0 and 1, there is no piecewise linear $\epsilon$-approximation of this function.

While these results establish the appropriateness of IBP for noise-free functions, real world data requires attention to noise. A standard method for tolerating noise is to use some form of averaging. In particular, we believe that the following change to the algorithm yields a method which works for noise in the function value, but the proof eludes us.

Instead of constructing a piecewise linear approximation based on a single $\epsilon$-net, consider forming $m$ $\epsilon$-nets, each net yielding a piecewise linear approximation $\tilde{f}_i$. We define $\tilde{f}$ to be the (pointwise) average of these $\tilde{f}_i$. Clearly $\tilde{f}$ is still piecewise linear. We believe that, the resulting $\tilde{f}$ will be a good approximation. We demonstrate the appropriateness of this method with an empirical study.

## 4 Empirical Justification

In this section we describe our experiments and show how the results support our conjecture. In particular, we note that instance-based learning techniques achieve the same accuracy as linear regression methods. We note that if the underlying function was exactly linear then both linear regression and IBP would yield perfect predictions. The value of IBP is that it requires that the function be only locally linear while linear regression requires that the function be globally linear. Also, IBP requires no *ad hoc* adjustments to the underlying method, which are often made when applying linear regression models.

| Acronym | Attribute Name | Unit |
|---|---|---|
| MCYT | Machine Cycle Time | Nanoseconds |
| MMIN | Minimum Main Memory | Kilobytes |
| MMAX | Maximum Main Memory | Kilobytes |
| CACH | Cache Memory Size | Kilobytes |
| CHMIN | Min Number of I/O Channels | Channels |
| CHMAX | Max Number of I/O Channels | Channels |

Table 3: Predictive attributes of the central processing units database.

| Vendor/Model | MCYT<br>CHMIN | MMIN<br>CHMAX | MMAX | CACH<br>PRP |
|---|---|---|---|---|
| IBM 4341-12 | 185<br>1 | 2000<br>6 | 16000 | 16<br>76 |
| DEC-Vax-11/750 | 320<br>1 | 512<br>5 | 8000 | 4<br>40 |
| Sperry 1100/94 | 30<br>12 | 8000<br>176 | 64000 | 128<br>1150 |

Table 4: Example instances of the CPU characteristics database.

### 4.1 Experimental Data

We chose to experiment with the database published by Ein-Dor and Feldmesser (1987) in the April 1987 issue of the Communications of the ACM. The authors described a stepwise multivariate linear regression technique for predicting the published relative performance (PRP) of central processing units. Each of the 209 cpu data instances is represented with six predictive attributes (described in Table 3). The other attributes included the vendor, the model name, and its published relative performance. Three example instances are shown in Table 4.

The purpose of their article was to describe a predictive model for computer systems that was simpler than queueing networks. Our purpose here is to demonstrate that IBP techniques can be used to describe models of equal simplicity and accuracy for approximating the published relative performance function.

We chose to experiment with this data set for several reasons:

1. The authors presented a case study that allowed us to contrast a linear regression model with an IBP model on the same data set.
2. We wanted to show that IBP models support predictive behavior for natural databases (as opposed to being restricted to carefully crafted artificial databases).
3. We were interested in investigating the predictive quality of IBP techniques when both the input and output attributes were numeric.

| Range of Relative Performance | Number of Instances | % Average Deviation | | |
|---|---|---|---|---|
| | | Linear Regression | IBP | |
| | | | Raw | Tuned |
| 0–20 | 31 | 72.17 | 81.38 | 55.62 |
| 21–100 | 121 | 28.64 | 27.88 | 29.64 |
| 101–200 | 27 | 28.57 | 27.91 | 31.60 |
| 201–300 | 13 | 23.93 | 19.16 | 22.21 |
| 301–400 | 7 | 21.49 | 22.12 | 27.14 |
| 401–500 | 4 | 18.72 | 16.80 | 16.20 |
| 501–600 | 2 | 17.35 | 11.86 | 30.49 |
| 600+ | 4 | 10.34 | 43.69 | 33.35 |
| All | 209 | 33.91 | 35.02 | 33.02 |

Table 5: Average deviation of relative performance predictions.

### 4.2 Experiment and Results

We carried out two experiments with the cpu performance data. In each case, the function **F** to be predicted was PRP. The results of both our own experiments and the original linear regression experiment are displayed in Table 5. All results are in terms of the average deviation of the predicted and actual cpu performance values.[2]

Ein-Dor and Feldmesser (1987) calculated the linear regression equation for cpu performance and then used it to predict each instance's relative cpu performance. Since regression minimizes the square of the absolute error, it will appear to be more accurate for large values when evaluated in terms of relative error.

In our first experiment (see column Raw in Table 5), each instance was described in terms of the 6 given attributes. The training and test sets were identical; they contained all 209 instances. We set the value for $N$ to 3%, meaning that the IBP prediction of a test instance's relative cpu performance was based on its 3% most similar instances in the concept set (excluding the test instance itself). The proximity algorithm fares relatively well in each range except the first and last. The values of the first (smallest-valued) range were sensitive to small absolute errors and thus produced the greatest relative error. The last range contains the highest values, which were few in number and highly scattered. Actually it would be easy to assign some measure of confidence with IBP which would suggest that the predicted value was tenuous, at best, for the high range.

The overall average deviation of predicted and actual values for the first experiment is surprisingly similar to that of the linear regression experiment, which was only slightly better.

What differed between the experiments is of greater importance. The regression experiment required a great deal of application-dependent knowledge while the first IBP experiment employed none. Ein-Dor and Feldmesser analyzed the data and subject domain and reviewed the linear correlations of the PRP values with the 6 independent attributes. They concluded that transformations of the independent

[2]The published value for overall average deviation of the linear regression model is 34.10 (Ein-Dor & Feldmesser, 1987), which disagrees with our calculation of 33.91. Otherwise, our calculations are in agreement.

| Derived Attribute | Attribute Transformation |
|---|---|
| Average Memory Size | $(\text{MMIN} + \text{MMAX})/2 \times 10^{-3}$ |
| Cache Memory Size | $\text{CACH} \times 10^{-1}$ (or 0 if none) |
| Channel Capacity | $(\text{CHMIN} + \text{CHMAX})/2 + 1$ $\times(1/\text{MYCT}) \times 10$ |

Table 6: Transformations of the 6 independent attributes.

and dependent attributes were needed to enhance the regression model's predictiveness. First, they chose to employ a square-root transformation of the dependent attribute. They then chose to employ 3 "tuned" attributes to be used as independent attributes for prediction of the square-root of the published relative performance values. The tuned attributes are average memory size, $1/10^{th}$ of the cache size, and channel capacity. The transformations are listed in Table 6.

The problem they faced is a typical one when using linear regression. What do you do when the dependent variable is not a linear combination of the attributes? In their case, by appealing to domain knowledge, specifically Grosch's law, they reasoned that the *square root* of the performance would be a linear combination of measured attributes. Finding this particular transformation requires a model of the domain. IBP does as well without introducing domain expertise.

In our second experiment (see column Tuned in Table 5), we used the same tuned input attributes for the IBP method as thosed used by the linear regression technique. This was done by calculating the values for the three tuned attributes and representing the instances in terms of them. The effect of tuning attributes is to give a better weight to each individual attribute. The average deviation results for this experiment also appear in Table 5. In this case, we find that the IBP predictions are slightly better than those given by the linear regression model, again measured in terms of average deviation.

## 5 Discussion

Continuous functions are an extremely large class of functions. In particular, they can represent the behavior of physical systems. Our ability to interact successfully with the world depends, in part, on our ability to predict the continuously changing environment. Given a sufficiently large sample size, we have shown that IBP is guaranteed to yield a good approximation for continuous functions. In the presence of noise, we demonstrated that IBP yields results equivalent to that of linear regression, but without having to make ad hoc assumptions. Moreover, the technique is incremental. In short, IBP is a simple, general, efficient technique which yields high quality predictions. As we have illustrated, the quality of predictions by IBP can be improved if one has appropriately weighted features. Finding computationally efficient means for calculating these weights, however, requires further research.

One of the criticisms of IBL techniques is that they have high storage requirements. Recent results, however, have suggested that simple learning algorithms can be applied that greatly ease storage requirements. Bradshaw's (1987) disjunctive-spanning algorithm uses an averaging technique in an effort to reduce storage requirements while maintaining high classification accuracies. Kurtzberg (1987) described an algorithm that, when trained on 288 instances (four copies of 72 handwritten symbols), saved only 121 instances and still achieved a 99.0% classification accuracy on a same-sized, disjoint set of test instances. Kibler and Aha (1987) showed that the same algorithm, when applied to Quinlan et al's (1986) hypothyroid disease data, saved an average of only 10 of the 220 training instances and still achieved a 97% accuracy on a disjoint set of 500 test instances. We (Kibler & Aha, 1988) have shown that the upper bound on the number of instances required for approximating concepts is proportional to the lengths of the boundaries separating concepts. IBL techniques do not appear to have large storage requirements for symbolic classification tasks. We anticipate that similar storage-saving techniques can be used to assist numeric value-prediction tasks.

A more severe criticism of instance-based representations is that they do not yield concise encapsulations of the data that can easily be understood and reasoned about by humans or machines. For example, BACON (Langley, 1981) discovers the ideal gas law ($pV/nT = c$, where $c$ is a constant) given numerically-valued data. IBP techniques cannot represent, let alone find, this relationship. BACON heuristically searches through a space of functional expressions to find this formula. Similarly, linear regression methods search through the space of linear equations to find an easily understood relationship between the independent and dependent attributes. IBP does not yield comprehensible summaries of the data. The compensating value of IBP is that it does not require that the data satisfy some predefined model.

### Acknowledgements

We would like to thank Marc Albert for suggesting a revision of Theorem 1.

### References

Aha, D. W., & Kibler, D. (1988). *Detecting and removing noisy instances from concept descriptions.* Manuscript submitted for publication.

Bradshaw, G. (1987). Learning about speech sounds: The NEXUS project. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 1–11). Irvine, CA: Morgan Kaufmann.

Connell, M. E., & Utgoff, P. E. (1987). Learning to control a dynamic physical system. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 456–460). Seattle, WA: Morgan Kaufmann.

Ein-Dor, P., & Feldmesser, J. (1987). Attributes of the performance of central processing units: A relative performance prediction model. *Communications of the Association for Computing Machinery, 30*, 308–317.

Kibler, D., & Aha, D. W. (1987). Learning representative exemplars of concepts: An initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 24–30). Irvine, CA: Morgan Kaufmann.

Kibler, D., & Aha, D. W. (1988). *Comparing instance-averaging with instance-saving learning algorithms* (Technical Report 88-06). Irvine, CA: University of California, Irvine, Department of Information and Computer Science.

Kurtzberg, J. M. (1987). Feature analysis for symbol recognition by elastic matching. *I.B.M. Journal of Research and Development, 31*, 91–95.

Langley, P. (1981). Data-driven discovery of physical laws. *Cognitive Science, 5*, 31–54.

Michalski, R.S., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 1041–1045). Philadelphia, PA: Morgan Kaufmann.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence, 18*, 203–226.

Quinlan, J. R., Compton, P. J., Horn, K. A., & Lazurus, L. (1986). Inductive knowledge acquisition: A case study. In *Proceedings of the Second Australian Conference on Applications of Expert Systems*. Sydney, Australia.

Schlimmer, J. C. (1987). Incremental adjustment of representations for learning. In *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 79–90). Irvine, CA: Morgan Kaufmann.

Stanfill, C., & Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM, 29*, 1213–1228.

Stanfill, C. (1987). Memory-based reasoning applied to English pronunciation. In *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 577–581). Seattle, WA: Morgan Kaufmann.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the Association for Computing Machinery, 27*, 1134–1142.

Vere, S. A. (1980). Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence, 14*, 139–164.

# AXIOMATIZATIONS IN THE METATHEORY OF NONMONOTONIC INFERENCE SYSTEMS

Philippe Besnard

IRISA
Campus de Beaulieu
35042 Rennes Cédex
FRANCE

**Abstract**

In this preliminary report, nonmonotonic inference systems are examined in an abstract setting independent from the inferential apparatus they develop, that is, they are regarded as relations between sets of formulas, premises on the one hand and conclusions on the other hand. Specifically, various axiomatizations for properties of nonmonotonic inference relations are given, which yield nice characterizations for some existing nonmonotonic inference systems. The approach proposed also displays some connection, in the form of incompatibility, between nonmonotonicity and the converse to the property expressed by the deduction theorem.

## 1 Introduction

Formalizing properties for inference systems has been achieved by Tarski [1956] and Gentzen [1969] who defined the notion of consequence relation. Precisely, a language being fixed so as to define the class of all formulas, in order for any relation $\vdash$ between sets of formulas and individual formulas, called an inference relation, to be a consequence relation, Gentzen set up the requirement of

. a reflexive property:

if $A \in \mathbf{T}$ then $\mathbf{T} \vdash A$

. a cut rule:

if $\mathbf{T} \vdash A$ and $\mathbf{S},A \vdash B$ then $\mathbf{T},\mathbf{S} \vdash B$

for all formulas $A$ and $B$ and sets of formulas $\mathbf{S}$ and $\mathbf{T}$, also called theories.

The notation $\mathbf{S},A$ for $\mathbf{S} \cup \{A\}$ and $\mathbf{T},\mathbf{S}$ for $\mathbf{T} \cup \mathbf{S}$ is standard in this context.

The point to be made now is that if $\mathbf{T} \vdash A$ and $\mathbf{S},A \vdash A$ then $\mathbf{T},\mathbf{S} \vdash A$ holding by virtue of the cut rule and $\mathbf{S},A \vdash A$ being postulated by the reflexive property, these two conditions cause any consequence relation $\vdash$ to enjoy

. the monotonicity property:

if $\mathbf{T} \vdash A$ then $\mathbf{T},\mathbf{S} \vdash A$

Formalizing properties for nonmonotonic inference systems has been undertaken by Gabbay [1985] and Makinson [1987] as such systems are being paid increasing attention [Bobrow 1980]. In the framework developed by Gabbay for inference relations $\mid\sim$ which are not monotonic, Gentzen's cut rule as formulated above is weakened to

. the cumulation property:

if $\mathbf{T} \mid\sim A$ then $\mathbf{T} \mid\sim B$ iff $\mathbf{T},A \mid\sim B$

which gathers

. cautious monotonicity:

if $\mathbf{T} \mid\sim A$ and $\mathbf{T} \mid\sim B$ then $\mathbf{T},A \mid\sim B$

and

. transitivity:

if $\mathbf{T} \mid\sim A$ and $\mathbf{T},A \mid\sim B$ then $\mathbf{T} \mid\sim B$

the latter being just an instance of the cut rule.

Now, instead of weakening monotonicity, dropping it altogether is worth investigating in case of nonmonotonic inference systems constructed out of consequence relations. Indeed, the notions attached to some variants of such a more radical revision of Gentzen's conditions for inference relations appear to qualify more existing nonmonotonic systems than the revision advocated by Gabbay (in particular, it seems desirable to qualify systems in [Gallaire & Minker 1978], that is Reiter's closed world assumption and Clark's predicate completion as well as logics in [Bobrow 1980], that is McCarthy's circumscription, Reiter's default logic and the nonmonotonic logic due to McDermott and Doyle). As a justification for all this, the present paper reports on the fact that, even at a preliminary stage of examination (conducted in a Gentzen-style framework but not in a Tarski-style framework because there is no deep difference between both [Besnard 1988]), the approach just proposed can be shown to yield significant results.

## 2 The framework

So, as a major departure from Gabbay's and Makinson's works, the nonmonotonic inference relations $\mid\sim$ under consideration here are supposed to be based on an underlying consequence relation $\vdash$ as characterized by Gentzen. In this context, it makes sense to formulate the following

. weak cumulation property:

if $\mathbf{T} \vdash A$ then $\mathbf{T} \mid\sim B$ iff $\mathbf{T},A \mid\sim B$

which basically says that theories equivalent in the sense of the consequence relation $\vdash$ are also equivalent in the sense of the inference relation $\mid\sim$. Not all existing nonmonotonic inference systems meet such a requirement. For instance, consider $\mid\sim$ standing for predicate completion and $\vdash$ standing for first order logic. Even though $\{P\} \vdash \neg P \Rightarrow P$, it is not the case that $\{P\},\neg P \Rightarrow P \mid\sim A$ iff $\{P\} \mid\sim A$.

Since the cumulation property consists of cautious monotonicity and transitivity, it is clear that four basic relaxation conditions arise for inspection

. if $\mathbf{T} \vdash A$ and $\mathbf{T} \mid\sim B$ then $\mathbf{T},A \mid\sim B$

. if $\mathbf{T} \vdash A$ and $\mathbf{T},A \mid\sim B$ then $\mathbf{T} \mid\sim B$

. if $\mathbf{T} \mid\sim A$ and $\mathbf{T} \vdash B$ then $\mathbf{T},A \mid\sim B$

. if $\mathbf{T} \mid\sim A$ and $\mathbf{T},A \vdash B$ then $\mathbf{T} \mid\sim B$

The first two form the weak version of the cumulation property. The third one combined with reflexivity is equivalent (the empty theory being not taken into account) to conservativeness stated below. Of special interest is the fourth one called

. compound transitivity:

if $\mathbf{T} \mid\sim A$ and $\mathbf{T},A \vdash B$ then $\mathbf{T} \mid\sim B$

An example for a nonmonotonic inference relation $|\sim$ satisfying only compound transitivity with respect to the relation $\vdash$ representing first order logic is given by default logic. It appears that one of the most striking pecularities of default logic (one symptom of which is that the inference relation $|\sim$ standing for default logic is not defined for all theories T) is shared by all inference relations $|\sim$ of the kind discussed.

**Property 1**: If an inference relation $|\sim$ satisfies compound transitivity with respect to a consequence relation $\vdash$ then for any theory T the restriction of $|\sim$ to T either is empty or includes the restriction of $\vdash$ to T

**Proof**: Given a theory T, assume that the first alternative possibility is not the case. Then $T |\sim A$ for some formula $A$. Now, for all formula $B$ such that $T \vdash B$ it follows that $T, A \vdash B$ by monotonicity. Thus, $T |\sim B$ is obtained by applying compound transitivity. □

Indeed, an inference relation whose restriction to some theory is empty may still conform to compound transitivity. The reason for this is that compound transitivity is a conditional property, henceforth it is satisfied in case its proviso is not.

The loose connection between a possibly nonmonotonic inference relation $|\sim$ and its underlying consequence relation $\vdash$ may be reinforced in a very natural way so that $|\sim$ encompasses $\vdash$ as described by the following

. conservativeness:

if $T \vdash A$ then $T |\sim A$

Examples of nonmonotonic inference relations that obey conservativeness and compound transitivity with respect to first order logic are provided by predicate completion, closed world assumption, circumscription, the nonmonotonic logic devised by McDermott and Doyle and also the normal fragment of default logic.

Following Makinson [1987], an inference relation $|\sim$ may be required to preserve the consistency induced by its underlying consequence relation $\vdash$, that is $|\sim$ may be required to satisfy

. relative consistency:

if $T |\sim A$ for all formulas $A$ then $T \vdash A$ for all formulas $A$

Examples are now the Horn fragment of both closed world assumption and predicate completion, the well-founded fragment of circumscription and the normal fragment of default logic.

Except for Property 1, all results given throughout the text hold for any (possibly nonmonotonic) inference relation $|\sim$ which satisfies conservativeness, but not necessarily relative consistency, with respect to an underlying consequence relation $\vdash$. Also, for taking into account any such inference relation $|\sim$, reflexivity need not be postulated since it follows from conservativeness with respect to consequence relation $\vdash$.

## 3 T-regular and CT-regular inference relations

The conditions exposed up to now form the basis for the study presented in the sequel.

**Definition**: Given an inference relation $|\sim$ with underlying consequence relation $\vdash$ such that conservativeness is satisfied, if compound transitivity is verified then $|\sim$ is said to be *CT-regular*, whereas if transitivity is verified then $|\sim$ is said to be *T-regular*.

To reformulate the above classification with the terminology just introduced, the Horn fragment of predicate completion, the well-founded fragment of circumscription and the normal fragment of default logic are all CT-regular.

Two observations are enough to characterize whether an inference relation is T – regular in case it is CT – regular and conversely.

**Property 2**: If an inference relation is T – regular then it is also CT – regular

**Proof**: If $\mathbf{T} \mid\sim A$ and $\mathbf{T}, A \mid\!- B$ then, first, $\mathbf{T} \mid\sim A$ and second, $\mathbf{T}, A \mid\sim B$ because $\mid\sim$ satisfies conservativeness with respect to $\mid\!-$. Since $\mid\sim$ is T – regular, it follows that $\mathbf{T} \mid\sim B$. □

**Property 3**: Not every CT – regular inference relation is T – regular

**Proof**: Taking $\mid\!-$ to refer to classical sentential logic, $\mid\sim$ is constructed as follows. Fix two propositional letters $P$ and $Q$. For every theory $\mathbf{T}$, in case $\mathbf{T} \cup \{\neg Q \wedge P\}$ is consistent for sentential logic then $\mathbf{T} \mid\sim A$ whenever $\mathbf{T} \cup \{\neg Q \wedge P\} \mid\!- A$, in case $\mathbf{T} \cup \{P \Rightarrow Q\}$ is consistent for sentential logic then $\mathbf{T} \mid\sim A$ whenever $\mathbf{T} \cup \{P \Rightarrow Q\} \mid\!- A$, otherwise $\mathbf{T} \mid\sim A$ whenever $\mathbf{T} \mid\!- A$. If $\mathbf{T} \mid\sim A$ is taken to hold only as just indicated then the resulting definition of $\mid\sim$ clearly shows that $\mid\sim$ satisfies conservativeness with respect to $\mid\!-$. To prove compound transitivity, assume $\mathbf{T} \mid\sim A$ and $\mathbf{T} \cup \{A\} \mid\!- B$. From this, in case $\mathbf{T} \cup \{\neg Q \wedge P\}$ or $\mathbf{T} \cup \{P \Rightarrow Q\}$ is consistent for sentential logic, $\mathbf{T} \mid\sim B$ follows from Gentzen's cut rule for sentential logic. Otherwise, $\mathbf{T} \mid\sim B$ is obtained as well, in quite a straightforward manner. So, $\mid\sim$ is CT – regular. But it is not T – regular because if $\mathbf{T}$ is empty then $\mathbf{T} \mid\sim P$ and $\mathbf{T}, P \mid\sim Q$ but $\mathbf{T} \mid\!\not\sim Q$. □

Incidentally, the proof shows that the normal fragment of default logic is not T – regular, it is only CT – regular. However, the well – founded fragment of circumscription is T – regular.

## 4 MP – regular and CMP – regular inference relations

The cut rule may be altered in such a way that, as a result, implication is given prominence over any other connective [Tarski 1956]. For instance, it may be required that an inference relation $\mid\sim$ satisfy

. modus ponens:

if $\mathbf{T} \mid\sim A$ and $\mathbf{T} \mid\sim A \Rightarrow B$ then $\mathbf{T} \mid\sim B$

Similarly to transitivity, modus ponens leaves room for variation and an inference relation $\mid\sim$ with underlying consequence relation $\mid\!-$ may be checked for satisfying

. compound modus ponens:

if $\mathbf{T} \mid\sim A$ and $\mathbf{T} \mid\!- A \Rightarrow B$ then $\mathbf{T} \mid\sim B$

**Definition**: Given an inference relation $\mid\sim$ with underlying consequence relation $\mid\!-$ such that conservativeness is satisfied, if compound modus ponens is verified then $\mid\sim$ is said to be *CMP – regular*, whereas if modus ponens is verified then $\mid\sim$ is said to be *MP – regular*.

**Property 4**: If an inference relation is MP – regular then it is also CMP – regular

**Proof**: If $\mathbf{T} \mid\sim A$ and $\mathbf{T} \mid\!- A \Rightarrow B$ then, first, $\mathbf{T} \mid\sim A$ and second, $\mathbf{T} \mid\sim A \Rightarrow B$ because $\mid\sim$ satisfies conservativeness with respect to $\mid\!-$. Since $\mid\sim$ is MP – regular, it follows that $\mathbf{T} \mid\sim B$. □

**Property 5**: Not every CMP – regular inference relation is MP – regular

**Proof**: An appropriate $\mid\sim$ with underlying $\mid\!-$ given by classical sentential logic is constructed as follows. Fix a propositional letter $P$. For every theory $\mathbf{T}$, in case $\mathbf{T} \mid\!\not\sim \neg P$ and $\mathbf{T} \mid\!\not\sim P$ then $\mathbf{T} \mid\sim A$ iff $\mathbf{T}, P \mid\!- A$ or $\mathbf{T}, \neg P \mid\!- A$, otherwise $\mathbf{T} \mid\sim A$ iff $\mathbf{T} \mid\!- A$. This being taken

as a definition for $|\sim$, it is routine to verify that $|\sim$ satisfies conservativeness with respect to $|\!-$. Now, assume $\mathbf{T} |\sim A$ and $\mathbf{T} |\!- A \Rightarrow B$. In case $\mathbf{T} |\!\not\sim \neg P$ and $\mathbf{T} |\!\not\sim P$ then either $\mathbf{T},P |\!- A$ or $\mathbf{T},\neg P |\!- A$. Since $\mathbf{T},P |\!- A \Rightarrow B$ and $\mathbf{T},\neg P |\!- A \Rightarrow B$, then either $\mathbf{T},P |\!- B$ or $\mathbf{T},\neg P |\!- B$. Hence $\mathbf{T} |\sim B$. Otherwise, from $\mathbf{T} |\!- A$ it follows that $\mathbf{T} |\sim A$. Thus $\mathbf{T} |\!- B$ because $\mathbf{T} |\!- A \Rightarrow B$. So, $\mathbf{T} |\sim B$ in this case too. That is, $|\sim$ is CMP – regular. But $|\sim$ is not MP – regular because if $\mathbf{T}$ is empty then $\mathbf{T} |\sim P$ and $\mathbf{T} |\sim P \Rightarrow \neg P \wedge P$ but $\mathbf{T} |\!\not\sim \neg P \wedge P$. □

The normal fragment of default logic is not MP – regular, it is only CMP – regular. In contrast, the well – founded fragment of circumscription is MP – regular.

It remains to elucidate how transitivity, in its mere as well as compound version, and modus ponens, also in its mere as well as compound version, articulate in the framework at hand. This is the subject matter to be dealt with next.

## 5 Comparing types of regular inference relations

Two fundamental properties that consequence relations dealing with implication are urged to satisfy are

. the deduction principle:

if $\mathbf{T},A |\!- B$ then $\mathbf{T} |\!- A \Rightarrow B$

as well as

. the rule of detachment:

if $\mathbf{T} |\!- A \Rightarrow B$ then $\mathbf{T},A |\!- B$

For an inference relation $|\sim$ with an underlying consequence relation $|\!-$, being CMP – regular or being CT – regular are closely related in view of $|\!-$ obeying the rule of detachment and the deduction principle.

**Property 6**: An inference relation with an underlying consequence relation satisfying the rule of detachment is CMP – regular if it is CT – regular

**Proof**: Assume $\mathbf{T} |\sim A$ and $\mathbf{T} |\!- A \Rightarrow B$. From the latter, $\mathbf{T},A |\!- B$ is obtained by the rule of detachment for $|\!-$. Now, if $\mathbf{T} |\sim A$ and $\mathbf{T},A |\!- B$ then $\mathbf{T} |\sim B$ because $|\sim$ is CT – regular. So, $\mathbf{T} |\sim B$ and $|\sim$ is CMP – regular. □

**Property 7**: An inference relation with an underlying consequence relation satisfying the deduction principle is CT – regular if it is CMP – regular

**Proof**: Assume $\mathbf{T} |\sim A$ and $\mathbf{T},A |\!- B$. From the latter, $\mathbf{T} |\!- A \Rightarrow B$ is obtained by the deduction principle for $|\!-$. Now, if $\mathbf{T} |\sim A$ and $\mathbf{T} |\!- A \Rightarrow B$ then $\mathbf{T} |\sim B$ because $|\sim$ is CMP – regular. So, $\mathbf{T} |\sim B$ and $|\sim$ is CT – regular. □

So, for an inference relation coping with implication, being CMP – regular or being CT – regular are basically equivalent, but being MP – regular or being T – regular are not so strongly connected.

**Property 8**: A MP – regular inference relation need not be T – regular

**Proof**: From classical sentential logic, a MP – regular $|\sim$ which is not T – regular is defined next. Fix two propositional letters $P$ and $Q$. In case $\mathbf{T} |\!- \neg P$ then $\mathbf{T} |\sim A$ iff $\mathbf{T} |\!- A$, in case $\mathbf{T} |\!- P$ and $\mathbf{T} |\!\not\sim \neg Q$ then $\mathbf{T} |\sim A$ iff $\mathbf{T},Q |\!- A$, otherwise $\mathbf{T} |\sim A$ iff $\mathbf{T},P |\!- A$. Now, it is not difficult to prove by case analysis that $|\sim$ is MP – regular. But $|\sim$ is not T – regular because if $\mathbf{T}$ is empty then $\mathbf{T} |\sim P$ and $\mathbf{T},P |\sim Q$ but $\mathbf{T} |\!\not\sim Q$. □

**Property 9**: A T – regular inference relation need not be MP – regular

Proof: It suffices to construct a relation $|\sim$ from classical sentential logic where $\mathbf{T} |\sim A$ for all formulas $A$ whenever $\mathbf{T}$ is inconsistent for sentential logic otherwise $\mathbf{T} |\sim A$ iff $A$ is in some consistent complete extension of $\mathbf{T}$ in the sense of sentential logic. From this definition, it is obvious that $|\sim$ is T-regular. However it is not MP-regular because if $P$ is a propositional letter then in case $\mathbf{T}$ is empty, $\mathbf{T} |\sim P$ and $\mathbf{T} |\sim P \Rightarrow P \wedge \neg P$ but $\mathbf{T} |\not\sim P \wedge \neg P$. □

In fact, for an inference relation coping with implication, being MP-regular or being T-regular are the same under the deduction principle and the rule of detachment.

**Property 10**: If a MP-regular inference relation satisfies the deduction principle then it is also T-regular

Proof: Assume $\mathbf{T} |\sim A$ and $\mathbf{T},A |\sim B$. Since $|\sim$ satisfies the deduction principle, $\mathbf{T} |\sim A \Rightarrow B$. Since $|\sim$ is MP-regular, if $\mathbf{T} |\sim A$ and $\mathbf{T} |\sim A \Rightarrow B$ then $\mathbf{T} |\sim B$. So $|\sim$ is T-regular. □

**Property 11**: If a T-regular inference relation satisfies the rule of detachment then it is also MP-regular

Proof: Assume $\mathbf{T} |\sim A$ and $\mathbf{T} |\sim A \Rightarrow B$. Since $|\sim$ satisfies the rule of detachment, $\mathbf{T},A |\sim B$. Since $|\sim$ is T-regular, if $\mathbf{T} |\sim A$ and $\mathbf{T},A |\sim B$ then $\mathbf{T} |\sim B$. So $|\sim$ is MP-regular. □

Property 11 is much less interesting than Property 10 because nonmonotonic relations do not satisfy the rule of detachment if they are to convey a standard concept of consistency as discussed in the next section.

## 6 Rule of detachment, deduction principle and nonmonotonicity

In a number of inference systems having at least implication and negation as connectives, consistency, i.e. the property that not every formula follows from a theory, is characterized by

. the ex falso quodlibet:

$\mathbf{T} \vdash A \Rightarrow (\neg A \Rightarrow B)$

A noticeable fact is that the ex falso quodlibet is not compatible with the rule of detachment for any CMP-regular inference relation having a quite common nonmonotonic character.

**Property 12**: Let $|\sim$ be an CMP-regular inference relation with an underlying consequence relation $\vdash$ satisfying the ex falso quodlibet. If $|\sim$ is nonmonotonic such that $\mathbf{T} |\sim A$ and $\mathbf{T},\neg A |\not\sim A$ for some $\mathbf{T}$ and $A$ then $|\sim$ cannot satisfy the rule of detachment.

Proof: Assume $\mathbf{T} |\sim A$ and $\mathbf{T},\neg A |\not\sim A$. Now, $\mathbf{T} \vdash A \Rightarrow (\neg A \Rightarrow A)$ by the ex falso quodlibet for $\vdash$. Since $|\sim$ is CMP-regular, $\mathbf{T} |\sim \neg A \Rightarrow A$. Assuming that $|\sim$ satisfies the rule of detachment, $\mathbf{T},\neg A |\sim A$, a contradiction. □

Application of Property 12 shows that circumscription, including its well-founded fragment and default logic, including its normal fragment, do no respect the rule of detachment.

Property 12 as formulated above is, in light of Property 2, Property 4 and Property 6, the most general statement about the incompatibility between nonmonotonicity and the rule of detachment applied to the ex falso quodlibet. However, explicitly stating the somehow weaker Property 13 may be worthwhile.

**Property 13**: Let $|\sim$ be a MP – regular inference relation satisfying the ex falso quodlibet. If $|\sim$ is nonmonotonic such that $\mathbf{T} |\sim A$ and $\mathbf{T},\neg A \not|\sim A$ for some $\mathbf{T}$ and $A$ then $|\sim$ cannot satisfy the rule of detachment.

**Proof**: Assume $\mathbf{T} |\sim A$ and $\mathbf{T},\neg A \not|\sim A$. Now, $\mathbf{T} |\sim A \Rightarrow (\neg A \Rightarrow A)$ by the ex falso quodlibet. Since $|\sim$ is MP – regular, $\mathbf{T} |\sim \neg A \Rightarrow A$. Assuming that $|\sim$ satisfies the rule of detachment, $\mathbf{T},\neg A |\sim A$, a contradiction. □

Also, a slight variant to Property 12 and Property 13 is worth mentioning, which can be introduced by noticing that, for a consequence relation, the ex falso quodlibet gives rise to a sufficient condition concerning inconsistency, that is,

. if $\mathbf{T} \vdash A$ then $\mathbf{T},\neg A \vdash B$ for all $B$

If, conversely, inconsistency only arises through the ex falso quodlibet then a necessary condition is in force, that is,

. if $\mathbf{T},\neg A \vdash B$ for all $B$ then $\mathbf{T} \vdash A$

which holds for intuitionistic and classical sentential logics for instance. Now, in the formulation of Property 12 and Property 13, let the consequence relation involved to allow for both the necessary and sufficient conditions just given and let relative consistency to hold. Then Property 12 and Property 13 hold not only for the special, no matter how fundamental, form of nonmonotonicity mentioned but indeed for any form of nonmonotonicity.

In contrast, nonmonotonicity and the deduction principle appear not to be connected.

**Property 14**: Not every nonmonotonic MP – regular inference relation violates the deduction principle

**Proof**: First, an inference relation $|\sim$ with underlying consequence relation $\vdash$ representing classical sentential logic has to be specified. To this end, a propositional letter $P$ is fixed. In case $\mathbf{T} \vdash \neg P$ then $\mathbf{T} |\sim A$ iff $\mathbf{T} \vdash A$ otherwise $\mathbf{T} |\sim A$ iff $\mathbf{T},P \vdash A$. This definition clearly yields the nonmonotonic MP – regular character of $|\sim$. That $|\sim$ conforms to the deduction principle is proved next. Assume $\mathbf{T},A |\sim B$. In case $\mathbf{T},A \vdash \neg P$ then $\mathbf{T},A \vdash B$ so that $\mathbf{T} \vdash A \Rightarrow B$ and $\mathbf{T},P \vdash A \Rightarrow B$. From either, $\mathbf{T} |\sim A \Rightarrow B$ follows by conservativeness. In case $\mathbf{T},A \not\vdash \neg P$ then $\mathbf{T} \not\vdash \neg P$ by monotonicity. So, $\mathbf{T},A,P \vdash B$ due to $\mathbf{T},A |\sim B$ and if $\mathbf{T},P \vdash A \Rightarrow B$ then $\mathbf{T} |\sim A \Rightarrow B$. Since $\mathbf{T},A,P \vdash B$ then $\mathbf{T},P \vdash A \Rightarrow B$ so that $\mathbf{T} |\sim A \Rightarrow B$ is obtained. □

**Property 15**: Not every nonmonotonic CMP – regular inference relation satisfies the deduction principle

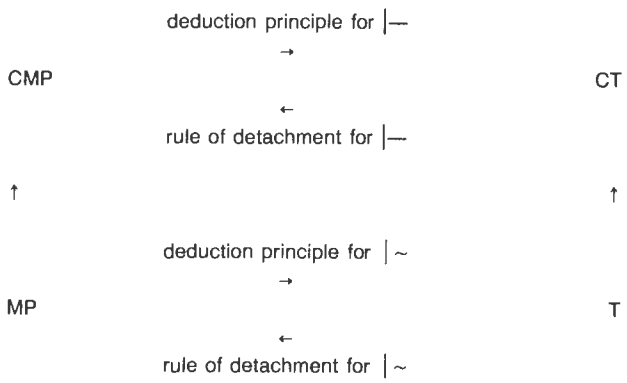
**Proof**: Define a CMP – regular $|\sim$ based on classical sentential logic by fixing two propositional letters $P$ and $Q$ and stating that in case $\mathbf{T} \not\vdash P$ and $\mathbf{T} \not\vdash \neg(Q \Rightarrow P)$ then $\mathbf{T} |\sim A$ iff $\mathbf{T},Q \Rightarrow P \vdash A$ otherwise if $\mathbf{T} \not\vdash \neg(P \Leftrightarrow Q)$ then $\mathbf{T} |\sim A$ iff $\mathbf{T},P \Leftrightarrow Q \vdash A$ else $\mathbf{T} |\sim A$ iff $\mathbf{T} \vdash A$. It can be easily checked that $|\sim$ is nonmonotonic and CMP – regular. If $\mathbf{T}$ consists of the unique formula $P \vee Q$ then $\mathbf{T} |\sim P$ and $\mathbf{T},P |\sim Q$ but $\mathbf{T} \not|\sim P \Rightarrow Q$ that is, the deduction principle does not hold for $|\sim$. □

## 7 Conclusion

In this paper, a research initiated by Gabbay and pursued by Makinson is addressed in such a manner that a complete picture of existing nonmonotonic inference systems is given by way of the properties they enjoy or not. However, investigating rather weak characterizations of inference systems turns out to be helpful for isolating specific features of nonmonotonicity thus meeting Gabbay's original motivation. For instance, identifying general properties of inference systems leads to a result establishing that

existing nonmonotonic inference systems strongly deny the rule of detachment. In fact, further development of the approach proposed in this paper, either along these lines or devoted for instance to nonmonotonic inference systems based on conditional logics with weak implication [Nute 1986] [Delgrande 1987] thus appear rather promising.

## 8 Appendix: Schema of types of regular inference relations




## 9 References


Besnard Ph. [1988]
*Gentzen/Tarski – Style Study of Non – Monotonic Inference*, in preparation.

Bobrow D. (ed.) [1980]
*Special Issue On Non – Monotonic Logics*, Artificial Intelligence 13 (1 – 2).

Delgrande J. [1987]
*A First – Order Logic for Prototypical Properties*, Artificial Intelligence 33, pp. 105 – 130.

Gabbay D. [1985]
*Theoretical Foundations for Non – Monotonic Reasoning in Expert Systems*, in: Logics and Models of Concurrent Systems (Apt ed.), Springer – Verlag, Berlin.

Gallaire H. & Minker J. (eds.) [1978]
*Logic and Databases*, Plenum Press, New York.

Gentzen G. [1969]
*Collected Papers of Gerhard Gentzen* (Szabo ed.), North Holland, Amsterdam.

Makinson D. [1987]
*General Theory of Non – Monotonic Inference*, draft.

Nute D. [1986]
*A Non – Monotonic Logic Based on Conditional Logic*, RR/01 – 0007, ACMC, University of Georgia, Athens.

Tarski A. [1956]
*Logic, Semantics, Metamathematics. Papers from 1923 – 1938.* (Woodger ed.), Clarendon Press, Oxford.

# Probabilistic Causal Reasoning

**Thomas Dean**[1] and **Keiji Kanazawa**
Department of Computer Science
Brown University
Box 1910, Providence, RI 02912

## Abstract

Predicting the future is an essential component of decision making. In most situations, however, there is not enough information to make certain predictions. In this paper, we develop a theory of causal reasoning for predictive inference under uncertainty. We emphasize a common type of prediction that involves reasoning about *persistence*: whether or not a proposition once made true remains true at some later time. We provide a decision procedure with a polynomial-time algorithm for determining the probability of the possible consequences of a set events and initial conditions. Problems in dealing with persistence by nonmonotonic temporal reasoning schemes are avoided by the use of simple probability information. The ideas in this paper have been implemented in a prototype system that refines a database of causal rules in the course of applying those rules to construct and carry out plans in a manufacturing domain.

Keywords: uncertainty, causal reasoning, probability, temporal reasoning

## I. Introduction

We are interested in the design of robust inference systems for generating and executing plans in routine manufacturing situations. We hope to build autonomous agents capable of dealing with a fairly circumscribed set of possibilities in a manner that demonstrates both strategic reasoning (the ability to anticipate and plan for possible futures) and adaptive reasoning (the ability to recognize and react to unanticipated conditions). In this paper, we develop a computational theory for temporal reasoning under uncertainty that is well suited to a wide variety of dynamic domains.

The domains that we are interested in have the following characteristics: (i) things cannot always be predicted accurately in advance, (ii) plans made in anticipation of pending events often have to be amended to suit new information, and (iii) the knowledge and ability to acquire predictive rules is severely limited by the planner's experience. Reasoning in such domains often involves making choices quickly on the basis of incomplete information. Although predictions can be inaccurate, it is often worthwhile for a planner to attempt to predict what conditions are likely to be true in the future and generate plans to deal with them.

Our theory includes (i) a polynomial-time decision procedure for probabilistic inference about temporally-dependent information, (ii) a space and time efficient method for refining probabilistic causal rules, and (iii) a mechanism to support planners in recognizing potential plan failures. This paper is primarily concerned with the first two of these. Details concerning the other are provided in the extended paper [Dean and Kanazawa, 1987a].

## II. Probabilistic Causal Theories

In order to explore some of the issues that arise in causal reasoning, we will consider some examples involving a robot foreman that directs activity in a factory. The robot has a plan of action that it is continually executing and revising. Among its tasks is the loading of trucks for clients. If our robot learns that a truck is more likely to leave than it previously believed, then it should consider revising its plans so that this truck will be loaded earlier. If, on the other hand, it predicts that all trucks will be loaded ahead of schedule, then it should take advantage of the opportunity to take care of other tasks which it did not previously consider possible in the available time.

In order to construct and revise its plan of action, the robot makes use of a fairly simple model of the world: a special-purpose theory about the cause-and-effect relationships that govern processes at work in the world (referred to as a *causal theory*). The robot's causal theory consists of two distinct types of rules which we will refer to as *projection rules* and *persistence rules*. We will defer discussion of persistence rules for just a bit.

As an example of a projection rule, the robot might have a rule that states that if a client calls in an order, then, with some likelihood, the client's truck will eventually arrive to pick up the order. The consequent prediction, in this case the arrival of a client's truck, is con-

[1]This work was supported in part by the National Science Foundation under grant IRI-8612644 and by an IBM faculty development award.

proposition in order to get rid of a lingering or persistent proposition: a feat that often proves difficult in nontrivial domains. If a commuter leaves his newspaper on a train, it is not hard to predict that the paper is not likely to be there the next time he rides on that train; however, it is quite unlikely that he will be able to predict what caused it to be removed or when the removal occurred.

When McDermott first proposed the notion of persistence as a framework for reasoning about change [McDermott, 1982], he noted that persistence might be given a probabilistic interpretation. That is exactly what we do here. We replace the single default rule of persistence used in most planning systems with a set of (probabilistic) rules: one or more for each fluent that the system is aware of. Our robot might use a persistence rule to reason about the likelihood that a truck driver will still be waiting at various times following his arrival at the factory. The information derived from applying such a rule might be used to decide which truck to help next or how to cope when a large number of trucks are waiting simultaneously. Each persistence rule has the form $PERSIST(P, \rho)$, where $P$ is a fluent and $\rho$ is a function of time referred to as a *survivor* function [Syski, 1979]. In our implementation, we consider only two types of survivor functions: exponential decay functions and piecewise linear functions. The former are described in Section IV., and the latter, requiring a somewhat more complex analysis, are described in [Dean and Kanazawa, 1987a]. Exponential decay functions are of the form $e^{-\lambda t}$ where $\lambda$ is the constant of decay. Persistence rules referring to exponential decay functions are notated simply $PERSIST(P, \lambda)$. Such functions are used, for example, to indicate that the probability of a truck remaining at the dock decreases by 5% every 15 minutes. The persistence rule $PERSIST(P, \lambda)$ encodes the fact that:

$$p(HOLDS(P, t) \mid HOLDS(P, t - \Delta)) = e^{-\lambda(t-\Delta)}$$

where $\Delta$ is a positive number indicating the length of an interval of time. Exponential decay functions are insensitive to changes in the time of occurrence of events that cause such propositions to become true, and, hence, are easy to handle efficiently.

There are a number of issues that every computational approach to reasoning about causality must deal with. One such issue involves reasoning about dependent causes [Pearl, 1985] (*e.g.,* the application of two probabilistic causal rules that have the same consequent effects, both of which appear to apply in a given set of circumstances but whose conditions are correlated). Another issue concerns handling other forms of incompleteness and nonmonotonic inference [Ginsberg, 1985] [Dean and Boddy, 1987] (*e.g.,* the robot might have a general rule for reasoning about the patience (persistence) of truck drivers waiting to be served and a special rule for how they behave right around lunch time or late in the day). While we agree that these problems are important, we do not claim to have any startling new insights into their solution. There is one area, however, in which our theory does offer some new insights, ditioned on two things: an event referred to as the *triggering event*, in this case the client calling in the order, and an enabling condition corresponding to propositions that must be true at the time the triggering event occurs. For example, the rule just mentioned might be conditioned on propositions about the type of items ordered, whether or not the caller has an account with the retailer, or the time of day. The simplest form of a projection rule is $PROJECT(P_1 \wedge P_2 \ldots \wedge P_n, E, R, \kappa)$. This says that $R$ will be true with probability $\kappa$ immediately following the event $E$ given that $P_1$ through $P_n$ are true at the time $E$ occurs. Restated as a conditional probability, this would be:

$$p(HOLDS(R, t + \epsilon) \mid HOLDS(P_1 \wedge P_2 \ldots \wedge P_n, t) \wedge OCCURS(E, t))) = \kappa$$

We assume that $P_1$ through $P_n$ are independent. Projection rules are applied in a purely antecedent fashion (as in a production system) by the inference engine we will be discussing. The objective is to obtain an accurate picture of the future in order to support reasoning about plans [Dean, 1987a] [Dean, 1987b].

Our approach, as described up to this point, is fairly traditional and might conceivably be handled by some existing approach [Pearl, 1985] [Duda *et al.*, 1981]. What distinguishes our approach from that of other probabilistic reasoning approaches is that we are very much concerned with the role of time and in particular the tendency of certain propositions (often referred to as *fluents* [McCarthy and Hayes, 1969]) to change with the passage of time. By adding time as a parameter to our causal rules, we have complicated both the inference task and the knowledge acquisition task. Complications notwithstanding, the capability to reason about change in an uncertain environment remains an important prerequisite to robust performance in most domains. We simply have to be careful to circumscribe a useful and yet tractable set of operations. In our case, we have allowed the computational complexity of the reasoning tasks and the availability and ease of acquisition of the data to dictate the limitations of our inference mechanism.

Our inference system needs to deal with the imprecision of most temporal information. Even if a robot is able to consult a clock in order to verify the exact time of occurrence of an observed event, most information the robot is given is imprecise (*e.g.,* a client states that a truck will pick up an order at *around* noon, or a delivery is scheduled to arrive *sometime* in the next 20 minutes). One of the most important sources of uncertainty involves predicting how long a condition lasts once it becomes true (*i.e.,* how long an observed or predicted fact is likely to *persist*). In most planning systems (*e.g.,* [Sacerdoti, 1977]) there is a single (often implicit) default rule of persistence [Dean and McDermott, 1987] that corresponds more or less to the intuition that a proposition once made true will remain so until something makes it false. The problem with using this rule is that it is necessary to predict a contravening

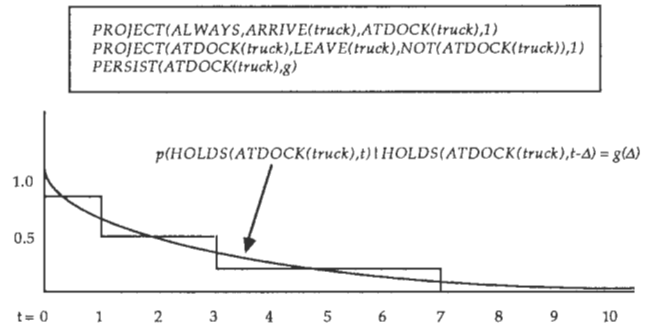



Figure 1: A simple causal theory illustrating the use of survivor functions

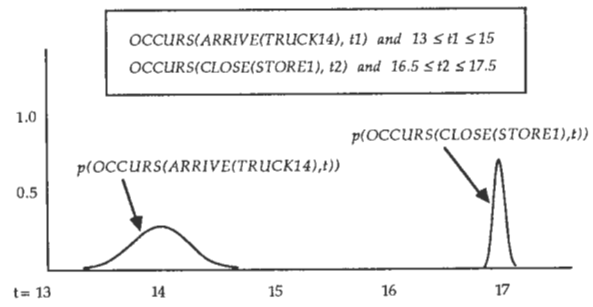



Figure 2: A set of basic facts and their probabilistic interpretation

and that concerns the form of probability functions used in causal rules and how they can be used to efficiently predict the causal consequences.

# III. Probabilistic Projection

In this section, we will try to provide some intuition concerning the process of reasoning about persistence, which we will refer to as *probabilistic projection*. A planner is assumed to maintain a picture of the world changing over time as a consequence of observed and predicted events. This picture is formed by extrapolating from certain observed events (referred to as *basic facts*) on the basis of rules believed to govern objects and agents in a particular domain. These governing rules are collectively referred to as a *causal theory*.

Figure 1 depicts a simple causal theory. Operators (*HOLDS*, *OCCURS*), predicates (*ATDOCK*), and constants (*TRUCK14*) are in upper case, while functions ($p$, $g$) and variables ($t$, *truck*) are in lower case. We refer to an instance of a fact (type) being true over some interval of time as a *time token*, or simply *token*. For example, *ARRIVE*(*TRUCK14*) denotes a general type of event whereas $OCCURS(ARRIVE(TRUCK14), t)$ denotes

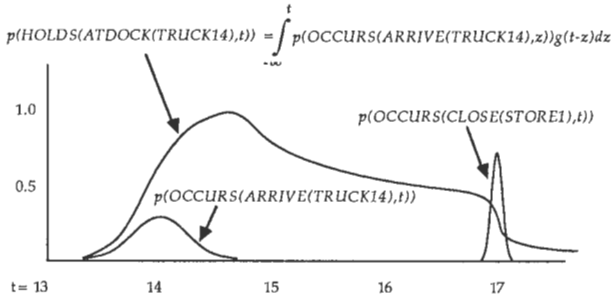



Figure 3: An example of simple probabilistic inference about persistence

a particular instance of *ARRIVE*(*TRUCK14*) becoming true. The predicate *ALWAYS* is timelessly true (*i.e.*, $\forall t\ HOLDS(ALWAYS, t)$). The function $\rho$, a survivor function, describes how certain types of propositions are likely to persist in lieu of further supporting or contravening information.

Figure 2 shows a set of basic facts corresponding to two events assumed in our example to occur with probability 1.0 within the indicated intervals. The system assumes that there is a distribution describing the probability of each event occurring at various times, and uses some default distribution if no distribution is provided.

Evidence concerned with the occurrence of events and the persistence of propositions is combined to obtain a probability function $\pi$ for a proposition $Q$ being true at various times in the future by convolving the density function $f$ for an appropriate triggering event with the survivor function $\rho$ associated with $Q$:

$$\pi(t) = \int_{-\infty}^{t} f(z)\rho(t-z)dz \qquad (1)$$

Figure 3 illustrates a simple instance of this kind of inference. Note that the range of the resulting probability function is restricted; after the point in time labeled 17, the persistence of *ATDOCK*(*TRUCK14*) is said to be *clipped*, and thereafter its probability is represented by another function not shown.

All probability computations are performed incrementally in our system. Each token has associated with it a vector which is referred to as its *expectation vector* that records the expected probability that the proposition corresponding to the token's type will be true at various times in the future.

The system updates the expectation vectors every time new propositions are added to the database, and also at regular intervals as time passes. In the update, a single pass sweep forward in time is made through the database. There is, according to the domain and granularity of data, a fixed *time step*, or a quantum by which we partition time. Starting at the "present time," we compute for each propo-

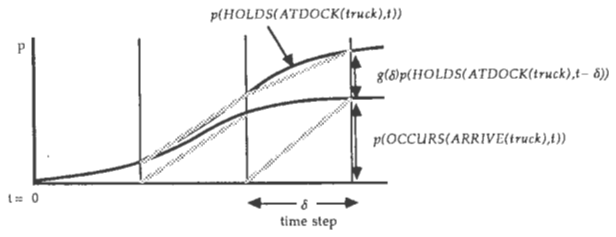



Figure 4: Computing the convolution integral incrementally

sition its expected probability for the time step according to the causal theory governing that type of proposition, and record it in the expectation vector. We compute the probability for all propositions, before moving on to the next time step. The process is repeated for some finite number of time steps.

For event causation, the update is straightforward; in the simplest cases, it is just a table lookup and copying of the density function into the vector. For the convolution, it is necessary to take steps to avoid computing the convolution integral afresh at each time step. We compute the convolution as a Riemann sum, successively summing over the time axis with a mesh of fixed size (the time step). By using the exponential decay form of survivor functions, it is possible to compute the convolution at a time step by looking only at the value for the last time step, independent of the time at which the proposition of interest became true. All that is required is to multiply the last value by the constant decay rate, and add it to any contribution from the causal distribution for that time step. The process is illustrated graphically in figure 4.

There are many details concerned with indexing and applying projection rules that will not be mentioned in this paper (but see [Dean and McDermott, 1987]). The details of probabilistic projection using exponential decay functions are described in Section IV.. Our update algorithm is polynomial in the product of the number of causal rules, the size of the set of basic facts, and the size of the mesh used in approximating the integrals. For many practical situations, performance is closer to linear in the size of the set of basic facts.

The convolution equation can be easily extended to handle the case of clipping. We add to (1) a term, the function $g$, corresponding to the distribution of an event which clips the state of a fact being true.

$$\pi(t) = \int_{-\infty}^{t} f(z)e^{-\lambda(t-z)}[1 - \int_{z}^{t} g(w)dw]dz \qquad (2)$$

The cumulative distribution of $g$ defines the degree to which it becomes unlikely that the fact represented by $\pi$ remains true in the world. We see that under certain conditions, (2) describes exactly what we desire. Unfortunately, there will be a tendency for the decay function and $g$ to count the same effects twice. In [Dean and Kanazawa, 1987b] we address methods by which this problem can be attacked in a different probabilistic framework.

# IV. Algorithmic Considerations

Probabilistic causal theories are composed of two types of rules, projection rules:

$$PROJECT(P_1 \wedge P_2 \ldots \wedge P_n, E, R, \kappa)$$

and persistence rules:

$$PERSIST(Q, \lambda)$$

where $P_1$ through $P_n$, $R$, and $Q$ are all fact types, and $E$ is an event type. We assume (statistical) independence of fact types so that, if we are interested in the conjunction $P_1 \wedge P_2 \ldots \wedge P_n$, we can assume that

$$p(HOLDS(P_1 \wedge P_2 \ldots \wedge P_n, t)) = \prod_{i=1}^{n} p(HOLDS(P_i, t)) \quad (3)$$

We define a relation $\prec_C$ on fact types so that $Q \prec_C R$ just in case there exists a rule of the form $PROJECT(P_1 \wedge P_2 \ldots \wedge P_n, E, R, \kappa)$ where $P_i = Q$ for some $i$. For any given set of causal rules, the graph $\mathcal{G}_{\prec_C}$ whose vertices correspond to fact types and whose arcs are defined by $\prec_C$ is likely to have cycles; this will be the cause of a small complication that we will have to resolve later. In this paper, we distinguish between fact types corresponding to propositions that hold over intervals and event types corresponding to instantaneous (point) events. For each occurrence (token) of a point event of type $E$, we will need its density function $p(OCCURS(E, t))$. Probabilistic projection takes as input a set of initial events and their corresponding density functions. Given the restricted format for projection rules, the only additional point events are generated by the system in response to the creation of new instances of fact types. For each token of fact type $P$, we identify a point event of type $TRANS(P)$ corresponding to the particular instance of that fact becoming true. In the process of probabilistic projection, we will want to compute the corresponding density function $p(OCCURS(TRANS(P), t)$. In addition to computing density functions, we will also want to compute the mass functions $p(HOLDS(P, t))$ for instances of facts.

In order to describe the process of probabilistic projection, we will divide the process into two different stages: *deterministic causal projection* and *probabilistic causal refinement*. The actual algorithms are more integrated to take advantage of various pruning techniques, but this simpler, staged, process is somewhat easier to understand. Deterministic causal projection starts with a set of tokens and a set of projection rules and generates a set of new tokens T by scanning forward in time and applying the rules

without regard for the indicated probabilities. This stage can be carried out using any number of simple polynomial algorithms (see [Dean and McDermott, 1987] [Hanks and McDermott, 1986]) and will not be further detailed here. Probabilistic causal refinement is concerned with computing density and mass functions for tokens generated by deterministic causal projection. In the following, all density and mass functions are approximated by step (*i.e.*, piecewise constant) functions. We represent these functions of time using vectors (*e.g.*, $mass(T)$ denotes the mass function for the token $T$ and $mass_i(T)$ denotes the value of the function at $t = i$). For each fact token $T_P$, we create a corresponding event token $T_{TRANS(P)}$ and define a vector $mass(T_P)$. For each event token $T_E$, we define a vector $density(T_E)$. We define an upper bound $\Omega$ on projection and assume that each mass and density vector is of length $\Omega^2$. Initially, we assume that

$$\forall T \in \mathsf{T} : 1 \leq i \leq \Omega : density_i(T) = 0 \wedge mass_i(T) = 0$$

Event tokens are supplied by the user in the form

$$\kappa = \int_{\mathsf{est}}^{\mathsf{lst}} p(OCCURS(E,t))dt$$

where est and lst correspond (respectively) to the earliest and latest start time for the token and $\kappa$ is the probability that the event will occur at all. We assume that the density function for such an event is defined by a Gaussian distribution over the interval from est to lst. For a token $T_E$ corresponding to a user-supplied initial event, it is straightforward to fill in $density(T_E)$. Probabilistic causal refinement is concerned with computing $mass_i(T_P)$ and $density_i(T_{TRANS(P)})$ for all fact tokens $T_P$ and all event tokens $T_{TRANS(P)}$. We partition the set of tokens $\mathsf{T}$ into fact tokens $\mathsf{T_F}$ and event tokens $\mathsf{T_E}$. Probabilistic causal refinement can be defined as follows:

```
Procedure: refine(T)
  for i = 1 to Ω:
      begin
        for T ∈ T_E:
            density-update(T, i);
        for T ∈ T_F:
            mass-update(T, i);
      end
```

Of course, all of the real work is done by density-update and mass-update. Each token has associated with it a specific *derivation* that is used in computing its mass or density. For a token $T_{TRANS(R)}$, this derivation corresponds to a rule of the form

$$PROJECT(P_1 \wedge P_2 \ldots \wedge P_n, E, R, \kappa)$$

and a set of antecedent tokens $\{T_E, T_{P_1}, T_{P_2} \ldots T_{P_n}\}$ used to instantiate the rule and generate the consequent token $T_R$. Given that

$$p(OCCURS(TRANS(R)), t) = \kappa * p(OCCURS(E,t)) * p(HOLDS(P_1 \wedge P_2 \ldots \wedge P_n, t)$$

and, assuming independence (3), we have

```
Procedure: density-update(T_TRANS(R), i)
  density_i(T_TRANS(R)) ←
      κ * density_i(T_E) * ∏_{j=1}^{n} mass_i(T_{P_j})
```

There is one problem with this formulation: it relies on all the mass and density functions for the antecedent conditions being already computed for the instant $i$. In the present algorithm, refine takes no care in ordering the tokens in $\mathsf{T}$. There are a number of ways of ensuring that the updates are performed in the correct order. The easiest is to partially order $\mathsf{T}$ according to $\prec_C$ and insist that $\mathcal{G}_{\prec_C}$ be acyclic, but this would preclude the use of most interesting causal theories. A more realistic method is to partition $\mathsf{T}$ with respect to an instant $i$ into those tokens that are *open* and those that are *closed*. Deterministic causal projection defines an earliest start time (est) for each token; for event tokens a latest start time (lst) is specified. An event token is open throughout the interval est to lst and closed otherwise. For fact tokens, we modify probabilistic causal refinement so that it closes a fact token $T_P$ as soon as $mass_i(T_P)$ drops below a fixed threshold. A fact token is open from its est until it is closed. All we require then is that for any $i$ the set of tokens that are open define an acyclic causal dependency graph using $\prec_C$. This restriction still allows for a wide range of causal theories. To get refine to do the right thing, we would have to apply refine only to open tokens and either sort the tokens using $\prec_C$, or (as is actually done) define refine so that if, in the course of updating a consequent token, refine finds an antecedent token that hasn't yet been updated, it applies itself recursively.

The derivation of a token $T_P$ corresponds to a rule of the form $PERSIST(P, \lambda)$ where $\lambda$ is the constant of decay for the fact type $P$, and an event token $T_{TRANS(P)}$. The procedure mass-update is a bit more difficult to define than density-update since it depends upon the type of decay functions used in persistence rules. In the case of exponential decay functions, the operation of density-update is reasonably straightforward.

Recall the basic combination rule for probabilistic projection:

$$\pi(t) = \int_{-\infty}^{t} f(x)\rho(t-x)dx$$

and suppose that $\rho$ is of the form $e^{-\lambda z}$ where $\lambda$ is some constant of decay, and that $f$ can be approximated by a step function as in

$$f(x) = \begin{cases} C_1 & s_0 \leq x < s_1 \\ C_2 & s_1 \leq x < s_2 \\ \ldots & \\ C_n & s_{n-1} \leq x < s_n \end{cases}$$

[2] There are some obvious optimizations to be made here.

We will take advantage of the fact that

$$\int_{s_j}^{s_k} f(x)dx = \sum_{i=j}^{k-1} \int_{s_i}^{s_{i+1}} f(x)dx$$

and

$$\rho(s_{k+1} - x) = e^{-\lambda\delta}\rho(s_k - x)$$

where $\delta = s_{k+1} - s_k$.

Making appropriate substitutions, we have

$$\begin{aligned}
\pi(s_{k+1}) &= \sum_{i=j}^{k} \int_{s_i}^{s_{i+1}} f(x)\rho(s_{k+1} - x)dx \\
&= \sum_{i=j}^{k-1} \int_{s_i}^{s_{i+1}} f(x)\rho(s_{k+1} - x)dx \\
&\quad + \int_{s_k}^{s_{k+1}} f(x)\rho(s_{k+1} - x)dx \\
&= e^{-\lambda\delta} \sum_{i=j}^{k-1} \int_{s_i}^{s_{i+1}} f(x)\rho(s_k - x)dx \\
&\quad + \int_{s_k}^{s_{k+1}} f(x)\rho(s_{k+1} - x)dx \\
&= e^{-\lambda\delta}\pi(s_k) + \int_{s_k}^{s_{k+1}} f(x)\rho(s_{k+1} - x)dx
\end{aligned}$$

It should be clear that updates depending upon such simple survivor functions can be performed quite quickly. Integration is approximated using Riemann sums with a mesh of fixed size (roughly) corresponding to $\delta$. We define the procedure mass-update as follows:

Procedure: mass-update($T_P, i$)
  $mass_i(T_P) \leftarrow$
    $e^{\lambda_P\delta} mass_{i-1}(T_P) + density_i(T_{TRANS(P)})$

The actual algorithms are complicated somewhat by the fact that the choice of mesh size may not coincide precisely with the steps in the step functions approximating survivor functions and distributions. We compensate for this by using a somewhat finer mesh in the update algorithms. The fact that we employ a fixed mesh size still causes small errors in the accuracy of the resulting mass and density functions, but these errors can be controlled. We have tried to make a reasonable tradeoff, taking into account that the finer the mesh the larger the mass and density vectors. Given that the step functions used for encoding survivor functions and distributions are only approximations, there is a point past which employing a finer mesh affords no additional information. We have found that a mesh size of half the smallest step in any step function works quite well in practice.

# V. Acquiring Persistence Rules

Statistical methods have not seen particularly wide application in AI. This is largely due to problems concerning the availability of the data necessary to employ such methods. Data provided from experts has been labeled as unreliable, and the use of priors in Bayesian inference has been much maligned. An alternative to expert judgements and estimating priors, is to integrate the data acquisition process into your system: have it gather its own data. In such a scheme, all predictions made by the system are conditioned only upon what the system has directly observed. Of course, this is unrealistic in many cases (*e.g.,* diagnostic systems whose decisions could impact on the health or safety of humans). In the industrial automation applications considered in this paper, however, not only is it practical, but it appears to be crucial if we are to build systems capable of adapting to new situations.

In this section, we describe a system for continually refining a database of probabilistic causal rules in the course of routine planning and execution. Given the focus of this paper, we will concern ourselves exclusively with the acquisition (or refinement) of persistence rules. Our warehouse planner keeps track of how long trucks stay around and uses this information to construct survivor functions for various classes of trucks. The system must be told which quantities it is to track and how to distinguish different classes of trucks, but given that, the rules it acquires are demonstrably useful and statistically valid in the limit.

The survivor function for a given class of trucks is computed from a set of data points corresponding to instances of trucks observed arriving and then observed leaving without being loaded[3]. It should be clear that, in general, a collection of data points will not define a survivor function uniquely. There are many ways in which to derive a reasonable approximation for such a function. For example, we might employ some form of curve fitting based on an expected type of function and the sample data. While such methods may yield more accurate approximations in some cases, for our application, there are simpler and more efficient methods.

Our system derives two parameters for constructing survivor functions. The first is an estimate of the delay (*i.e.,* the initial interval of time during which the function remains constant), and the second corresponds to the rate of decay during the function's period of descent. We simply use the arithmetic mean of the samples to compute the rate of decay. With both of the simple classes of functions we have considered, the exponential decay and the linear decay functions, computing, respectively, the persistence parameter ($\lambda$) and the slope is trivial. In the case of an exponential decay, we use the mean as the half-life of the function.

Computing the delay interval is also quite simple. Recall that, in our examples, the delay corresponds to the

[3]There is actually more information to be had. For example, instances of trucks observed arriving and subsequently observed to be absent (the exact time of leaving unknown) are presumably relevant to the problem at hand, and, in fact, it is possible to make use of such information by making various additional assumptions. We will not, however, consider such complications here.

interval of time during which no trucks are likely to leave. Each data point is represented as an integer corresponding to how long a particular truck stayed around. Keeping in mind that there will be occasional aberrations, we choose to ignore some percentage of the data points corresponding to those that are far from the mean. There are more sophisticated means of doing this, but we simply sort the data points for each class of trucks in increasing order, and set the delay to be the length of time corresponding to some data point in the $k$th percentile of the resulting sorted list, where $k$ defaults to 5. This provides a reasonable approximation to the actual functions, and it is very fast to compute.

We can now sketch the simple algorithm utilized in our system. As noted, we need to collect data for each class of interest. The data for each class is collected in a data structure along with various intermediate quantities used by the update algorithm (*e.g.,* since the algorithm calls for the arithmetic mean of the data points it is convenient to incrementally compute the sum of the elements of the collection). The *class* data type has the following accessor functions associated with it ($c$ is an instance of *class*):

- type($c$): the type of the associated survivor function: linear or exponential
- lambda($c$): the rate or slope
- delay($c$): the delay
- history($c$): a vector corresponding to the sorted collection of data points (individual data points are referenced using history($c$)($i$) where $i$ is an integer index)
- insts($c$): the number of data points in the collection
- sum($c$): the sum of the items in the collection
- offset($c$): a percentile indicating the bottom $n$ data points in the sorted collection to be ignored in computing the delay (defaults to 5%)

Assuming that $c$ is an instance of *class* and $p$ is a new data point, the acquisition algorithm can be described as follows:

```
Procedure: acquire(c, p)
  history(c) ← insert(p, history(c));
  insts(c) ← insts(c) + 1;
  sum(c) ← sum(c) + p;
  lambda(c) ← rate(c, ((sum(c)/insts(c)) − delay(c)));
  delay(c) ← history(c)(⌈insts(c) * offset(c)⌉)
```

The function insert is assumed to insert a data point into a sorted collection. The function rate depends on the type of survivor function used:

```
Function: rate(c, μ)
  if μ = 0
     then +∞
     else if type(c) = linear
             then 0.5/μ
             else if type(c) = exponential
                     then log 2/μ
```

Although we have tested our approach extensively in simulations and have found the acquired persistence data to converge very rapidly to the correct values, we do not claim that the above methods have any wider application. The simplicity of the algorithm and its incremental nature are attractive, but the most compelling reason for using it is that the algorithm works well in practice. Probabilistic projection does not rely upon a particular method for coming up with persistence rules. As an alternative, the data might be integrated off line, using more complex (and possibly more accurate) methods.

It should be noted that our system is given the general form of the rules it is to refine. It cannot, on the basis of observing a large set of trucks, infer that trucks from one company are more impatient than those from another company, and then proceed to create two new persistence rules where before there was only one. The general problem of generating causal rules from experience is very difficult. We are currently exploring methods for distinguishing different classes of trucks based on statistical clustering techniques (*e.g.,* Kolmogorov-Smirnov's D-statistic [Dunn and Clark, 1974] and Shapiro-Wilk's W-statistic [Shapiro and Wilk, 1965]). Using such methods, it appears to be relatively straightforward to determine that a given data set corresponds to more than one class, and even to suggest candidate survivor functions for the different classes. However, figuring out how to distinguish between the classes in order to apply the different survivor functions is considerably harder.

# VI. Conclusions

In this paper, we have sketched a theory of reasoning about change that extends previous theories [McDermott, 1982] [Shoham and Dean, 1985]. In particular, we have shown how *persistence* can be modeled in probabilistic terms. Probabilistic projection is a special case of reasoning about continuously changing quantities involving partial orders and other sorts of incomplete information, and as such it represents an intractable problem. We have tried to identify a tractable core in the inferences performed by probabilistic projection.

In [Dean and Kanazawa, 1987a], we describe a planning system capable of continually refining its causal rules. The system makes predictions, observes whether or not those predictions come to pass, and modifies its rules accordingly. It is capable of routine data acquisition and updates its probabilitistic rules in the course of everyday operation. Initial experiments with the prototype system have been very encouraging. We believe that the inferential and causal rule refinement capabilities designed into our system are essential for robots to perform robustly in routine manufacturing situations. We hope that our current investigations will yield a new view of strategic planning and decision making under uncertainty based on the idea of continuous probabilistic projection.

## References


[Dean, 1987a] Thomas Dean. An approach to reasoning about the effects of actions for automated planning systems. *Annals of Operations Research*, 1987.

[Dean, 1987b] Thomas Dean. Large-scale temporal data bases for planning in complex domains. In *Proceedings IJCAI 10*, Milan, Italy, IJCAI, 1987.

[Dean and Boddy, 1987] Thomas Dean and Mark Boddy. Incremental causal reasoning. In *Proceedings AAAI-87*, pages 196–201, Seattle, Washington, AAAI, 1987.

[Dean and Kanazawa, 1987a] Thomas Dean and Keiji Kanazawa. *Persistence and Probabilistic Inference.* Technical Report CS-87-23, Brown University Department of Computer Science, 1987.

[Dean and Kanazawa, 1987b] Thomas Dean and Keiji Kanazawa. *Probabilistic Temporal Reasoning.* Technical Report forthcoming, Brown University Department of Computer Science, 1987.

[Dean and McDermott, 1987] Thomas Dean and Drew V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[Duda *et al.*, 1981] R.O. Duda, P.E. Hart, and Nilsson N.J. Subjective bayesian methods for rule-based inference systems. In B.W. Webber and N.J. Nilsson, editors, *Readings in Artificial Intelligence*, Tioga, Palo Alto, CA, 1981.

[Dunn and Clark, 1974] Olive Jean Dunn and Virginia A. Clark. *Applied Statistics: Analysis of Variance and Regression.* John Wiley and Sons, 1974.

[Ginsberg, 1985] M.L. Ginsberg. Does probability have a place in non-monotonic reasoning? In *Proceedings IJCAI 9*, Los Angeles, CA, IJCAI, 1985.

[Hanks and McDermott, 1986] Steve Hanks and Drew V. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings AAAI-86*, pages 328–333, Philadelphia, Pa., AAAI, 1986.

[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.

[Pearl, 1985] Judea Pearl. A constraint propagation approach to probabilistic reasoning. In *Proceedings of the 1985 AAAI/IEEE Sponsored Workshop on Uncertainty and Probability in Artificial Intelligence*, 1985.

[Sacerdoti, 1977] Earl Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier Publishing Company, Inc., 1977.

[Shapiro and Wilk, 1965] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality. *Biometrika*, 52:591–612, 1965.

[Shoham and Dean, 1985] Yoav Shoham and Thomas Dean. Temporal notation and causal terminology. In *Proceedings Seventh Annual Conference of the Cognitive Science Society*, Cognitive Science Society, 1985.

[Syski, 1979] Ryszard Syski. *Random Processes.* Marcel Dekker, New York, 1979.

# Search Strategies For Conspiracy Numbers

*Norbert Klingbeil*
*Jonathan Schaeffer*

Computing Science Department,
University of Alberta,
Edmonton, Alberta
Canada T6G 2H1

Abstract

McAllester's *Conspiracy Numbers* algorithm is an exciting new approach to minimax search that builds trees to variable depth without application-dependent knowledge. The likelihood of the root achieving a value is expressed as that value's conspiracy number: the minimum number of leaf nodes required to change their value to cause the root to change to that value. Initial experience indicates that the algorithm places too much emphasis on depth rather than breadth of search. Several variations on the conspiracy numbers search strategy are reported, each adding an increasing degree of breadth to the search. The ideas have been implemented in a program that solves tactical chess problems. Experiments indicate that the new algorithms are capable of solving 41% more problems than McAllester's original proposal.

Keywords: conspiracy numbers, alpha-beta, minimax search, search algorithms.

## 1. Introduction

There are many well-known methods for efficiently searching minimax trees. *Alpha-beta* [5] and *SSS** [13], for example, are elegant algorithms that greatly reduce the search effort required. However, both have a fundamental limitation: they are constrained to fixed depth searches. This results in tremendous search overhead by the exploration of sub-trees which have small chances of success, yet must be considered to be certain the algorithm returns the correct result.

Disciples of artificial intelligence have long scoffed at these so called *brute-force* methods and sought a more knowledge-based approach. Berliner's *B** [1, 10] is an elegant best-first proof procedure that expands minimax trees to variable depths using knowledge to guide the search. Unfortunately, its application dependent knowledge requirements are its undoing: it is too difficult to reliably determine the *optimistic* and *pessimistic* values required to bound the true value of every node in the tree.

McAllester's *Conspiracy Numbers* algorithm is a new approach to minimax search [8, 9]. Rather than searching to a fixed depth, the algorithm selectively expands nodes in the tree until a specified degree of confidence is achieved in the root value. Confidence is defined by a value's conspiracy number: the minimum number of leaf nodes that must change their value (or *conspire*) to cause the root of the tree to change to that value. The novelty of the algorithm is that it selectively expands nodes in an *application independent* manner, without requiring, for example, extensive leaf node domain-dependent knowledge such as *B** requires. As the algorithm is new, there is little theoretical and experimental data on its performance.

This paper explores alternate search strategies for conspiracy numbers. McAllester's original paper described a strategy that resulted in continually expanding the left-most son of interior nodes [8]. Experiments show this method often results in a program descending to ridiculous depths that have no practical chances of success. By adding some breadth to the search, a significant improvement in performance is possible. These ideas have been implemented in a program that solves tactical chess problems. Experimentally, adding additional breadth to the tree and enhancing the algorithm to converge on the best son of the root, not just the best value, has resulted in a 41% increase in the problem solving capabilities of the program.

Section 2 provides an overview of the conspiracy numbers algorithm. In section 3, alternate search strategies are considered. These strategies have been implemented and the experimental results are

presented in Section 4. Finally, Section 5 presents the conclusions and further work.

## 2. Conspiracy Numbers

Conspiracy numbers provide a measure of the difficulty to change the current minimax value of a node. In Figure 1, assuming the root is a maximum node, how many leaf nodes in the tree have to change their value, as a result of being searched one ply deeper, to cause the value at the root (*Vroot*) to become 2? The simplest way would be if node $J$'s value changed to 2. Another way would be for both nodes $F$ and $G$ to change their values appropriately. Nodes $F$ and $G$ form a *set of conspirators* for increasing *Vroot* to 2; both have to *conspire* to achieve this result. Node $J$ also forms a set of conspirators for increasing *Vroot* to 2; in this case the *minimal* set. The minimum number of leaf nodes that must conspire to change *Vroot* to a specific value is called the *conspiracy number (CN)* for that value. Figure 2 shows the conspiracy numbers for Figure 1 along with the minimal set of conspirators for each value.

| Value | CN | Nodes to Change |
|---|---|---|
| -3 | 2 | (E and (F or G)) |
| -2 | 2 | (E and (F or G)) |
| -1 | 2 | (E and (F or G)) |
| 0 | 1 | (E or J) |
| 1 | 0 | |
| 2 | 1 | (J or K) |
| 3 | 2 | (E and (J or K)) or (F and G) |

**Figure 2. Conspirators.**

It turns out that there are simple recursive relations for calculating the conspiracy numbers of a node from the conspiracy numbers of its descendents. In what follows, let $m$ denote the minimax value of a node and $v$ denote the value we would like to change $m$ to.

At a leaf node, changing $m$ to any other value requires a conspiracy of only that node itself, and hence has a conspiracy number of 1. If we do not want to change the node's value, then no conspiracy is required and the conspiracy number is 0. If the leaf node is also a terminal node, then there is no way to change its value and a conspiracy number of $\infty$ is assigned. Hence, the conspiracy numbers for a leaf node are:

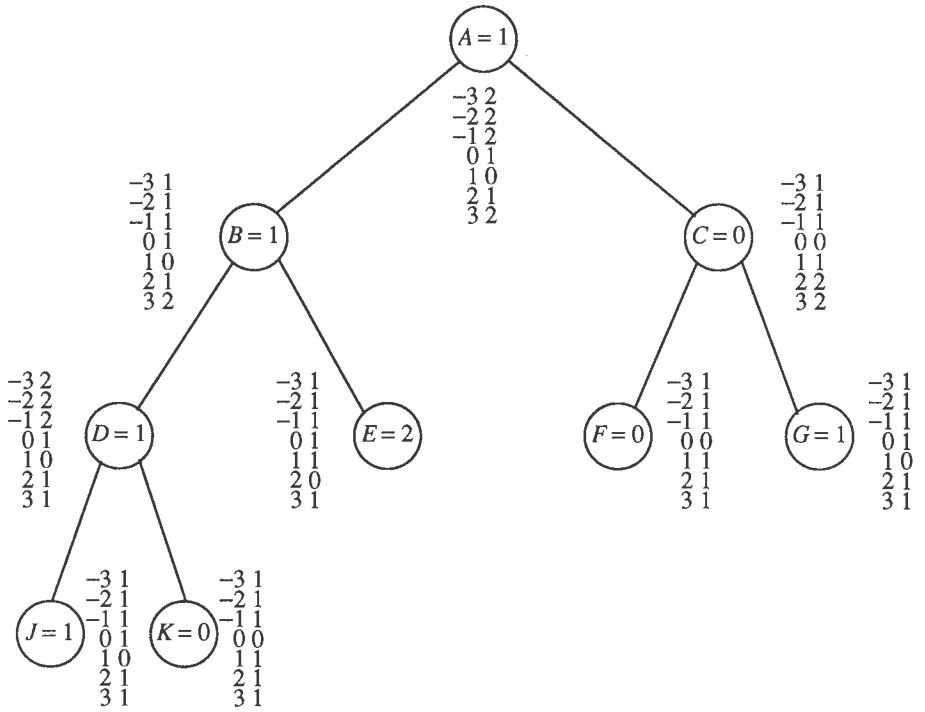



**Figure 1. Conspiracy Numbers.**

$$CN(v) = \begin{cases} 0 & \text{if} \quad v=m \\ 1 & \text{if} \quad v \neq m \\ \infty & \text{if} \quad \textit{terminal node} \end{cases}$$

At a maximizing interior node, to increase the value to $v$ requires only one of the sons to change its value to $v$. Assuming that the conspiracy number for each son has already been calculated, then the minimum number of conspirators required to increase the node to $v$, $\uparrow CN(v)$, is just the minimum number of conspirators to increase one of the sons to $v$. This yields the following relation:

$$\uparrow CN(v) = \begin{cases} 0 & \text{for all } v \leq m \\ \underset{\textit{all sons } i}{MIN} \uparrow CN_i(v) & \text{for all } v > m \end{cases}$$

To decrease the node's value to $v$, $\downarrow CN(v)$, requires all sons whose value is greater than $v$ to decrease their value to $v$. Given the minimal set of conspirators for decreasing each son to $v$, *all* members of each of these sets must conspire together to decrease the node's value to $v$. Therefore:

$$\downarrow CN(v) = \begin{cases} 0 & \text{for all } v \geq m \\ \sum_{\textit{all sons } i} \downarrow CN_i(v) & \text{for all } v < m \end{cases}$$

For a minimizing interior node, the following dual relations apply:

$$\uparrow CN(v) = \begin{cases} 0 & \text{for all } v \leq m \\ \sum_{\textit{all sons } i} \uparrow CN_i(v) & \text{for all } v > m \end{cases}$$

$$\downarrow CN(v) = \begin{cases} 0 & \text{for all } v \geq m \\ \underset{\textit{all sons } i}{MIN} \downarrow CN_i(v) & \text{for all } v < m \end{cases}$$

Figure 1 shows the conspiracy numbers for each node, with $\uparrow CN$ and $\downarrow CN$ merged into one vector. It is worth noting that if $v < w$ then $\uparrow CN(v) \leq \uparrow CN(w)$ and $\downarrow CN(w) \geq \downarrow CN(v)$. Also, given a set of conspirators for changing the value of a node to $v$, $(v \neq m)$, this same set can conspire to change the node to any value between $m$ and $v$.

Since conspiracy numbers represent the difficulty of changing the value of a node, one way they can be used is to judge the accuracy of the root value. A *conspiracy threshold (CT)* is introduced that specifies the minimum number of conspirators required before we consider it unlikely a node can take on that value. A value $v$ is a likely value if $CN(v) < CT$. The range of likely root values is given by $[Vmin, Vmax]$, where $Vmin \leq Vroot \leq Vmax$.

The algorithm continues to search until it has narrowed the range of likely values to just one value. Once all root values but one have been ruled out, we expect further search will not change that value. The higher the threshold, the greater the confidence in the final root value.

Given a range of likely root values, how do we rule out all but one of them? The obvious way is to rule them out one by one, starting with either *Vmax* or *Vmin*. To rule out *Vmin* or *Vmax*, the algorithm tries to increase the corresponding conspiracy number to at least *CT*. This is done by "proving" that a member of the minimal conspiracy set will not conspire with the other members of the set to help change the value of the root node to either *Vmin* or *Vmax*.

During each step of the tree growth procedure, the algorithm must choose to either *rule_out_Vmax* or *rule_out_Vmin*. Faced with these two alternatives, it chooses to attempt to rule out the value which is furthest from *Vroot*. If both are equidistant from the root value, it then arbitrarily chooses to *rule_out_Vmin*. Having made a decision to rule out *Vmax*, for example, a leaf node from the minimal set of conspirators must be found to search one ply deeper (or *expanded*). To find this node, the algorithm descends from the root using the following procedure:

a) at a maximizing node

   Only one successor node must increase its value to *Vmax* for the parent root node to do likewise. The most likely branch is the one requiring the least number of conspirators to increase it to *Vmax*. Computing $CN(Vmax)$ of each successor, choose the successor node requiring the minimum conspirators. If more than one branch has the minimum, arbitrarily choose the left-most one.

b) at a minimizing node

   Here there may be many descendent nodes that have to increase their value to increase the parent node to *Vmax*. Each such branch contains conspirators which together form the set of conspirators to increase this node to *Vmax*. Again the algorithm can choose to traverse any of the appropriate branches and we arbitrarily choose to take the left-most one.

Having reached a leaf node, that node is expanded (i.e. searched one ply deeper). Since each descendent may yield a favourable or unfavourable

assessment, the descendents are ordered according to the results of their evaluation. By putting the more favourable descendents first, this increases the chances that the left-most descendent is the best, justifying the above choices. The minimax value and conspiracy numbers are passed back up the tree, resulting in new numbers along the path from the root to the leaf node.

What is being accomplished by expanding this node? If we are successful at increasing the value of this node to *Vmax*, then the number of conspirators in this set has been decreased by one and therefore other members of the set can be expanded to see if they will conspire successfully. If the value is less than *Vmax* and the expanded node is minimizing, then we may have been successful at increasing the number of conspirators at the root (i.e. increased the minimal set of conspirators). The number of conspirators may have reached *CT*, resulting in a narrowing of the range of likely values at the root. At a maximizing expanded node with a value less than *Vmax*, nothing has been accomplished towards ruling out *Vmax*.

A dual strategy exists for ruling out *Vmin*. This tree growth procedure was McAllester's original proposal. A detailed description of the algorithm can be found in [12].

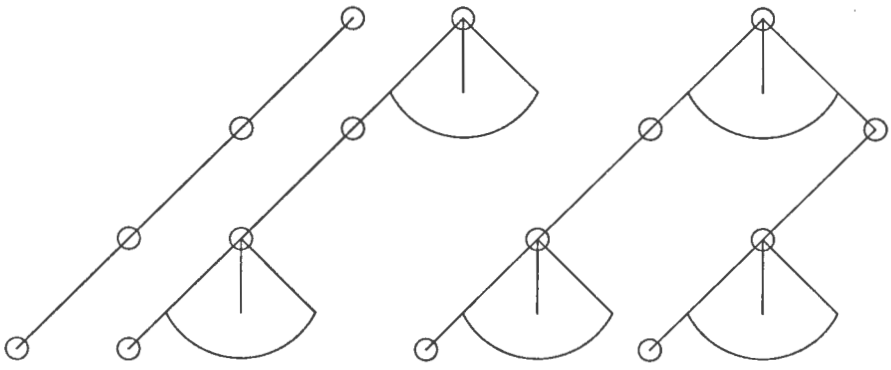
## 3. Search Strategies

The tree growth procedure as originally defined by McAllester is rather simple in that they always explore the left-most son whose value and conspiracy number is within the search range. Figure 3a illustrates this strategy, with circles representing nodes in the tree that are expanded. Note that this is an idealized version of the resulting tree, ignoring branches whose value or conspiracy number is not in range.

Early experiments using conspiracy numbers on trees with randomly generated terminal node values (strongly ordered trees using the method described in [7]) indicated that the program would often descend to ridiculous depths following the left-most branch at every node. As a result, the program usually had enormous difficulty converging to a value [3]. Examination of the program's behaviour suggests that more *breadth* and less *depth* could improve performance.

One way of thinking about minimax trees is that some nodes are *AND* nodes (all sons must be considered) while others are *OR* nodes (at least one son must be considered). This can be seen from the formulas used to compute conspiracy numbers: the MIN operation requires all sons to be considered, while the SUM nodes requires at least one. McAllester's search strategy involves considering the left-most son at OR nodes, even though, in theory, any of the branches could be taken. Of course, since branches are sorted at expanded nodes in order of leaf node value, there is a high probability that the left-most son is best.

Breadth can be added to the search by simply altering the search strategy at OR and AND nodes. One simple modification to the algorithm is to restrict descending from an OR node until all of the sons of that node have been expanded to a fixed minimum depth. The philosophy here is that all the sons can contribute conspirators towards achieving the goal, rather than continually relying on the left-most son. If the left-most son has difficulty achieving the required number of conspirators, McAllester's algorithm will keep trying that branch until success is achieved one way or the other, when in fact another branch may be able to find the requisite conspirators more quickly. For example, if this depth is set to one, then before descending deep into the tree from an interior node,



a) McAllester b) Minimal breadth c) OR round-robin

**Figure 3. Search Strategies**

each son of this node must be expanded. These expansions may achieve (or get closer to) the required conspirators before descending the left-most son. With a depth restriction of 1, this method is called *minimal breadth,* and is illustrated in Figure 3b) with arcs used to emphasize the notion that all sons of that node are considered.

An alternative to always examining the left-most son at an OR node would be to cycle through all the sons, since any of them can contribute conspirators. Every time this node is reached, the algorithm can use a round robin approach to ensure each is treated equally. This strategy is called *OR round robin* and is illustrated in Figure 3c). Note that the middle branch of the root has not been expanded, as would be the case, for example, if its value was out of the range of likely values.

At AND nodes, there is not much to gain by adding breadth since all sons must be considered anyway. However, an algorithm with both *AND round robin* and *OR round robin* strategies would result in a *breadth-first* search of the tree. The breadth-first terminology should be qualified since it is not a conventional breadth-first search; the breadth is only for nodes of the tree whose values lie in the (changing) range of likely values.

An additional enhancement is possible when one considers that the algorithm converges on a value, not a best son. If all the values in the range $[Vmin, Vmax]$ are only achievable from the same son of the root, then further search is unnecessary. The algorithm may not know the value of the tree, but it knows which branch is best. In the context of game trees, this *best move cutoff* would allow the program to stop searching once it has found the best move, not just the best value.

## 4. Experiment Results

The conspiracy numbers algorithm has been implemented in a program that solves tactical chess problems. The most notable enhancement to the algorithm has been the inclusion of *iterative deepening* on the conspiracy threshold [12]. Rather than starting the program with a high threshold, the algorithm begins with a threshold of 2 and having solved that, increases it to 3, and so on. This method has the advantage of being able to stop the program at any time and know the degree of confidence in the root value. Further details on the implementation can be found in [4].

Many experiments with chess programs have used the 300 position test set taken from the book *Win At Chess* [11]. However, the problem set is not difficult and it does not take much effort to solve more than 80% of the problems. This causes problems if the test suite is used for comparing algorithms in that a superior performance translates into solving an additional small percentage of the problems. Conspiracy numbers (the minimal breadth variant) and alpha-beta have been compared using this test suite [12]. Initially alpha-beta out-performs conspiracy numbers, but as the program running time is increased, conspiracy numbers solves more problems than alpha-beta. The effectiveness of alpha-beta is hampered by the exponential growth in the tree, where on average a factor of 5 in computing power is required to search an additional depth or *ply* deeper.

The following experiments were done using a set of problems taken from the *Encyclopedia of MiddleGame Combinations* [2]. The problem set consists of 90 positions, the first 5 problems from each section of the encyclopedia plus a few additional difficult ones. Many of these problems are quite challenging, even for a human! The experiments were run on a Sun 3/75. Each position was run until 300,000 nodes were expanded; roughly 30-40 minutes of CPU time.

Figure 4 illustrates the performance of 4 conspiracy numbers variants: McAllester's original proposal, minimal breadth, breadth-first, and OR round robin (*ORRR*) enhanced with best move cut-offs. The problems correctly solved are plotted against the number of nodes expanded by the algorithm. The results clearly show the advantage of adding some breadth to the search and the effectiveness of the best move cut-off. Two of the round robin variants (AND, OR) are not shown, although their results are comparable to the breadth-first version shown.

A significant and surprising finding was that the average conspiracy threshold required to solve the problems is between 2 and 3. This threshold is usually achieved quite quickly in the search. The additional effort spent in reaching 300,000 expansions results in the answers achieving an average threshold of 9. This confirms the value of iterative deepening.

The heart of the problem with McAllester's version lies in the evaluation function used at leaf nodes. If it were perfect, then taking the left-most son at every node would work and McAllester's original algorithm would be preferred. But perfect information is rarely available. Marsland and Campbell define the trees typically built by game searching programs to be *strongly ordered,* where sons of a node can be ordered so that 70% of the time the best son is known and 90% of the time it is in the top 25% [6]. McAllester's strategy will only consider exploring other sons if the left-most son does not have any appropriate conspirators under it. If the best move is not in the left-most position, it may be some time until

this algorithm finishes exploring all sons left of the best son and then starts exploring the best son. The round robin algorithms give these other sons a greater chance of being explored earlier, thus increasing the chances of finding the best son more quickly.

The goal of McAllester's algorithm and the round robin alternatives is to find the correct value of the root node. However, a frequently occurring case is where one branch is clearly better than all other branches but the algorithm has not converged yet on a single root value. These algorithms will all shift their attention to this one branch, trying to find its correct value. The best move cutoff enhancement would recognize that the best move has been found, but not its value, and increment *CT* at this point causing the algorithm to explore other branches as well. Consequently, this enhancement increases the achievable threshold, allowing greater exploration of other branches in the tree. As a result, the enhancement does quite well, as shown in Figure 4, and can be viewed as a significant improvement to the algorithm.

In comparing the four algorithms, the emphasis was on the number of problems solved by each, and not on the type of problems each is capable of solving. Of the 90 positions, all four algorithms were able to solve 34 common problems, while 32 were not solvable by any. Of the remaining 24, 2 required the depth possible only through McAllester's version, and 4 required the breadth of the round robin schemes. Thus, although OR round robin with best move cutoffs is a compromise between depth and breadth and performs quite well, there will always be problems requiring extremities of breadth or depth.

## 5. Conclusions

How good is the conspiracy numbers algorithm? Experiments reported in [12] indicate conspiracy numbers to be superior to alpha-beta for solving tactical chess problems †. However, it is difficult to make a fair comparison of the two algorithms. The confidence in the answer returned by a fixed depth alpha-beta search is measured by the depth of search.

† One example reported is of a problem with a 15-ply solution. In one hour of compute time, an alpha-beta searcher was only able to search 8-ply. Conspiracy numbers found the solution in 10 minutes.

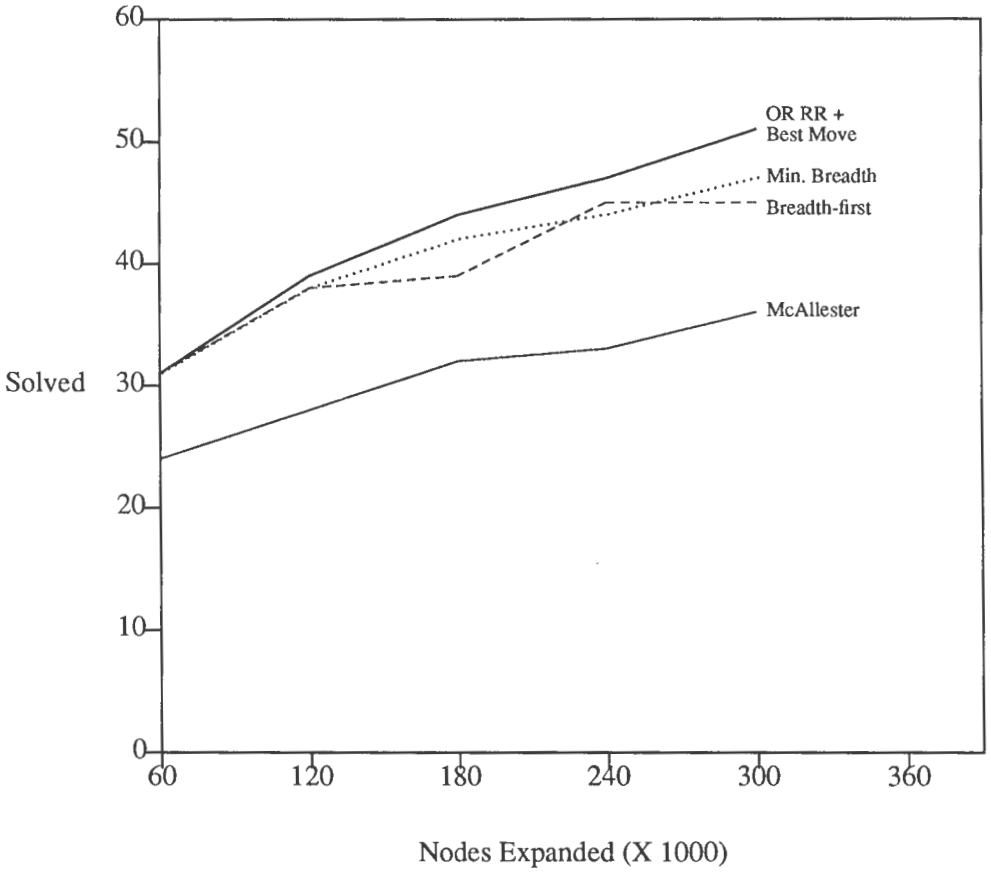



**Figure 4: Search Strategy Comparison.**

Confidence in conspiracy numbers is measured by the conspiracy threshold achieved. In practice, it is not clear what level of confidence is required before one starts to believe the answers, whether one uses alpha-beta or conspiracy numbers.

Results were disappointing when conspiracy numbers was used in a chess program that plays positional as well as tactical chess [12]. It appears that the program does a lot of unproductive searching deep in the tree. The problem seems to be with the stability of the evaluation function. From move-to-move, the assessment of the position would change, if only by a small amount (unlike in the tactical case), resulting in the root having difficulty converging to a value.

Conspiracy numbers is an exciting, new approach to minimax search. It has several important advantages over conventional alpha-beta search approaches, notably the ability to grow trees to variable depth in an application independent manner without any enhancements to leaf node evaluators. However, its disadvantages include the question of algorithm convergence (it is not guaranteed to converge) and the problem of determining a satisfactory termination threshold.

There are several promising areas of research to be explored with conspiracy numbers. First, alpha-beta search enhancements (such as transposition tables) can be used to improve search efficiency. Second, algorithm performance is strongly tied to the accuracy of leaf node evaluations yielding trade-offs of quality (and therefore cost) of evaluation versus tree size. Third, there are interesting possibilities for using conspiracy number information in an alpha-beta searcher. Finally, there are some interesting possibilities for a hybrid algorithm combining B* and conspiracy numbers. B* relies on artificially constructed probabilities to make its search decisions. Conpiracy numbers can be viewed as probabilities that, unlike B*, come directly from the search tree. Combining the two algorithms is an obvious path to explore.

The algorithm is still in its infancy and there is plenty of room for enhancements. The search strategies presented in this paper represent the first attempt at improving the algorithm's performance. Although the 41% increase in the program's ability to solve chess problems is significant, it certainly is not the final word on this topic.


## Acknowledgements

Thanks to Robert Enns for his work on our initial implementation of Conspiracy Numbers. This research was funded by Natural Sciences and Engineering Research Council of Canada grant A8173.


## References


1. H.J. Berliner, The B* Tree Search Algorithm: A Best First Proof Procedure, *Artificial Intelligence 12*, 1 (1979), 23-40.
2. Chess Informator, ed., *Encyclopedia of Middlegame Combinations*, Sahovski Informator, Beograd, 1985.
3. R. Enns and N. Klingbeil, *unpublished experiments*, Department of Computing Science, Univerity of Alberta, 1986.
4. N. Klingbeil, *Implementing Conspiracy Numbers*, M.Sc thesis, Department of Computing Science, Univerity of Alberta, 1987. In preparation.
5. D.E. Knuth and R.W. Moore, An Analysis of Alpha-Beta Pruning, *Artificial Intelligence 6*, (1975), 293-326.
6. T.A. Marsland and M.S. Campbell, Parallel Search of Strongly Ordered Game Trees, *Computing Surveys 14*, (1982), 533-551.
7. T.A. Marsland, A. Reinefeld and J. Schaeffer, Low Overhead Alternatives to SSS*, *Artificial Intelligence 31*, 1 (1987), 185-199.
8. D.A. McAllester, A New Procedure for Growing Mini-Max Trees, Technical Report, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1985.
9. D.A. McAllester, Conspiracy Numbers for Min-Max Search, *Artificial Intelligence*, 1987. In press.
10. A.J. Palay, The B* Tree Search Algorithm - New Results, *Artificial Intelligence 19*, 2 (1982), 145-163.
11. F. Reinfeld, *Win At Chess*, Dover Books, 1945.
12. J. Schaeffer, Conspiracy Numbers, *Artificial Intelligence*, 1987. To appear. Also to appear in *Advances in Computer Chess V*, D. Beal and H. Berliner (ed.), Elsevier Press, 1987.
13. G.C. Stockman, A Minimax Algorithm Better Than Alpha-Beta?, *Artificial Intelligence 12*, 2 (1979), 179-196.

# THE COOPERATIVE APPLICATION OF MULTIPLE NATURAL CONSTRAINTS TO THE MOTION CORRESPONDENCE PROBLEM

Michael R. W. Dawson

The Centre For Advanced Study
In Theoretical Psychology

The University of Alberta

Edmonton, Alberta, Canada

## INTRODUCTION

The human visual system is capable of producing the illusion of movement from a rapid succession of static images. In order to generate this apparent motion, the identity of elements must be maintained over time. Therefore the visual system must be able to identify an element in one position in the first image (Frame I) and another element in a different position in the second image (Frame II) as constituting different glimpses of the same moving element. A system that can correctly make such identifications, called motion correspondence matches, has solved what Ullman (1979) calls the motion correspondence problem. The discovery of the principles exploited by the visual system to solve the motion correspondence problem is an important goal of apparent motion research (Attneave, 1974).

The motion correspondence problem is an example of a problem of underdetermination, because the information input to the system is consistent with more than one interpretation. A solution to the motion correspondence problem might be described as a list of identity matches between Frame I and Frame II elements. However, if there are N different elements (e.g., small dots) in both Frames I and II, then there are N! different lists of matches that could be constructed. Therefore in order to say how the motion correspondence problem is solved, one must describe the procedure used to select the correct member from this set of N! possible lists.

The natural computation approach to vision (e.g., Marr, 1982) attempts to deal with problems of underdetermination in the following manner: The correct solution to a problem is selected from a set of potential solutions by exploiting one or more natural constraints. These constraints are "natural" in the sense that they describe very general properties that are true of the physical world. The solution that is eventually selected by the system must therefore be consistent with the input to the system, and must also be consistent with the properties defined by the natural constraints. Thus a natural computation approach to the motion correspondence problem would require the description of a set of constraining properties such that (i) these properties are general characteristics of physical motion, and (ii) these properties lead to the selection of the correct set of motion correspondence matches.

There have been a number of suggestions about the natural constraints that could be applied to the motion correspondence problem (for a brief review, see Dawson & Pylyshyn, 1988). This paper argues that these different principles are not individually sufficient to account for human perceptions of apparent motion. Instead, different principles must be applied simultaneously. The purpose of this paper is to describe a cooperative network that attempts to accomplish this for multiple constraints. This description proceeds as follows: First, there is a brief review of two previous approaches to the motion correspondence problem. Second, the empirical limitations of these approaches are discussed. Third, the characteristics of a new model of motion correspondence processing are detailed. This model simultaneously applies constraints that (i) minimize the distance moved by elements, (ii) minimize differences in the direction moved by elements, (iii) encourage one-to-one mappings between elements in the two frames of view. Finally, the performance of this model is compared to models that do not apply these constraints at the same time.

## SOME NATURAL CONSTRAINTS ON APPARENT MOTION

### THE MINIMUM DISTANCE PRINCIPLE

Ullman (e.g., 1979) has argued that human motion correspondence processing is best studied using the motion competition paradigm (see also Burt & Sperling, 1981; Chang & Julesz, 1984). In this paradigm, apparent motion is produced by replacing a central Frame I element with two lateral Frame II elements, one to the left and one

to the right of the original position of the Frame I element. The Frame I element is seen to move either to the left or to the right.

One of the strongest predictors of the perceived direction of motion in a competition display is interelement distance (i.e., the distance between an element in Frame I and an element in Frame II). All things being equal, the visual system prefers short correspondence matches (e.g., Burt & Sperling, 1981; Ullman, 1979, Chap. 2). Therefore if motion to the left in a competition display involves a shorter interelement distance than motion to the right, then motion to the left will be preferred. Thus one constraint on solutions to the motion correspondence problem could be a minimum distance or "nearest neighbour" principle -- the visual system attempts to assert a motion correspondence match between a Frame I element and the element nearest to it in Frame II.

Ullman (1979, pp. 114-118) argues that the minimum distance principle is a natural constraint. He has shown that when movement in 3D space is projected onto a 2D surface (i.e., the retina), slow movements are much more probable than fast movements. Thus a preference for short motion correspondence matches (i.e., slow movements) could indicate the exploitation of a property determined by the geometry of the viewing situation. Ullman's minimal mapping theory applies the minimum distance principle, and successfully solves the motion correspondence problem for a wide variety of apparent motion displays.

## THE COVER PRINCIPLE

Proximal visual features usually arise from physical features on the surface of solid objects. Thus visual elements do not appear or disappear from view, except when physical features become occluded by the movement of edges (either due to the rotation of the object in question or to different objects moving through the line of sight). In visual environments such as ours, element occlusions are low probability events. This is because the density of edges is much lower than the density of surfaces (Marr, 1982). This suggests a further constraint on solutions to the motion correspondence problem, which Ullman (1979) calls the cover principle. The cover principle is the claim that Frame I elements do not inexplicably disappear, and that Frame II elements do not inexplicably appear. In Ullman's minimal mapping theory, the cover principle is implemented by
requiring that each Frame I element be matched to at least one Frame II element, and that each Frame II element be matched to at least one Frame I element.

## THE UNIQUENESS CONSTRAINT

Many researchers (e.g., Attneave, 1974; Kolers, 1972; Ullman, 1979) have observed that the human visual system prefers to make one-to-one mappings between frames of view when apparent motion is perceived. In other words, the visual system tends to discourage motion percepts in which one Frame I element splits apart to form two Frame II elements, or in which two Frame I elements fuse together to form a single Frame II element. Some models of stereo correspondence explicitly enforce a preference for one-to-one mappings between frames using what has been called the uniqueness constraint (e.g., Marr & Poggio, 1976).

The uniqueness constraint is also a plausible natural constraint for the motion correspondence problem, because the physical integrity of objects ensures that object splits or fusions during movement should occur with very low probabilities. Indeed, Ullman's revised minimal mapping theory (1979, chap. 3) applies such a constraint by incorporating a penalty for splits and fusions into the "nearest neighbour" cost function that is minimized subject to the cover principle.

## THE RELATIVE VELOCITY CONSTRAINT

Dawson (1986; Dawson & Pylyshyn, 1986) has attempted to solve the motion correspondence problem by applying constraints other than those used by Ullman (1979) in minimal mapping theory. Minimal mapping theory incorporates the assumption that the movement of any one element in an apparent motion display is independent of the movement of any other display element. However, this assumption is not true of human perceptions of apparent motion. Dawson (1987) demonstrated that perceptions of motion competition displays could be changed by surrounding the display with a context that was also set into apparent motion. Motion correspondence matches that were in the same direction as the motion of the context were preferred, even if these matches were longer than other potential matches.

Other researchers have demonstrated related effects on the perceived direction of motion (Chang & Julesz, 1984; Green, 1983; Krumhansl, 1984). Sensitivity to the relative motion of display elements also appears to be an important factor in organizing global motion percepts (e.g., Cutting & Proffitt, 1982; Gogel, 1974; Johansson, 1950; Proffitt & Cutting, 1979, 1980). As well, physiological studies have identified neurological channels that appear to be sensitive to the relative motion of different elements, or to the interaction between different patterns of movement (e.g., Bridgeman, 1972; Frost & Nakayama, 1983; Hammond & Mackay, 1977;

Kaji & Kawabata, 1985). Together, these results all point to a different type of constraint to be applied to the motion correspondence problem. This has been called the relative velocity constraint (e.g., Dawson, 1986). According to this constraint, the visual system should attempt to assign motion correspondence matches such that elements near one another in Frame I should travel in similar directions.

It has been argued that the relative velocity constraint is a natural constraint, because it arises from the physical integrity of objects (Dawson, 1986; Dawson & Pylyshyn, 1988). The integrity of objects places constraints on the transformations that can be applied to the configuration of the parts that comprise these objects. For example, if an object is rigid or near rigid, then one should not be able to move it through space in such a way that a part of it between two other parts at time t is not between the two others at time t+1.

Dawson (1986; Dawson & Pylyshyn, 1986) has developed a discrete relaxation algorithm that applies the relative velocity constraint to the motion correspondence problem. This model is a departure from minimal mapping theory (Ullman, 1979), insofar as the cover principle is explicitly abandoned, and there is no sensitivity to interelement distances. Nevertheless, this model has been shown to solve a wide variety of motion correspondence problems, and therefore has demonstrated the plausibility of the relative velocity constraint.

## LIMITATIONS OF THE NATURAL CONSTRAINTS

The models described above have been relatively successful in providing the correct motion correspondence matches for a variety of apparent motion displays. However, it is still the case that these models do not provide correct solutions (i.e., the solutions generated by the human visual system) for relatively simple displays. This is because the principles exploited in these different models are not independently sufficient constraints on solutions to the motion correspondence problem.

## LIMITATIONS OF MINIMAL MAPPING THEORY

As was described above, Ullman's (1979) minimal mapping theory applies both the minimum distance constraint and the cover principle. Both of these constraints lead to empirical limitations of the model. For example, the results of Dawson (1987) indicate that perceptions of motion competition displays are sensitive to patterns of relative motion. This sensitivity is not incorporated into Ullman's cost function, which only measures the magnitude of potential motion correspondence matches.

The cover principle provides another set of problems for minimal mapping theory. There is reason to believe that the cover principle may not always hold for human perceptions of apparent motion. Specifically, the cover principle may not be applied in cases where there is independent visual evidence for the presence of an occluding edge. Sigman and Rock (1974) demonstrated that the presence of occluding surfaces is an important variable in the perception of apparent motion. If the disappearance of an element cannot be accounted for by the presence of an occluding surface, than an attempt is made to cover the element with a motion correspondence match (i.e., if possible, the disappearance of an element from one image location is dealt with by asserting that the element moved to a new location). If the disappearance of an element can be accounted for by the presence of an occluding surface, then a motion correspondence match is not computed for the element (i.e., it is not covered, and is therefore not seen to move).

The evidence above suggests that the cover principle might only hold in the absence of visual evidence for occluding edges. However, this too is not correct. Consider the motion competition display depicted in Figure 1. According to the cover principle, both Frame II elements must be covered by a motion correspondence match, and therefore minimal mapping theory is required to generate the solution depicted in Figure 1(a). However, it is relatively easy to create a situation in which human observers see the motion depicted in Figure 1(b): one element is seen to move in one direction, while another element suddenly appears in view. Note that this percept violates the cover principle as implemented in minimal mapping theory, and that this occurs in a display where no visual evidence for occluding edges is present.

## LIMITATIONS ON THE RELATIVE VELOCITY CONSTRAINT

Dawson's (e.g., 1986) discrete relaxation labeling model also has difficulty accounting for some basic empirical facts about apparent motion perception. This is because the only information used in this model to constrain solutions to the problem is relative velocity information.

For instance, Dawson (1987) reports that while perceptions of motion competition displays are affected by the presence of a moving context, subjects are still sensitive to interelement distance. Indeed, if this were not the case, then the threshold measurements used to contradict Ullman's (1979) independence hypothesis could not have been obtained. However, sensitivity to interelement distances was not built into Dawson's algorithm.

A more striking example of the type of difficulty facing this model is revealed by considering a general limitation on relative velocity information. Any display that has only one Frame I element does not have any relative velocity information, because the relative velocity constraint requires maximizing the similarity of motion direction of at least two Frame I elements. Thus Dawson's (e.g., 1986) model cannot compute the correct motion correspondence matches for displays like the competition display depicted in Figure 1, or for the simplest apparent motion case in which there is only one element in both frames of view.

## THE SIMULTANEOUS APPLICATION OF CONSTRAINTS

Underlying the discussion in the previous section are two basic points: First, constraints like the minimum distance principle and the relative velocity constraint are not individually sufficient to account for all solutions to the motion correspondence problem. Second, the cover principle provides numerous difficulties for a theory of motion correspondence processing. It would appear that an adequate theory of motion correspondence processing would require that (i) the cover principle be abandoned, and (ii) both the minimum distance and the relative velocity constraints be applied cooperatively. This section describes a particular model that meets these requirements.

### INITIAL REPRESENTATION

The model takes as input the (x, y) coordinates of elements in Frames I and II of an apparent motion display. On the basis of this information, a local, parallel network of connected processing units is constructed. Each of the units in the network represents one of the possible motion correspondence matches for the input display. Thus if the display has N elements in Frame I, and M elements in Frame II, the network that is constructed has N * M processing units. Each of these units can have an activation level ranging in value from 0.00 to 1.00. The activation state of the set of units in the network at time t is represented as the column vector $\underline{a}(t)$.

At the start of processing, each unit is assigned an activation level of 0.50. At the end of processing, the activation level of each unit will be either 1.00 (indicating that the motion correspondence match represented by the unit is included in the solution to the problem) or 0.00 (indicating that the motion correspondence match represented by the unit is not included in the solution to the problem). Note that with this type of representation, the cover principle is not enforced. Any of the units can be driven to 0.00 at the end of processing, which would not be permitted under the cover principle.

The processing units are linked to one another by connections of varying strengths. Connections can be excitatory or inhibitory, and are symmetric (namely, the strength of the connection from unit 'a' to unit 'b' is equal to the strength of the connection from unit 'b' to unit 'a'). Connection strengths are defined a priori in the model by applying the uniqueness, the proximity, and the relative velocity constraints. In principle, the network is "massively parallel", for it is possible for each of its units to be connected to every other unit. As a result, the connections among the units can be represented by the square matrix W, which has N * M rows and columns. In practice, however, some units may not be connected to one another (i.e., the connection between them has a strength of 0.00). This is because the constraints used to define the connections are applied locally, as is described below.

### DEFINING CONNECTIONS USING THE UNIQUENESS CONSTRAINT

Let $\underline{U}$ be a square matrix that represents the connections among units defined by only applying the uniqueness constraint. Recall that this constraint is used to inhibit the splitting or fusing of display elements during perceived movement. The matrix $\underline{U}$ is constructed in the following manner: all units that represent motion correspondence matches emanating from the same Frame I element are given mutually inhibitory connections of strength equal to -1.00. For instance, if units i and j represent matches originating from the same display element, then $U_{ij}$ and $U_{ji}$ are both set to -1.00. As well, all units that represent motion correspondence matches terminating at the same Frame II element are given mutually inhibitory connections of strength equal to -1.00. All other connections in $\underline{U}$ are given a strength of 0.00.

### DEFINING CONNECTIONS USING THE PROXIMITY CONSTRAINT

Let $\underline{P}$ be a square matrix that represents the connections among units defined by only applying the proximity constraint. Recall that this constraint is used to encourage the selection of shorter motion correspondence matches. $\underline{P}$ is defined by creating a single excitatory connection between each unit and itself; as a result $\underline{P}$ is a diagonal matrix. The strength of the connection ranges from 0.00 to 1.00, and is a function of the length of the motion correspondence match represented by the unit. The shorter this match is, the stronger is the excitatory connection. Following Ullman's (1979) analysis of the proximity constraint, an exponential function is used to map match length onto

connection strength: $P_{ii}$ = EXP ( 1.00 * scale * length of motion correspondence match i), where the "scale" variable is simply used to sharpen or flatten the differences between connection strengths.

## DEFINING CONNECTIONS USING THE RELATIVE VELOCITY CONSTRAINT

Let V be a square matrix that represents the connections among units defined by only applying the relative velocity constraint. Recall that this constraint is used to ensure that elements near one another in Frame I are seen to move with similar velocities. V is constructed in the following manner: First, neighbourhood relationships among matches are determined. Two matches are neighbours if and only if they emanate from different Frame I elements which are sufficiently near one another to fall within a specified computational radius (equal to 5.00 distance units in the current implementation). Second, the relative velocities for all pairs of neighbouring matches are computed by taking the Euclidean distance between the endpoints of the matches after being centered to a common origin. This distance is used to create the connection between the units representing the neighbouring matches. If matches i and j are neighours, then $V_{ij}$ = 2.0 - EXP (-1.00 * scale * Euclidean distance between matches i and j). Thus it is possible for relative velocity connections to be inhibitory or excitatory, ranging in possible strength from -1.00 to 1.00.

## DEFINING THE OVERALL CONNECTIVITY MATRIX

The square matrix W represents the set of connections among units defined on the basis of the simultaneous application of the three constraints defined above. W is simply defined as the scaled sum of the three individual connectivity matrices: W = scale * ( U + P + V). The main effect of the scale that is used is simply to increase or decrease the number of iterations required for the network to converge. In the examples described below, the scale value used was 0.02.

Note that this approach to defining the overall set of connections allows future constraints to be incorporated into the model. For instance, some current work on apparent motion is focusing upon the role that geometric and topological constraints may play in the motion correspondence process (e.g., Chen, 1985; Green, 1986). These types of contraints could be used to define new connectivity matrices, which in turn could be simply added into W.

## UPDATING THE ACTIVITY OF THE UNITS

The activity vector a is iteratively updated by using W as a source of positive feedback (e.g., Anderson, Silverstein, Ritz, & Jones, 1977). The updating equation that is applied is:

$$a(t+1) = a(t) + W * a(t)$$

In addition, the values in a are bounded by a simple cutoff rule -- they are restricted to range from 0.00 to 1.00. A similar cutoff rule is applied in Anderson et al's "brainstate-in-a-box" model of categorical perception. A mathematical property of the "brainstate-in-a-box" model (which is a consequence of the symmetric structure of W) is that eventually a is driven into a state in which all of the units are at one of the cutoff activity levels (i.e., either 0.00 or 1.00 in the motion correspondence model described above). Thus this iterative application of positive feedback will result in a converging to a state that represents a solution to the motion correspondence problem.

## PERFORMANCE OF THE MODEL

The model is capable of generating the correct set of motion correspondence matches for a variety of apparent motion displays. Figure 2 depicts some example solutions for displays involving various rotations and translations of elements. These solutions are "correct" in the sense that they are generated by the human visual system. These solutions are also generated by minimal mapping theory (Ullman, 1979) and by Dawson's (1986) discrete relaxation model.

The model also demonstrates significant improvements in performance over Dawson's (1986) discrete relaxation model. The current model computes the correct solutions for displays that this previous model could either not process, or for which it generated incorrect solutions. These results are depicted in Figure 3. In all cases, this improvement in performance can be attributed to the fact that proximity information is available in addition to relative velocity information. The model also deals with displays for which minimal mapping theory generates incorrect solutions (see Figure 4). One reason for this is sensitivity to relative velocity information. Another reason for this is that the cover principle is not operating in the multiple constraint model.

The model is also capable of generating correct solutions to symmetric patterns of apparent motion (see Figure 5). Attneave (1974) noted that human perceptions of these types of displays varied predictably as a function of whether the number of Frame I elements was odd or even. Attneave suggested that this empirical finding would be very difficult for a data-driven motion correspondence

algorithm to emulate. It would appear that this result can be accounted for by the simultaneous application of three different natural constraints on apparent motion.

## REFERENCES


Anderson, J.A., Silverstein, J., Ritz, S., & Jones, R. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. Psychological Review, 84, 413-451.

Attneave, F. (1974). Apparent movement and the what-where connection. Psychologia, 27, 108-120.

Bridgeman, B. (1972). Visual receptive fields sensitive to absolute and relative motion during tracking. Science, 178, 1106-1108.

Burt, P., & Sperling, G. (1981). Time, distance, and feature trade-offs in visual apparent motion. Psychological Review, 88, 171-195.

Chang, J., & Julesz, B. (1984). Cooperative phenomena in apparent movement perception of random-dot cinematograms. Vision Research, 24, 1781-1788.

Chen, L. (1985). Topological structure in the perception of apparent motion. Perception, 14, 197-208.

Cutting, J., & Proffitt, D. (1982). The minimum principle and the perception of absolute, common, and relative motion. Cognitive Psychology, 14, 211-246.

Dawson, M. (1986). Using relative velocity as a natural constraint for the motion correspondence problem. U.W.O. Centre For Cognitive Science Technical Memorandum No. 27.

Dawson, M. (1987). Moving contexts do affect the perceived direction of apparent motion in motion competition displays. Vision Research, 27, 799-809.

Dawson, M., & Pylyshyn, Z. (1986). Using relative velocity information to constrain the motion correspondence problem: Psychophysical data and a computational model. Proceedings of the Sixth Canadian Conference on Artifical Intelligence. Toronto: John Wiley & Sons.

Dawson, M., & Pylyshyn, Z. (1988). Natural constraints on apparent motion. In Z. Pylyshyn (Ed.) Computational Processes In Human Vision: An Interdisciplinary Perspective. Norwood, N.J.: Ablex, in press.

Frost, B., & Nakayama, K. (1983). Single visual neurons code opposing motion independent of direction. Science, 220, 744-745.

Gogel, W. (1974). The adjacency principle in visual perception. Quarterly Journal Of Experimental Psychology, 26, 425-437.

Green, M. (1983). Inhibition and facilitation of apparent motion by real motion. Vision Research, 23, 861-865.

Green, M. (1986) What determines correspondence strength in apparent motion? Vision Research, 26, 599-607.

Hammond, P., & MacKay, D. (1977). Differential responsiveness of simple and complex cells in cat striate cortex to visual texture. Exploratory Brain Research, 30, 275-296.

Johansson, G. (1950). Configurations In Event Perception. Uppsala, Sweden: Almqvist and Wiksell.

Kaji, S., & Kawabata, N. (1985). Neural interactions of two moving patterns in the direction and orientation domain in the complex cells of cat's visual cortex. Vision Research, 25, 749-753.

Kolers, P. (1972). Aspects Of Motion Perception. Oxford: Pergammon Press.

Krumhansl, C. (1984). Independent processing of visual form and motion. Perception, 13, 535-546.

Marr, D. (1982). Vision. San Francisco: W.H. Freeman.

Marr, D., & Poggio, T. (1976). Cooperative computation of stereo disparity. Science, 194, 283-287.

Proffitt, D., & Cutting, J. (1979). Perceiving the centroid of configurations on a rolling wheel. Perception & Psychophysics, 25, 389-398.

Proffitt, D., & Cutting, J. (1980). An invariant for wheel-generated motions and the logic of its determination. Perception, 9, 435-449.

Sigman, E., & Rock, I. (1974). Stroboscopic movement based on perceptual intelligence. Perception, 3, 9-28.

Ullman, S. (1979). The Visual Interpretation Of Motion. Cambridge, Mass.: MIT Press.

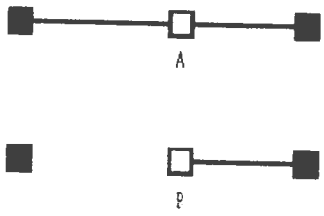



Figure 1.

Solutions to a motion correspondence display, where empty squares indicate the positions of Frame I elements, and filled squares indicate the positions of Frame II elements. (a) Solution required by enforcing the cover principle in minimal mapping theory. (b) Human perceptions of the display.

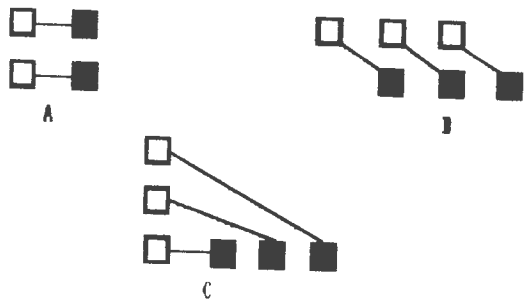



Figure 2.

Performance of the model with the exponential scaling parameters set to 0.075. In all cases, these solutions are generated by the human visual system. (a) Translation. (b) Ternus translation. (c) Translation with rotation.

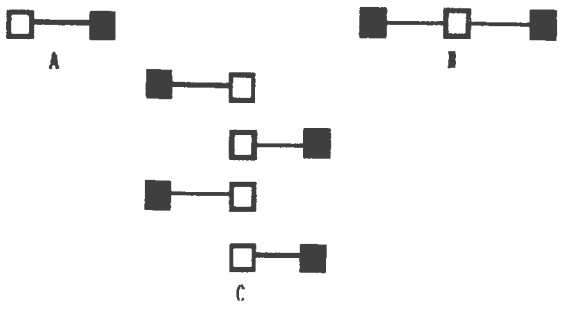



Figure 3.

Improvements of the current model over Dawson's (1986) discrete model. The previous model could not generate any of these correct solutions. (a) Translation of a single element. (b) The split of a single element. (c) Shearing movement.

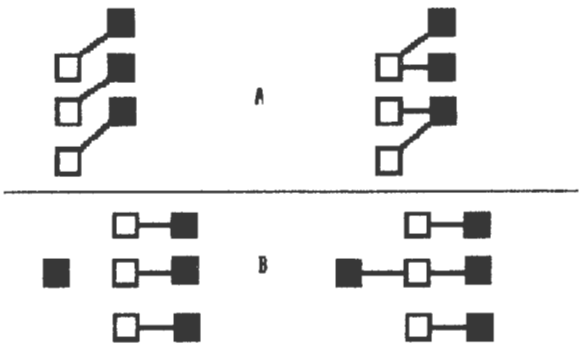



Figure 4.

Improvements of the current model over minimal mapping theory. The current model generates the correct solutions on the left; minimal mapping theory generates the incorrect solutions on the right.

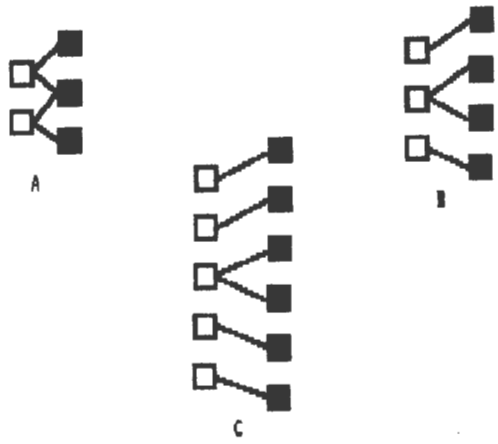



Figure 5.

The current model generates the "minimal cover" solution for symmetric patterns of motion, which is also generated by the human visual system. Note that this is done without enforcing the cover principle.

AUTHOR'S NOTES

The research reported in this paper was supported by NSERC operating grant No. A2038 awarded to the author.
Correspondence should be addressed to Dr. Michael Dawson, Centre For Advanced Study In Theoretical Psychology, The University of Alberta, Edmonton, Alberta, Canada T6G 2E9.

# Structure Recognition by Connectionist Relaxation: Formal Analysis

Paul Cooper
Department of Computer Science
University of Rochester
Rochester, NY 14627
cooper@cs.rochester.edu

## Abstract

A formal description is given of a connectionist implementation of discrete relaxation for labelled graph matching. The application is fast parallel indexing from structure descriptions. The network is limited by complexity considerations to the detection and propagation of unary and binary consistency constraints.

The convergence of the algorithm is proven. The desired behavior of the algorithm is formally specified, and the fact that the network correctly computes this is proved. Explicit and exact space and time resource requirements are developed.

Keywords: connectionist network, attributed graph matching, discrete relaxation, constraint propagation

## 1. Introduction

This paper provides a formal description and analysis of a connectionist network that matches labelled graphs in a limited way. The network is a specialized and massively parallel implementation of discrete relaxation or constraint satisfaction which propagates unary and binary constraints [Mackworth 1977, Hummel and Zucker 1983].

The network was designed for the indexing task in recognizing structured objects [Shapiro and Haralick 1981, Feldman 1985a]. The utility of discrete relaxation in such vision applications is well known [Waltz 1975, Davis and Rosenfeld 1981, Hinton 1977, Kitchen and Rosenfeld 1979]. The experimental performance of the network described here was investigated in earlier work involving the recognition of Tinker Toy objects in images [Cooper and Hollbach 1987].

In this paper, the complexity and correctness characteristics of the network are demonstrated. Following a detailed description of the network design and operation, the convergence of the algorithm is proven. The desired behavior of the algorithm during relaxation is formally specified, then it is demonstrated that the network conforms to this desired behavior. Complexity analysis shows that the network requires $O(N^4)$ space, where N is the number of nodes in the graphs to be matched. Worst case time complexity is linear in the number of nodes and arcs, although convergence in a small constant number of time steps is more likely. The feasibility of the algorithm given its resource requirements is discussed.

## 2. Network Design

In this section a connectionist network design is given that can represent two labelled graphs, and all possible matches of the graphs to each other. As in all connectionist implementations, the lack of an interpreter suggests the adoption of the unit/value principle - every value which can play a role in the computation is represented explicitly as a unit. Of course, the network must be of finite size. Therefore, a size bound N on the number of nodes in the graphs is required. Aside from the this size limitation, the network will be universal for any pairwise graph matching problem.

Units representing the nodes of the graphs are designated $p_i$. A unit representing an arc between nodes $p_e$ and $p_f$ is designated $r_{ef}$. Only the nodes are labelled, and the label associated with $p_i$ is designated $l_i$. The $l_i$ are the output of the $p_i$, and are taken to be integers between 1 and 10. Superscripts are sometimes used to refer to particular graphs, ie. $G^A$, $l_i^A$, $r_{ef}^A$.

Every possible pairing of nodes and arcs is represented explicitly by a unit. The pairing of node $p_i^A$ with node $p_j^B$ is a unit designated $m_{ij}$. The units which represent arc pairings are designated $n_{ef,gh}$ (for the pairing of arcs $r_{ef}^A$ and $r_{gh}^B$). These units are also refered to as *node matching units* and *arc matching units*, respectively.

These sets of units are best conceived of in terms of 2 arrays, as in Figure 1. The left-hand array represents the nodes of both graphs, and all possible node pairings.

The right-hand array represents the arcs of the graphs, and all possible arc pairings. If exactly one unit in each row and column of each array were active, this would represent an isomorphism between the 2 graphs.

The sets of units are connected in three logically independent ways. All connections are symmetric, with weights of unity.

*Graph Input Connections*

- all $p_i^A$ connect to all $m_{ij}$ for all j
- all $p_j^B$ connect to all $m_{ij}$ for all i
- all $r_{ef}^A$ connect to all $n_{ef,gh}$ for all gh
- all $r_{gh}^B$ connect to all $n_{ef,gh}$ for all ef

These connections allow the representation of the input graphs to interact with the matchings units.

*Winner-take-all Connections*

Each $m_{ij}$ and $n_{ef,gh}$ participates in a winner-take-all competition (WTA) against the other units in its row and column. There are a variety of implementations of WTA nets, any of which is suitable. For simplicity in describing unit functions later, a WTA network is used in which each unit compares itself to the maximum of all the others.

- each $m_{ij}$ is connected to $m_{ik}$ for all k and $m_{kj}$ for all k
- each $n_{ef,gh}$ is connected to $n_{ef,ab}$ for all ab and $n_{ab,gh}$ for all ab

*Cross-array Connections*

Because the arcs are undirectional, each arc pairing $n_{ef,gh}$ is consistent with four node pairings $m_{ij}$. A set of connections from one array to the other reflect this fact.

- each $n_{ef,gh}$ connects to exactly 4 $m_{ij}$ as follows:
  - (i) i=e & j=g
  - (ii) i=e & j=h
  - (iii) i=f & j=g
  - (iv) i=f & j=h

With the $p_i^A$ designated by numerals, the $p_j^B$ designated by letters, and the $r_{ef}^A$ and $r_{gh}^B$ designated correspondingly, Figure 1 demonstrates these cross-array connections.

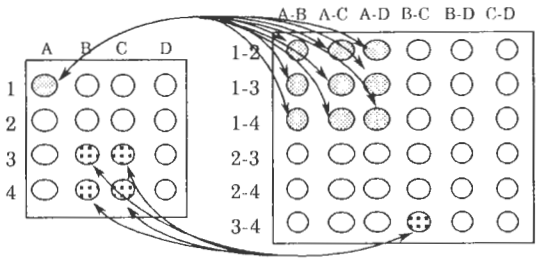



Figure 1: node matching array, arc matching array, and example constraint propagation links

## 3. Network Operation

Now that the configuration of the network is established, the relaxation process which matches the graphs is described. There are three major aspects to the computation. First, the network is initialized to compute the particular problem instance. Second, local uniquenesses are found in both graphs and matched. (These are referred to as match "seeds"). Finally, matches are propagated as far as possible.

The relaxation itself is a synchronous computation. Each time step consists of 2 substeps. The first substep is the constraint propagation phase. In this substep, node matching units send messages to consistent arc matching units, and vice versa. The second substep is the winner-take-all contest. Units which win the contest signify matches between particular nodes or particular arcs. Losers signify incorrect matches; these units eventually turn off. As the computation proceeds, the high ("matched") potential propagates bach and forth, growing larger and larger locally consistent matches from the seed matches. The match terminates when the largest possible number of locally consistent matches have been determined.

### 3.1 Internal Unit Structure

The definitions of Feldman and Ballard [1982] are adopted as the formal basis for the connectionist units. Thus, roughly, units have inputs, state, potential or activation, and output.

Graph units, including the $p_i^A$ $p_j^B$ $r_{ef}^A$ and $r_{gh}^B$, are very simple. They each have a set of discrete states {*off*, *on*}. The potential corresponding to state *off* is 0 for the $p_i$ and $r_{ef}$, while the *on* potential is 1 for the $r_{ef}$ and equal to $l_i$ for the $p_i$. Unit output is equal to unit potential. These units are locked in one state (and potential) or the other for the duration of the computation. Which units are active depends on the particular instance of $G^A$ and $G^B$.

The matching units are more complex. Each $u_i$ (where $u_i$ is an $m_{ef}$ or $n_{ef,gh}$) is a tuple $<S_i, X_i, D_i>$ consisting of

- $S_i$ - the unit's state, where $S_i \in$ { *off*, *contending*, *bound* }
- $X_i$ - the unit's potential (equal to its output), where $X_i \in \{0, 1, 2, \ldots 20\}$
- $D_i$ - a set of data inputs to the unit

The data inputs $D_i$ are conveniently divided into sites, with a vector of inputs at each site. Thus:

$$D_i = \{ D_{graph}, D_{WTA}, D_{cross\text{-}array} \}$$

corresponding to the 3 types of connections attached to each matching unit. Note that the same subscript is used to designate the unit, state, and potential. Thus,

the state of a typical node matching unit $m_{ij}$ is designated simply $S_{ij}$.

### 3.2 Unit Functions

The state, potential and output of a unit are generally functions of the previous state, previous potential, and current input. This computation, the unit function, is now specified for the matching units. (The other units do not change state, so their unit function is trivial). The same unit function is computed by each matching unit $u_{ij}$ independently.

The unit functions are different during initialization and during relaxation. The unit function for initialization is given first. For each unit $u_{ij}$, 2 specific data inputs of $D_{graph}$ are identified and designated as $d_i$ and $d_j$. These inputs are the outputs from the graph units in the row and column of the unit $u_{ij}$. With these conventions, the unit function for a node matching unit is:

if initialization step then
  if $d_i = d_j$ then
    $S_{ij} \leftarrow$ *contending*
    $X_{ij} \leftarrow 1$
  else
    $S_{ij} \leftarrow$ *off*
    $X_{ij} \leftarrow 0$

Note that while the activation functions for the node and arc matching units are the same, the effect is different for the two kinds of units. In the node matching array, all units representing matches between nodes *with the same label* are initialized to the *contending* state. In the arc matching array, all possible arc matches (corresponding to arcs which actually exist in this problem instance) are activated.

Because the network is universal for any graphs up to size N, unless the input graphs are very dense, relatively few of the available units are active after the initialization step.

The unit function for the matching units during a regular processing step is more complex. Recall that the computation is synchronized, with two sub-steps per time step. The unit function must therefore access a time state variable as well as previous state and a subset of current inputs. The unit function for the node matching units is:

if $S_i$ = *contending*
  if winner-take-all substep then
    if $X_i > \max(D_{WTA})$ then
      $S_i \leftarrow$ *bound*
      $X_i \leftarrow \beta \leftarrow 5$
    else
      if $X_i < \max(D_{WTA})$ then
        $S_i \leftarrow$ *off*
        $X_i \leftarrow 0$
      else
        $S_i \leftarrow$ *contending*
        $X_i \leftarrow 1$
  else if propagation substep then
    $X_i \leftarrow \Sigma d$ where $d \in D_{cross\text{-}array}$

The arc matching units have the same unit function as the node matching units, except the potentials corresponding to the states *contending* and *bound* are 1 and 0 instead of $\beta$ and 1. These potentials were chosen for convenience in the proofs which follow in the Appendix.

## 4. Correctness

To show the network correctly computes what is desired, it is necessary to first show that it converges. The nature of the units and the way they change state constrain the network so it has a convex state space, as the following proof demonstrates.

**Theorem 1:** The Network Converges

**Proof:**

Clearly, only the matching units $m_{ij}$ and $n_{ef,gh}$ are relevant. At the completion of each time step, each such unit $u_i$ has state $s_i \in$ { *off, contending, bound* }. Define a global goodness measure or total for the network $T = \Sigma_i T_i$ as the sum of the individual unit goodnesses, where

$T_i = 1$ if $S_i$ = *bound*
$T_i = 0$ if $S_i$ = *off*
$T_i = -1$ if $S_i$ = *contending*

Now, units in state *bound* and state *off* cannot change state. Therefore, if a unit $u_i$ changes state, $T_i$ must increase. Therefore, T is a monotonically increasing function. But the number of units is a finite constant, so T can only increase a finite number of times. Therefore, the network must eventually converge to a stable state.

**QED**

The next step is to show that once the network has converged, it is in the desired state. One way to do this is to show that at each time step the algorithm does what is expected, and that it can do no more when it converges. Therefore, the proof of correctness is derived from understanding the nature of the algorithm, formally specifying this, and showing that the network state change conforms to the goal.

The algorithm is designed to detect unique matches between two labelled graphs, and to propagate these matches as far as symmetry and consistency allow. Interestingly, the criterion "as far as symmetry and consistency allows" is a function of time. Following

this intuitive structure, the proof proceeds by defining the following:

unique unary (node) matches
unique binary (arc) matches
an arc matchable by propagation at time t
a node matchable by propagation at time t
matchable graph elements

Formal yet unsurprising definitions for these designations are given in the Appendix. For example, a unique unary node match occurs when there exists a node in the candidate object graph with a label that is unique to that graph, there exists a node in the target model graph with a label unique to that graph, and the two unique labels agree. The key to defining the nodes and arcs matchable by propagation at time t is formalizing the intuitively straightforward concepts of consistency and symmetry. Node matches are consistent with arc matches if each node is one end of the matched arc.

That the network can correctly execute each necessary substep is shown in the Appendix in the following Lemmas:

**Lemma 1**: The network detects unique node matches
**Lemma 2**: The network detects unique arc matches.
**Lemma 3**: The network assigns a unique consistent match to arcs matchable at time t by time t+1, (if the computation has not already terminated).
**Lemma 4**: The network assigns a unique and consistent match to nodes matchable at time t by time t+1, (if the computation has not already terminated).

The proof of each Lemma follows in a straightforward way from the appropriate definitions as above, and the unit functions. With these Lemmas available it is a simple matter to show the following.

**Theorem 2**: The network assigns a unique consistent match to every matchable graph element.

**Proof**: (Contradiction)
Assume the network converges in K steps, so $T^K = T^{K-1}$. Assume that there exists a matchable graph element which is not assigned a match. By Lemmas 1 and 2, the network assigns matches for the unary and binary local uniquenesses, so it cannot be these. By Lemmas 3 and 4, the network matches those elements which are matchable by propagation at time $t < K$. Therefore, if a matchable graph element exists that is not matched, it must be matchable by propagation for some $t > K-1$, say $t=C$. To reach this state at $t=C$ (which is different from the state at $t=K-1$), the network must change state during every timestep from $t=K-1$ to $t=C$. But $T^K = T^{K-1}$, and the network does not change state after $t=K-1$.
**QED**

### 3.1 Correctness: Isomorphisms and Near Matches

Suppose the graphs are the same size. If all the nodes and arcs of $G^A$ are matchable (with matchable defined as in the Appendix), the matching computed by the network is the unique isomorphism.

But suppose the sizes are equal, yet more than one isomorphism exists between the graphs. In this case, not all the graph elements are matchable. For this to be true, there must exist some kind of local symmetry in the graph, beyond which the constraint of consistent matching cannot propagate. In this situation, all (ambiguous) isomorphic matches between the graphs are represented in the final state of the network, by competing matching units whose state has stabilized at *contending*. Clearly, if necessary, case analysis could be used to further disambiguate. The worst case for symmetric ambiguity occurs when there are no match seeds. Then, no matchable local uniquenesses exist in the graphs at all. This degenerate situation is not likely to occur in a visual recognition problem.

Finally, we must consider what happens if the graphs are completely dissimilar. That is, either their sizes differ, or their structure differs, or both. It is still true that the network computes all seed matches, and propagates each match as far as consistently possible. The final matching has the largest possible locally isomorphic subgraphs matched, with each containing a seed match.

## 4. Complexity

### 4.1 Space Complexity

In analyzing the complexity of a connectionist network, the most important characteristic is the number of units required for the computation. But the fixed size of the universal network is clear:

number of node units $p_i = N$
number of node matching units $m_{ij} = N^2$
number of arc units $r_{ef} = \frac{1}{2}N(N-1)$
number of arc matchings units $n_{ef,gh} = [\frac{1}{2}N(N-1)]^2$

So the total number of units is $O(N^4)$. The number of connections is also $O(N^4)$, if a reasonably efficient implementation is used for the winner-take-all networks. The $O(N^2)$ fan-out can be circumvented by using output trees.

### 4.2 Time Complexity

Suppose we designate the number of nodes plus the number of arcs in a graph A as $SIZE^A$, and similarly for graph B.

**Theorem 3**: The network converges in K time steps, where $K <= \min(SIZE^A, SIZE^B)$.

**Proof:**

Consider the smaller of the two graphs if they have unequal sizes, and suppose without loss of generality that it is graph A. The maximum time it could take for a seed match to propagate to some matchable graph element is $SIZE^A$ steps., It cannot take longer because no graph element can be matched twice (ie. cycles cannot be traversed), and propagation only occurs when matches are being made. Finally, while graph B might have a longer path, propagation can only occur for the shorter length of time that the elements of A are being matched.

**QED**

One of the interesting aspects of this result is what it is not. That is, one might expect that the longest propagation path would be equal in length to the longest path between any 2 nodes in the graph (the diameter of the graph). This is not true, however, because propagation depends not only upon the path taken, but also upon *when* the propagation steps occur and whether or not symmetries are present at a given time. Sometimes a node becomes matchable by a very roundabout propagation path.

Of course, the occurence of the worst case is highly unlikely. With this parallel implementation, it is much more reasonable to suppose that the algorithm converges in at most a small constant number of time steps. Experiments [Cooper and Hollbach 1987] confirm this.

### 4.3 Network Complexity: Feasibility

The feasibility of an algorithm can judged from its complexity. The network is $N^4$ in units and connections, and guaranteed to require less than $SIZE^A$ steps. For small N the space requirement is reasonable, especially if the number of required units is compared to that available in the human brain. For larger N, an obvious way to reduce space complexity would be to use hierarchical structure representations.

The feasibility of the design has also been demonstrated constructively, in the experiments of Cooper and Hollbach [1987]. Even for experiments with N about 10, it would be quite straight-forward to implement a 10,000 unit network. Such a network implemented on a parallel processor such as the Butterfly can even be expected to be simulated in reasonable time [Fanty 1986].

## 5. Discussion and Conclusions

The central contribution of this paper has been to describe a connectionist implementation of discrete relaxation for labelled graph matching, and prove the correctness of its operation. In general, discrete relaxation is well understood [Mackworth 1977, Mackworth and Freuder 1985, Hummel and Zucker 1983, Mohr and Henderson 1986] and its utility has been demonstrated, especially in vision applications [Waltz 1975, Davis and Rosenfeld 1981, Hinton 1977]. The structure recognition or labelled graph matching application for relaxation has also been explored [Kitchen and Rosenfeld 1979], but not with a parallel implementation. The specifics of the connectionist implementation illuminate some interesting issues.

The massively parallel implementation of discrete relaxation represents one end of the time/space complexity spectrum. Advantageously, explicit and precise time and space complexities are easy to derive from the design, and have been given above. As with most parallel implementations, space costs have been traded off in favor of execution time, yet the space complexity remains feasible. The explicit nature of the implementation also makes extending the algorithm in parallel trivial, and its resource requirements remain obvious.

It is also interesting to observe how working within the connectionist paradigm constrains a design. The requirement that there be no interpreter means each relevant value must be represented by a unit. This suggests in the context of the graph matching problem, that representing matchings for ternary or higher relations would become prohibitively expensive. Furthermore, bandwidth limitations on the messages the units can send each other (inspired by neural communication bandwidth limitations) restrict the character of what can be computed with unary and binary constraints. For example, it would be difficult to modify the algorithm so that more than one unary predicate could be used (eg. [Kitchen and Rosenfeld 1979]). Likewise, bandwidth limitations restrict the potential utility of the unary predicate itself. Interestingly, the overall resource requirements of the algorithm suggest an upper bound on the number of parts a structure representation can have before a more efficient representation such as a hierarchy is required. Structure representations with more than a small constant number of parts, like in the small double digits, would become unwieldy.

The paper contains somewhat stronger and more specific results about what can be matched than are

typically found (cf. the definitions of matchable by propagation for nodes and arcs). These results were obtainable because the exact nature of the algorithm was very tightly constrained (eg. only unary and binary constraints, with propagation occurring only under very specific conditions).

The particular performance characteristics of the algorithm suit the task problem - fast indexing from structure descriptions - very well. With this implementation, the theoretical speed one expects from massively parallel designs is available. Complete recognition (with verification) requires that a complete correspondence between object and model be established, and this algorithm is not guaranteed to do so. But it can be used to quickly reject large numbers of model candidates, so later more expensive processes can be more confidently and efficiently applied to the remaining candidates.

There are two obvious directions to develop the work. First, one could attempt to generalize the algorithm beyond the application for which it is designed. In recent work, Swain and Cooper [1988] addressed this issue by developing a parallel implementation of the classic and generally applicable constraint satisfaction algorithms of Mackworth [1977] and Hummel and Zucker [1983]. In Swain and Cooper [1988], correctness characteristics are inherited from the original formulation of the algorithm, instead of requiring a separate proof.

A major limitation of the current design that is shares with all discrete relaxation algorithms is its discrete nature, requiring exact input. In future work, I plan to investigate indexing from structure with uncertain and inexact structure descriptions. Connectionist implementations seem to be natural hosts for such problems, although they often involve state spaces with local minima, and thus require optimizations [Hopfield and Tank 1985]. The implementation and proof given here provide a basis from which to investigate such more general indexing problems.

**Acknowledgements**

Jerry Feldman was an indispensable help throughout. Paul Chou provided invaluable last-minute motivation, suggestions and criticism. Nigel Goddard and Susan Hollbach assisted in the original work. Mike Swain pushed me to the next step.

This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008. The latter contract supports the Northeast Artificial Intelligence Consortium (NAIC). I thank the Xerox Corporation University Grants Program for providing equipment used in the preparation of this paper.

**References**

Cooper, Paul R and Susan C. Hollbach. "Parallel Recognition of Objects Comprised of Pure Structure", Proceeding DARPA Image Understanding Workshop, L.A., February 1987.

Davis, L.S. and A. Rosenfeld. "Cooperating Processes for Low-Level Vision: A Survey", *Artificial Intelligence*, 17:245-163, 1981.

Fanty, M. "A Connectionist Simulator for the BBN Butterfly", TR 164, Computer Science Department, University of Rochester, January 1986.

Feldman, Jerome A. "Connectionist Models and Parallelism in High Level Vision", *Computer Vision, Graphics, and Image Processing 31,* 178-200, 1985a.

Feldman, Jerome A. "Energy and the Behaviour of Connectionist Models", TR 155, Deparment of Computer Science, University of Rochester, 1985b.

Feldman, J.A. and D. Ballard. "Connectionist models and their properties," *Cognitive Science*, 6, 205-254, 1982.

Hinton, G.E. *Relaxation and Its Role in Vision.* PhD Thesis, University of Edinburgh, 1977.

Hopfield, J.J. and D.W. Tank. "Neural Computation of Decision in Optimization Problems", *Biol. Cyber.* 52: 141-152, 1985.

Hummel, Robert A. and Steven W. Zucker. "On the Foundations of Relaxation Labelling Processes", *IEEE Trans. PAMI-5*, No. 3, 1983.

Kitchen, Les and A. Rosenfeld. "Discrete Relaxation for Matching Relational Structures", *IEEE Trans. Systems, Man, and Cybernetics, SMC-9, 12,* 869-874, 1979.

Mackworth, A.K. "Consistency in Networks of Relations", *Artificial Intelligence*, 8:99-118, 1977.

Mackworth, A.K. and E. C. Freuder. "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", *Artificial Intelligence*, 25:65-74, 1985.

Mackworth, Alan K., Mulder, Jan A. and William S. Havens. "Hierarchical Constraint Propagation: Exploiting Structured Domains in Constraint Satisfaction Problems", *Computational Intelligence*, 1: 118-126, 1985.

Mohr, R. and T.C. Henderson. "Arc and Path Consistency Revisited", *Artificial Intelligence*, 28:225-274, 1986.

Shapiro, Linda G. and Robert M. Haralick. "Structural Descriptions and Inexact Matching", *IEEE Trans PAMI-3*, No. 5, pp. 504, 519, September 1981.

Swain, Michael A. and Paul R. Cooper. "Parallel Hardware for Constraint Satisfaction". *Proceedings of the DARPA Image Understanding Workshop* , Boston, MA, April 1988.

Waltz, D. "Understanding Line Drawings of Scenes with Shadows", *The Psychology of Computer Vision*, McGraw-Hill, 1975.

## Appendix

**Definition 1:**

a node $p_e^A$ has a *unique unary (node) match* $m_{eg}$ to node $p_g^B$ iff

(i) $l_e^A = l_g^B$

& (ii) $\forall p_f^A \; [(e \neg = f) \supset (l_e^A \neg = l_f^A)]$

& (iii) $\forall p_h^B \; [(g \neg = h) \supset (l_g^B \neg = l_h^B)]$

In the candidate object graph, there some node with a unique label (ie. no other node in the candidate object has that label). In the target model graph, some node has the same label, and that label is unique in the target model as well.

**Definition 2:**

an arc $r_{ef}^A = (p_e^A, p_f^A)$ has a *unique binary (arc) match* $n_{ef,gh}$ to arc $r_{gh}^B = (p_g^B, p_h^B)$ iff

(i) $l_e^A \neg = l_f^A$ & $l_g^B \neg = l_h^B$
(the node labels on the arcs are not equal)

& (ii) either $[(l_e^A = l_g^B) \& (l_f^A = l_h^B)]$
or $[(l_e^A = l_h^B) \& (l_f^A = l_g^B)]$
(the label pairs on the ends of the arcs match in one direction or the other)

& (iii) $\forall r_{wx}^A = (p_w^A, p_x^A) \; [(ef \neg = wx) \supset (\neg(l_e = l_w \& l_f = l_x) \mid \neg(l_e = l_x \& l_f = l_w))]$
(the label pair in the candidate object graph is unique)

& (iv) $\forall r_{yz}^B = (p_y^B, p_z^B) \; [(gh \neg = yz) \supset (\neg(l_g = l_y \& l_h = l_z) \mid \neg(l_g = l_z \& l_h = l_y))]$
(the label pair in the target model graph is unique

**Definition 3:**

an arc $r_{ef}^A$ is *matchable by propagation at time t* if

(i) $\exists r_{gh}^B [S_{ef,gh} = \textit{contending}]$

and either

(ii) $(S_{eg} = \textit{bound} \; \& \; S_{fh} = \textit{bound}) \mid (S_{eh} = \textit{bound} \; \& \; S_{fg} = \textit{bound})$
(both ends are consistently matched, in one direction or the other)

or (iii) 1 of 4 analogous situations is true. One of the 4 cases is described as follows (the others are obtained from the other 4 consistent permutations of the e,f,g, and h parameters):
$(S_{eg} = \textit{bound}) \&$
$(S_{fh} = \textit{contending}) \&$
$\forall x \; [(x \neg = h \; \& \; \exists r_{gx}^B) \supset (S_{fx} = \textit{off})]$ (*)
(One end is matched, the other end has a consistent contending match, and the matched end has no symmetry with respect to propagation. In other words, $p_g^B$ is connected to *exactly 1 node* $p_h^B$ which is a consistent contending match with $r_{ef}^A$ ).

**Definition 4:**

a node $p_e^A$ with label $l_e^A$ is *matchable by propagation at time t* if

(i) $\exists p_g^B [S_{eg} = \textit{contending}]$
($p_e^A$ can match some node)

& (ii) $\exists r_{ef}^A, r_{gh}^B [S_{ef,gh} = \textit{bound} \; \& \; ((S_{fh} = \textit{bound}) \mid (l_e^A \neg = l_f^A))]$
(at least 1 connected and consistent arc match exists, with either the other end matched, or dissimilar labels on the ends of each arc)

& (iii) $\forall r_{ex}^A \; \exists r_{yz}^B [(x \neg = f \; \& \; S_{ex,yz} = \textit{bound}) \supset (y = g \; \& \; z \neg = h) \mid (z = g \; \& \; y \neg = h)]$
(for all the other matched arcs connected to e, their matches are consistent with the match of ef to gh)

& (iv) $\forall r_{gx}^B \; \exists r_{yz}^A [(x \neg = h \; \& \; S_{gx,yz} = \textit{bound}) \supset (y = e \; \& \; z \neg = f) \mid (z = e \; \& \; y \neg = f)]$
(for all the other matched arcs connected to g, their matches are consistent with the match of ef to gh)

**Definition 5:**

a graph element (node or arc) is *matchable* if

(i) it is a node or arc with a unique match

(iii) it is a node or arc matchable by propagation at time t, for any t

**Lemma 1:** The network detects unique unary (node) matches.

**Proof:**

After time step 0, consider node matching unit $m_{ij}$. If $p_i^A = p_j^B$ is a unique unary label match, then $S_{ij} = \textit{contending}$ and $P_{ij} = 1$. But also:

$\forall x, m_{xj} \; [(x \neg = i) \supset (S_{xj} = \textit{off} \; \& \; P_{xj} = 0)]$

& $\forall y, m_{iy} \; [(y \neg = j) \supset (S_{iy} = \textit{off} \; \& \; P_{iy} = 0)]$

In other words, $m_{ij}$ is uniquely active in its row and column.

The first (propagation) substep of Time Step 1, nothing is changed. So, in the second (WTA) substep of Time Step 1:

$\forall x, m_{xj} \; [(x \neg = i) \supset (X_{ij} > X_{xj})]$

& $\forall y, m_{iy} \; [(y \neg = j) \supset (X_{ij} > X_{iy})]$

Therefore, at the end of Time Step 1, $S_{ij} = \textit{bound}$.

**QED**

**Lemma 2:** The network detects unique binary (arc) matches.

**Proof:**

Consider each $n_{ef,gh}$ after Time Step 1a, the propagation

phase. Recall that each $n_{ef,gh}$ is connected to exactly 4 $m_{ij}$; $m_{eg}$ $m_{eh}$ $m_{fg}$ $m_{fh}$. At the beginning of time step 1, each of these $m_{ij}$ had a potential of 1 (if $l_i = l_j$) or 0. Therefore, the potential X of any $n_{ef,gh}$ is the number of contending valid node-node matches which are consistent with the arc-arc match $n_{ef,gh}$.

Now, suppose $n_{ef,gh}$ is a unique binary match. Recall from condition (i) of the definition of unique binary match that the nodes on the ends of the matched arcs may not be the same. Therefore, $X_{ef,gh} \neg = 4$. To have $X_{ef,gh} = 3$ is impossible. So, $X_{ef,gh}$ must be 0,1, or 2. If $n_{ef,gh}$ is a unique binary match, then $X_{ef,gh} = 2$, and:

either $X_{eg} = 1$ & $X_{fh} = 1$

or $X_{eh} = 1$ & $X_{fg} = 1$ (from condition (ii) of the definition).

But because the particular combination of pairs is unique by definition, all the other elements of the row and column containing $n_{ef,gh}$ have $X < 2$ (from condition (iii) and condition (iv) of the definition).

Therefore, the potential of $n_{ef,gh}$ is larger than that of all units in its row and column, so it wins the WTA competition.

And, at the end of Time Step 1, $S_{ef,gh} = bound$.

**QED**

**Lemma 3:** If an arc $r_{ef}^{A}$ is matchable by propagation at time t and the network is still active (ie. $T^t \neg = T^{t-1}$) then the network assigns that arc a unique consistent match at the end of time t+1

**Proof:**

Case 1 (condition (ii) of definition above is true):

From the network topology, $X_{ef,gh} = X_{eg} + X_{eh} + X_{fg} + X_{fh}$ after the propagation substep beginning time step t+1. From condition (ii), $X_{ef,gh} = 2\beta$. But $X_{ef,gh}$ is the largest potential in its row:

$$\forall x,y\,[\,\neg(x = g \,\&\, y = h) \supset (x_{ef,gh} > X_{ef,xy})\,]$$

because $X_{ef,xy} = X_{ex} + X_{ey} + X_{fx} + X_{fy}$. But at most one of these terms is $\beta$ (if x=g or y=h) and the rest must be zero, because $p_e$ is bound to $p_g$ and $p_f$ is bound to $p_h$. $X_{ef,gh}$ is also the largest potential in its column, by an analogous argument.

Therefore, after the WTA substep, $S_{ef,gh} = bound$.

Case 2 (condition (iii) of definition above is true):

Only the example case is proved. The other 3 analogous cases are identical, with systematically varied subscripts.

Again $X_{ef,gh} = X_{eg} + X_{eh} + X_{fg} + X_{fh}$ after the propagation substep beginning time step t+1. But $X_{eg} = \beta$ and $X_{eh} = X_{fg} = 0$ therefore. Also, $X_{fh} = 1$. Overall, $X_{ef,gh} = \beta + 1$. But this potential is greater than all others in the row:

$$\forall x,y\,[\,\neg(x = g \,\&\, y = h) \supset (x_{ef,gh} > X_{ef,xy})\,]$$

because $X_{ef,xy} = X_{ex} + X_{ey} + X_{fx} + X_{fy}$ and:

(i) suppose x=g or y=h. Then 1 term is $\beta$, say $X_{eg}$ for x=g, without loss of generality. Then $X_{ey}$ is certainly zero. But $y \neg = h$, so both the other terms must be zero as well, from the implication (*) in condition

(ii) suppose neither x=g nor y=h. Then $X_{ex} = X_{ey} = 0$. Because $S_{fh} = contending$, $X_{fx} <= 1$ and $X_{fy} <= 1$. So $X_{ef,xy} < \beta + 1$.

Analogously, the potential $X_{ef,gh}$ is greater than all others in the column.

Therefore, by the completion of the WTA at the end of time step t+1, $S_{ef,gh} = bound$.

**QED**

**Lemma 4:** If a node $p_e^{\Lambda}$ is matchable by propagation at time t and the network is still active (ie. $T^t \neg = T^{t-1}$), then the network assigns that node a unique consistent match at the end of time t+1.

**Proof:**

If $S_{fh}$ is *bound*, it must be that $S_{eh}$ is *off*, as is $S_{fg}$. Alternately, $S_{eg}$ is *contending* and $l_e^A \neg = l_f^A$, so again $S_{eh}$ and $S_{fg}$ are *off*.

Now, after propagation $X_{eg} = \Sigma_{a\sigma} X_{ea,g\sigma}$. Furthermore, $X_{eg}$ is the largest potential in its row:

$\forall y\,[\,(y \neg = g \,\&\, y \neg = h) \supset X_{eg} > X_{ey}\,]$

because:

(a) $S_{ef,gh} = bound$, so $X_{eg}$ receives an input of 1 from $n_{ef,gh}$ and no other element in the same row as $m_{eg}$ receives an input for $S_{ef,gh}$ ($S_{eh}$ would, but as we just saw, from condition (ii) of the definition, $S_{eh} = off$).

(b) If some other $m_{ey}$ receives an input, then some $n_{ex,yz}$ is in state bound as well. (In other words, node e has another bound arc connected to it). But, by the consistency condition (iii), if $S_{ex,yz} = bound$, then either y=g or z=g. In either case, $X_{eg}$ always receives an input of 1 as well.

Therefore, taken together, (a) and (b) dictate that $X_{eg}$ is always at least 1 greater than all other $X_{ey}$ in the same row.

An analogous argument using condition (iv) of the definition holds true for all units in the same column as $X_{eg}$.

Therefore, $X_{eg}$ is larger than all other units in its row and column following the propagation substep.

Therefore, after the WTA substep, $S_{eg} = bound$.

**QED**

# A multi-paradigm development system for exploratory environments

Anne Bergeron, Lorne H. Bouchard and Renaud Nadeau

Département de mathématiques et d'informatique
Université du Québec à Montréal
P.O. Box 8888, Station "A"
Montréal, QC H3C 3P8

**Abstract**

Based on our experience of the problems involved in the development of specialized exploratory environments, we describe the design and implementation of a multi-paradigm (functional, logical and object-oriented) development system for exploratory environments. The presentation focuses mainly on how the paradigms have been integrated in the language, on the implementation of the language and on its integration into the Macintosh environment.

## 1. The design and implementation of exploratory environments

An *exploratory environment* is an environment, realized by a computer program, which allows the student to explore a subject or field of study. Perhaps the best known example of an exploratory environment is the turtle-geometry subset of Logo [Papert 1980] [diSessa and Abelson 1981], an environment in which the student explores finite plane geometry. The bead necklace model [Bouchard and Emirkanian 1986] is the seed of an exploratory environment for studying the structure and the operations on words and lists in Logo. Further examples may be found in the work of Ennals [Ennals 1985], where Prolog is used to model parts of the humanities, and in the ideas at the origin of the Smalltalk language [Goldberg and Robson 1983]. The common feature which links these efforts is the use of a programming language to represent knowledge of some area, the topic of the exploratory environment.

The concept of an exploratory environment evolved rapidly to include more specialized constructions: for example, the addition of knowledge to the exploratory environment and the inclusion of tools specifically tailored for a given subject area, a spreadsheet for example. The knowledge makes the environment smarter and hence richer and more challenging. The tools reduce the programming effort required on the part of the student without sacrificing any functionality in the area of study. Exploratory environments are a generalization of the notion of *microworld* [Papert 1980] [Thompson 1987] and are an active research topic in the field of computer-aided learning and intelligent computer-aided instruction.

In this context, the interest in multi-paradigm programming environments developed rapidly [Borne 1983] [Drescher 1984] [diSessa and Abelson 1986]. The problems of the representation of knowledge is certainly the main reason for this interest since in every discipline we find a collection of facts, rules, theories, structuring principles, and algorithms as well as descriptions of systems and models. Hence many formalisms can and indeed must be used at the same time.

Since 1985, the authors have been involved in the design of exploration environments and in the creation of tools to support the development of such environments. This work initially began with the design of an environment for the construction of animated fairy tales [Bergeron and Nadeau 1986]. A prototype for the project was initially implemented in Smalltalk. In order to port the project to Logo on the Macintosh, an object-oriented extension for Logo was implemented. The software tools were then adapted to the construction of an exploratory environment for Newtonian mechanics. The more recent work has been performed in the context of research into the design of "off the shelf" software tools for education [Paquette et al. 1987]. The principal aim of this project is to build a number of exploratory environments in which traditional programming tools are enriched by the addition of knowledge representation and processing tools. The fundamental assumption of this project is that this merger simplifies the knowledge processing required to support environments that encourage active learning: e.g., free and structured exploration, the elaboration of conjectures, the generation of predictions and the objectivation of knowledge.

The logical aspects underlying the elaboration and the verification of conjectures and the emphasis on knowledge bases in current exploratory environment research makes logical programming an essential part of the development system. This however should not be at the expense of the rapid prototyping facilities [Teitelman and Masinter 1981] [Sheil 1986] and the execution speed of functional programming - in this case Scheme, a dialect of Lisp [Rees and Clinger 1986]. Finally, the modularity offered by an object-oriented programming language is attractive for large systems. Prisme [Nadeau 1988] is a development system which is based on the integration of these three paradigms.

Note that the development systems for exploratory environments share most of the features of artificial intelligence programming languages [Bobrow and Stefik 1986] and indeed are closely modelled on them. However, the hardware available for the development and the delivery of exploratory environments is limited for economic reasons to microcomputers such as the Macintosh.

We first give an example of an exploratory environment which is currently under development. Then, in sections 3 and 4 we discuss some of the design and implementation issues of Prisme.

## 2. Stage-setting: an illustrative exploratory environment

The goal of *Mise en scène* (stage-setting) is to offer young children (6 to 10 year olds) the opportunity to describe and observe the behavior and the interactions of different characters. These animated characters may possess specific factual or strategic knowledge, they react to events and can express themselves in words and music. The child uses a declarative style of programming with a simplified "natural language" style interface. Technical problems, such as the animation of the scene, the synchronization of picture and sound, the management of the coordinate system used for locating objects in the scene, are all hidden from the user.

The exploratory environment appears to the child as a collection of windows which either show a part of the scene or the text describing characters and the script, as for example in Figure 1.

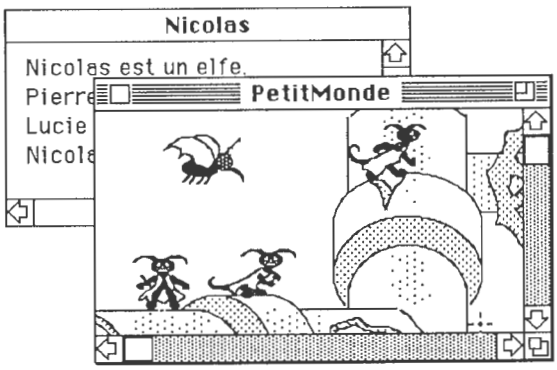



**Figure 1**

The description of a character is made using several types of declarations. A character is described first in terms of his resemblance to one or more of the other characters. For example:

*"Nicolas est un elfe."*
*"Nicolas est un musicien"*

These declarations place the character *Nicolas* : he inherits the features of the *elfe* and *musicien* characters. *Nicolas* can also maintain relationships with other characters and with objects in the scene. For example:

*"Jacques est l'ami de Nicolas".*
*"Lucie aussi".*
*"Nicolas a une flute."*

Finally, the script for the characters is described, as for example:

*"Nicolas va chercher Lucie."*
*"Nicolas et Lucie vont au bord de l'étang."*
*"Nicolas joue de la flute."*
*"Nicolas visite tous ses amis."*

The same system could also be used by several children to allow them to study character interaction.

## 3. The development system

Given the number and the variety of the problems raised by the implementation of this project, the choice - or rather the construction, since none was found to exist - of the development system is crucial. The designer's environment must be highly interactive and must support the exploratory style of programming [Sheil 1986], since the specifications are ill-defined at least initially. Furthermore, the system should facilitate problem decomposition and offer tools appropriate to each type of sub-problem (imperative or declarative styles of programming, data abstraction, etc...).

The Prisme language [Nadeau 1988] is at the heart of the development system. It is implemented as an interpreter for a language based on the harmonious integration of the three programming paradigms: functional, logical and object-oriented programming. Prisme is an extension of the Scheme dialect of Lisp [Rees and Clinger 1986] which includes Horn clauses [Kowalski 1979] and the definition of objects which are similar to *actors* [Lieberman 1986].

## 4. The multi-paradigm integration in Prisme

### 4.1 Interaction with the interpreter

The syntax of Prisme is almost the same as that of Scheme. At the top-level, a user interacts with Prisme by typing in or selecting forms to be processed. In general, a form is an atom or pair (list) of forms, where the first element of a form plays a special role. The primitive *define* is used to name *values*, *objects* and *functions* ; note that variable names start with an underscore. For example:

```
(define pi 3.141592)
(define minimum 4)
(define
     (Surface _radius)
     (product pi (square _radius)))
```

and a call to the function Surface is a form which is written in the usual way:

```
     (Surface (sum minimum 4))
```

In Prisme, this form is evaluated as in Scheme.

Procedures, called *relations* to distinguish them from the functions already described, may be defined in Prisme using Horn clauses. A clause consists of a list of terms, the first of which is called the *head* and the others are called the *goals* of the clause. Typically a term is a non-atomic form the first element of which is either a relation or a function name and the other elements are constants or terms. Variable names starting with an underscore identify logical variables. For example,

```
(defclause
   (Proposition _expr)
   (atom? _expr))

(defclause
   (Proposition (or _expr1 _expr2))
   (Proposition _expr1)
   (Proposition _expr2))

(defclause
   (Proposition (not _expr))
   (Proposition _expr))
```

```
(define
   (meta-eval _form)
   (cond
      ((null? _form) _form)
      ((variable? _form) (var-value _form))
      ((symbol? _form) (sym-eval _form))
      ((pair? _form)
         (route
            (meta-eval (car _form))
            (cdr _form)))
      (T _form)))

(define
   (route _functor _arglist)
   (cond
      ((function? _functor)
         (apply _functor
            (apply list _arglist)))
      ((procedure? _functor)
         (apply _functor
            (apply list _arglist)))
      ((system? _functor)
         (apply _functor
            (apply list _arglist)))
      ((clause? _functor)
         (resolve
            (cons _functor _arglist)))
      ((object? _functor)
         (bind _functor)
         (evlis _arglist)
         (unbind _functor))
      (T error)))
```

### 4.3 Interpretation of functions

Functions are defined in Prisme as in Scheme. Also as in Scheme, the scope of a variable is either global or local to a function. Thus, when a function is defined, Prisme parses its definition and indexes the variables. When the function is called, a block of variables is allocated and stacked. All local variable references inside the function body are to this block.

Extensibility of the functional part of an Prisme program is achieved as in Scheme by defining additional functions. But also, since the calling convention for Prisme functions is the same as that for function written in C, a Prisme program may be extended by linking-in procedures which have been hand-coded in C. This is particularly useful in cases where programming efficiency is critical or when developing extensions which are to be widely used.

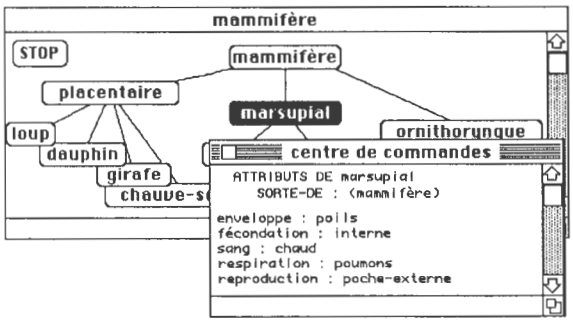



Figure 2

In developing exploration environments, the functional programming part of Prisme is used extensively in input and output tasks, in particular for managing graphic interfaces. Perhaps this reflects the individual bias of the authors: they were all weaned on Lisp! Figure 2 shows a *browser* that allows the child to examine and modify a database.

### 4.4 Interpretation of clauses

The *resolver* is an intepreter for Horn clauses [VanCaneghem 1986] which *resolves* each clause of a list of clauses until all clauses are satisfied. During resolution, the current goal is *unified* with the head of another clause, which means that a substitution of terms for variables is sought which renders the two expressions identical. If unification with this clause succeeds, the current goal is replaced by the remaining goals of the clause. Whenever unification fails, the resolver automatically backtracks to a point where another clause may be tried. When the resolver has exhausted all goals, it returns the values of the variables which were assigned during the unification processes.

In Prisme, variables in clauses and variables in functions are implemented in the same way: this means that we can arbitrarily nest clauses and functions and access the variables from the resolver or the evaluator. Some care must be taken in interpreting the value of a logical variable from the functional part, since the value of logical variables are constructed using structure-sharing. For example, if the evaluator accesses a logical variable the value of which is a list , the evaluator must first reconstruct the list from the structures generated by the resolver.

Since relations and function may be nested, pattern-matching can be used for parameter passing in Prisme. Indeed, referencing part of list structures is a repetitive task which can be avoided by using the unification algorithm to "destructure" a list. Consider the following definition of a procedure, defined as a relation, which is used to process a list which specifies a tour of a number of places, represented as coordinate pairs.

```
(defclause
   (Visit [[_xCoor | _yCoor]| _remaining])
   (set! _place (GetName _xCoor _yCoor))
   (MoveTo _place _xCoor _yCoor)
   (Visit _remaining))
```

The functional version of this procedure must explicitly use selectors to destructure the list and must also test for the terminating condition.
In the clausal version, resolution failure stops the procedure.

```
(define (Visit _coorList)
   (cond
      ((null? _coorList) nil)
      (t (set! _xCoor (caar _coorList))
         (set! _yCoor (cdar _coorList))
         (set! _place
            (GetName _xCoor _yCoor))
         (MoveTo _place _xCoor _yCoor)
         (Visit (cdr _coorList)))))
```

In it is interesting to note that for this example, the clausal form of the procedure actually runs much faster under the interpreter than the functional version.

The definition of procedures as a set of clauses in Prisme opens up the whole new world of logic programming. Of particular interest for exploratory environments are deductive data bases, which are used to represent knowledge, and natural language processing capabilities. The simplicity and effectiveness of the Definite Clause Grammar (DCG) formalism [Pereira and Warren 1980] is particularly useful for defining simplified "natural language" style interfaces. The following example displays a fragment of the DCG used in *Mise en scène*, described in Section 2.

```
(defrule
   (Sentence _meaning)
   (NounPhrase _who)
   (VerbPhrase _who _what _meaning))

(defrule
   (NounPhrase _who)
   (ProperNoun _who))

(defrule
   (VerbPhrase _who _what _meaning)
   (BeVerb)
   (Relation _who _what _meaning))

(defrule
   (Relation _who _what
      (create-link _who _what))
   (IndefiniteDet)
   (Noun _what))

(defrule
   (Relation _who _whom
      (_relation _who _whom))
   (IndefiniteDet)
   (RelationName _relation )
   (Preposition)
   (NounPhrase _whom))

(defrule (ProperNoun Nicolas { Nicolas }))
(defrule (Noun elf { elf }))
(defrule (RelationName friend { friend }))
```

### 4.5 Interpretation of objects

The *switcher* is the component responsible for switching the execution environment. A local environment is represented as a list of value-variable pairs. The *bind* and *unbind* procedures are used to update and to restore the environment. Local values, functions, objects and relations may be associated with an object. The switcher simply superposes the local environment over the global environment, hence local definitions temporarily hide definitions of the same name that are defined.

Objects may be linked together and inherit properties of the objects they are linked to. The links are specified dynamically once an object has been defined. A collection of objects may be structured or restructured a posteriori: this is one way Prisme supports exploratory programming [Sheil 1986].

The (re)definition of the symbols in an ancestor is propagated to its descendants, except when they (re)define the symbol. Prisme supports multiple inheritance, e.g., an object may have many ancestors and many descendants.

In the following example, the object nicolas is successively defined as a descendant of window, elf and musician.

```
(define nicolas (new-object window))
(create-link nicolas elf)
(create-link nicolas musician)
```

The links between objects are stored in Prisme as a data base of clauses which may easily be consulted by a user program. For example, the clause

```
(defclause
   (FamilyOf _ancestor)
   (link _descendant _ancestor)
   (Visit _descendant)
   (FamilyOf _descendant))
```

will Visit each descendant of the common _ancestor.

Objects can be used in Prisme to structure, with little overhead, a set of functions and relations.

Objects can be used to implement generic procedures. In particular, a mechanism similar to *advise* in Interlisp could be implemented in Prisme to help extend inherited functions by enclosing their definitions in a local wrapper.

Objects are an elegant technique to use for implementing contextual data bases. For example, the adjacent Prisme program defines a DCG which is adequate for the characters in *Mise en scène* . However we wish to introduce linguistic and semantic variations from one character to another: indeed, the vocabulary of a musician is different from that of an acrobat and the meaning of "play" is strongly context dependent. This problem is solved by structuring the characters as a set of objects which share a common grammar but also own a private lexicon. This partition of the base of clauses improves the efficiency of the interface - since each lexicon is kept small - whilst permitting the expression of subtle shades of meaning.

Furthermore, since clause inheritance is additive, it is possible for many objects to share the common definition of a predicate, yet still be able to extend it if need be. In effect this allows us to achieve in Prisme something which is much better than the *monde (world)* concept which is available in Prolog II [VanCaneghem 1986].

Objects are very useful for managing animated objects. An animated object is characterized by local state variables such as its position, its speed, its orientation or a set of views. Given the general definition of an animated object, we can easily describe objects which are more specialized in their actions, in the way that they move about or in their relationship to the environment. The approach to animation developed in *Mise en scène* appears to be sufficiently general to be usable for graphic simulations in science as for example, in studies of free fall or of collisions, etc.

The user interface of Prisme is itself written in Prisme: in particular, Windows, Menus and Dialogs are implemented as objects.

Finally, objects are used extensively in a uniform implementation of information processing tools which generalize the operations found in spreadsheets and databases [Bergeron 1988].

## 5. Implementation

Prisme has been implemented in C on the Macintosh. The choice of C makes Prisme not only portable but allows Prisme to be extended easily and efficiently. Indeed, since the Macintosh is a only a microcomputer, the system must be kept close to the hardware if we are to expect an acceptable level of performance.

Prisme is completely integrated with the Toolbox and the development system makes full use of the resources of the Macintosh: desk accessories, text editors, graphic editors, etc.

## 6. Status of the project

The interpreter for Prisme is complete and is currently in beta-test. Compilers for the functional, logical and object-oriented subsets are under development as well as a *project manager* , which is a library of procedures which allows a developer to configure an exploratory environment which is tailor-made for a group of users.

## 7. Conclusion

Prisme is a development system which seems to live up to to our expectations, at least for the moment. Now comes the acid test which is to implement in Prisme a number of the prototypical exploratory environments proposed as part of the Loupe project [Paquette et al. 1987].

In our view, the development of Prisme is an application to the microcomputer environment of the research done in artificial intelligence (AI) on languages for knowledge representation. Perhaps if Prisme is used one day to support in the laboratory the increasing number of AI courses offered in our universities, then we shall consider that we have repaid part of our debt to AI.


## Acknowledgements

The design and implementation of exploratory environments is the topic of Anne Bergeron's Ph.D. research. Renaud Nadeau implemented the Prisme system over a period of a year, as part of his Master's project, under the supervision of Lorne H. Bouchard. Anne Bergeron was responsible for testing the implementation and was the first real user of the system. Although Renaud Nadeau is responsible for the initial design of Prisme, all three authors have contributed also.


## References


Abelson, H. et G. J. Sussman. *Structure and Interpretation of Computer Programs*, MIT Press, 1985, 542 p.

Bergeron, A. Environnements d'exploration: représentation et traitement de l'information, *Proc. First International Conference on Intelligent Tutoring Systems*, Montréal, June 1988, (in press).

Bergeron, A. and R. Nadeau. Building Microworlds in an Object-Oriented Logo, *Logo 86 Conference*, Boston MA, 1986, p. 265.

Bobrow, D.G. and M.J. Stefik. Perspectives on Artificial Intelligence Programming, p. 581-587 in [Rich and Waters 1986].

Borne, I. Les objets dans Logo, BIGRE No. 37, *Journée d'étude sur les L. O. O.*, Oct. 1983.

Bouchard, L. H. and L. Emirkanian. The Bead-Necklace Model for Words and Lists in Logo, *Logo 86 Conference*, Boston MA, 1986, p. 35-38.

diSessa, A. and H. Abelson. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics* , The MIT Press, Boston MA, 1981, 477 p.

diSessa, A. and H. Abelson. Boxer: A Reconstructible Computational Medium, *Communications of the ACM* , Sept. 1986, p. 859-868.

Drescher, G. and J. Ressler. Object-Lisp User Manual, Lisp Machine Inc., 1984.

Ennals, R. *Artificial Intelligence: Applications to Logical Reasoning and Historical Research* , Ellis Horwood, Chichester UK, 1985.

Goldberg, A. and D. Robson. *Smalltalk-80: The Language and its Implementation* , Addison-Wesley, Reading MA, 1983, 714 p.

Kowalski, R. *Logic for Problem Solving* , North-Holland, New-York, 1979, 287 p.

Lieberman, H.. Using Prototypal Objects to Implement Shared Behavior in Object-Oriented Systems, *OOPSLA '86 Proceedings,* Sept. 1986, p. 214-223.

Nadeau, R.. *Intégration de trois paradigmes de programmation* , Mémoire de maîtrise, Département de mathématiques et informatique, UQAM, 1988.

Papert, S. *Mindstorms: Children, Computers and Powerful Ideas* , Basic Books, New York, 1980, 230p.

Paquette, G. et al. Un environnement de conception de logiciels-outils pour l'apprentissage par traitement des connaissances, Proposal to APO Québec, Aug. 1987, 45 p.

Pereira, F. and D. Warren. Definite Clause Grammars for Language Analysis: A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence* , Vol. 13, 1980, p. 231-278.

Rees and Clinger (Eds). Revised Report on the Algorithmic Language Scheme, *SIGPLAN Notices* , Vol. 21, No. 2, Dec. 1986.

Rich, C. and R.C. Waters (Eds). *Readings in Artificial Intelligence and Software Engineering* , Morgan Kaufmann, Los Altos CA, 1986, 602 p.

Sheil, B. Power Tools for Programmers, p. 574-580 in [Rich and Waters 1986].

Steele, G and G.J. Sussman. The art of the interpreter, AI Memo No. 453, MIT, May 1978.

Stefik, M. and D. Bobrow. Object-Oriented Programming : Themes and Variations, *AI Magazine* , Vol. 4, No.4, Winter 1986, p. 40-62.

Teitelman, W. and L. Masinter. The Interlisp Programming Environment, *IEEE Computer* , Vol. 14, No. 4, 1981, pp. 25-34.

Thompson, P.W. Mathematical Microworlds and Intelligent Computer-Assisted Instruction, p. 83-109, in Kearsly, G. P. (Ed.), *Artificial Intelligence & Instruction: Applications and Methods*, Addison-Wesley, 1987, 351 p.

VanCaneghem, M. *L'Anatomie de Prolog*, Inter Editions, Paris, 1986, 191 p.

# A Hybrid Approach to Finding Language Errors and Program Equivalence in an Automated Advisor

*Xueming Huang* and *Gordon I. McCalla*

ARIES Laboratory
Department of Computational Science
University of Saskatchewan
Saskatoon, Canada S7N 0W0

*Abstract* ---- This paper presents a model of a language expert which recognizes language bugs and determines program equivalence in the context of the SCENT programming advisor being developed at the University of Saskatchewan. A hybrid approach, which combines the advantages of knowledge-based and theorem proving techniques, has been used in the language expert. Knowledge-based debugging is based on a debugging graph which is intended to recognize common program constructs and common bugs which frequently appear in students' programs. Empirical studies of the performance of the language expert show that it has achieved high performance in detecting and identifying language errors.

*Key Words* ---- **intelligent advising, language knowledge, automated debugging, program equivalence.**

## 1. INTRODUCTION

Students' programs often contain bugs resulting from misconceptions in the use of the programming language. Moreover, a variety of language constructs can be used to produce a correct language. Finding the language-related bugs (or language bugs for short) is essential for a program advising system to provide appropriate feedback to the student, while recognizing equivalence between programs is important for the system to "understand" the student's program.

Misunderstanding of the language knowledge is not the only source of bugs in a program. Misunderstanding the task of the program and using an erroneous strategy can also generate bugs. Most previous program advising systems use only one debugging component to detect all kinds of bugs. This approach complicates bug recognition because different kinds of knowledge have to be invoked at the same time. It also increases the size of the knowledge base since the same language bug occurs multiply in different strategies and different tasks. Furthermore, it is difficult to find a knowledge representation which is ideal for all kinds of knowledge. This paper presents the model of a separate *language expert* which solely deals with language bugs and provides language expertise for an intelligent program advising system. Complete details can be found in [Huang 87].

The language expert is developed as a component of the SCENT advisor [McCalla 86] [McCalla 88] whose architecture is shown in Fig. 1. Each box in the figure represents a different component of SCENT. Besides the language expert itself, of particular concern to this research is the *strategy judge/diagnostician (SJ/Diag)* component. The SJ/Diag component determines the strategy a student is employing and detects buggy strategies. It provides an ideal program of the employed strategy for use by the language expert in its detection of language bugs. The language expert in turn provides information about equivalence between programs to help other components of the advisor to understand the student's program. These components are being developed in [Barrie 88], [Pospisil 88] and [Escott 88].

The remainder of this paper presents the approach to recognizing language bugs taken by the language expert and contrasts it with related work in automated debugging. Section 2 surveys previous automated debugging systems, focussing on their approaches to finding language bugs. Section 3 presents the model of the language expert. Section 4 gives an empirical evaluation of this model. Section 5 concludes the paper with a summary of contributions of the research and suggestions for future research directions.

## 2. RELATED WORK

Several different approaches have been used in recognizing bugs in programs. PGM [Ruth 76] and LAURA [Adam 80] attempt to find a representation in which programs can be matched against a pre-stored general algorithm. This approach doesn't work well due to the wide variability of programs. There is usually more than one possible algorithm to solve a non-trivial problem.

SNIFFER [Shapiro 81] and PUDSY [Lukey 80] try to match a set of plans (or formulae) explicitly specified by the students against the dynamic behavior of a program (also represented in plans or formulae) obtained from execution of the program. This approach is not suitable for an automated advisor since explicitly specifying intentions is difficult for novices. Also, it may have trouble in recognizing equivalent programs which use different program constructs because these programs may have different dynamic behaviors.

PROUST [Johnson 85] has made a great effort at automatically deciphering the student's intentions underlying the code of a program. It analyzes programs using a knowledge base in which programs are represented at three levels: problems, goals and plans. By figuring out students' intentions, PROUST can handle programs using more than one strategy. A main problem with PROUST is that ultimately a student's code is matched with plans at a low level of language constructs. At this low level, programs can vary in thousands

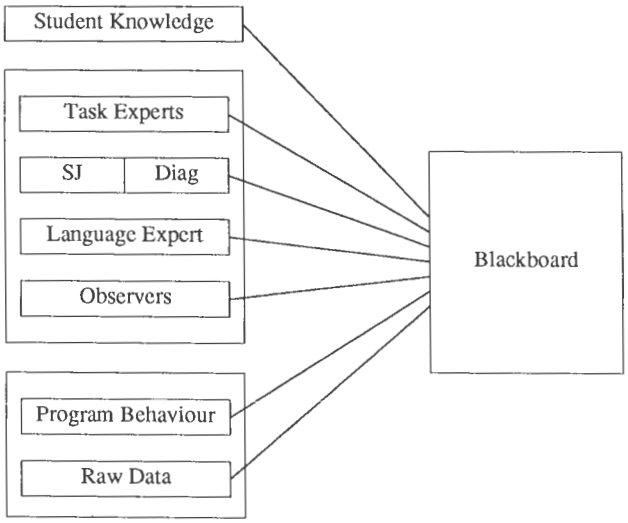



Fig. 1. The Architecture of the Current SCENT Advisor

of ways, thus requires the knowledge base to have thousands of plans, otherwise it may miss some programs.

TALUS [Murray 87] uses a computational logic to formalize knowledge of a programming language. Debugging a program is realized by using a logic theorem prover to prove the equivalence between each segment of the program and the corresponding segment of the pre-stored reference program using the same algorithm. The deductive ability of a theorem prover enables TALUS to understand more program variations than PROUST. A drawback of using theorem proving in automated debugging is its potential inefficiency. Since determining program equivalence is NP-hard, computation required in theorem proving is likely to combinatorially explode as the size of the program grows.

The first two approaches are not suitable for an automated advisor. Each of the other two approaches, using a multi-level knowledge base and using a theorem prover, has some advantages over the other. It seems reasonable to combine knowledge-based and theorem proving techniques to gain the benefits of both. This is the approach taken in the development of the language expert model in this research.

## 3. A MODEL OF THE LANGUAGE EXPERT

One important property of the language expert is that it makes no attempt to understand the global plans (i.e. the strategies) underlying the student's program. The strategy used in the student's program and mistakes at the strategy level are caught by the SJ/Diag component of the intelligent advisor. Thus, a canonical knowledge representation for the language can be used in the language expert and the size of the knowledge base can be minimized. Of course, it is almost impossible to understand a program at the strategy level without understanding the code. Thus, the SJ/Diag component has to communicate with the language expert during the strategy judging process. In the SCENT advisor, a message sent by the SJ/Diag component to the language expert is a segment of the student's program paired with a segment of a pre-stored ideal program which at the language level correctly implements a (correct or buggy) piece of strategy known by the advisor. This pair of program segments is the input of the language expert. The language expert then determines the equivalence of the two program segments, and outputs the results. These results will help the SJ/Diag component to determine whether the two program segments are equivalent at the strategy level. If they are, then any differences at the language level from the ideal segment are considered language bugs. A program segment can also be the whole program, so this scheme also works for finding language bugs in the whole student program.

Throughout this section, the pair of programs in Fig. 2 is used to show the approach of the language expert. Each of these programs is used to solve the **descendants** problem which requires the student to collect and to return the list of all descendants of a given person in a family tree. The function **children** inside the programs finds the children of a given person in the family tree.

### 3.1. The Debugging Mechanism

The language expert accomplishes debugging by combining knowledge-based techniques and theorem proving techniques. It proceeds as follows: the language expert first consults a knowledge base to match the student's program segment (or student segment for short) against the ideal program segment (or ideal segment for short), and tries to explain differences between the program segments. The match is carried out from the outer levels of function application to inner levels. Thus, as the match goes on, the program segments are split and become smaller and smaller.

Any differences which cannot be explained by the knowledge base are sent to a theorem prover which tests the equivalence of the ideal segment to the student segment (which is usually a small part of the original segment). A set of general axioms and theorems may allow the theorem prover to recognize programs which are not understood by the knowledge base, and it can often do this without suffering

```
(defun descendants (people)
  (cond ((null people) nil)
        ((listp people) (append (append (list (cdr people))
                                        (cdr (descendants (car people))))
                                (descendants (cdr people))))
        ((atom people) (descendants (children people)))
  )
)
```

(a) The Student Program

```
(defun descendants (people)
  (cond ((null people) nil)
        ((atom people) (descendants (children people)))
        (t (append (cons (car people) (descendants (car people)))
                   (descendants (cdr people))))
  )
)
```

(b) The Corresponding Ideal Program

Fig. 2. A Pair of Programs for the Descendants Problem

inefficiency caused by combinatorial explosion of computation since it is usually working with small program segments. The following sub-sections will discuss in detail these two parts of the debugging mechanism.

### 3.1.1. Knowledge-Based Debugging

Knowledge-based debugging is implemented using a debugging graph through which program segments are filtered, top-down, to see if they match program concepts, deviations and variations pre-stored in the graph. The debugging graph is intended to incorporate common Lisp program constructs which are frequently used in students' programs and bugs frequently made by students. The skeleton of the graph is shown in Fig. 3 and the sub-graphs for list construction functions and list break-down functions (whose roots, *list-cons* and *list-break,* can also be found in Fig. 3) are shown in Fig. 4 and Fig. 5, respectively.

Nodes of the graph are entities. They store knowledge of program constructs and keep procedures that match the constructs against the tested program. Each node also has a procedure to execute the match, to pass control and to return match results. Different levels of the hierarchy represent different levels of abstraction. This hierarchy allows knowledge to be well organized and to be efficiently searched. The second lowest level is the primitive level at which program constructs are stored. Differing from other levels, nodes at the lowest level store common deviations and correct variations of the program constructs stored at the second lowest level.

Links classify the program constructs and guide the match process to go from the higher levels to the lower levels of the debugging graph. The meaning of each match step is stored in a link but not in a node because matching a node has a determinate meaning only when a certain link was traversed to get to this node. For example, if a buggy program segment is (cons x y) (which matches the node *w-cons* ---- see Fig. 4), the bug could be either a "cons for append" or a "cons for list". If the match arrives the *w-cons* node after traversing the left-most out-link of the *list-build* node, however, a "cons for list" bug is determined.

There are four types of links: refinement (R), implementation (I), variation (V) and deviation (D). R links denote various levels (except the primitive level) of abstraction while I links indicate the primitive level of abstraction. They are used to guide the matching of ideal program segments against stored program constructs in the knowledge base. V links and D links are used to find out and explain differences between the ideal segments and the student segments.

To use the debugging graph to detect language bugs in a student program, the graph interpreter first matches the ideal program segment against the debugging graph to find out the program construct used in the ideal segment. This first stage of the match process starts at the root of the debugging graph, and finishes when an I link is passed. The terminal node of this I link (called the *pit-node)* contains the program construct being looked for.

At the second stage of the match process, the interpreter proceeds to detect bugs by matching the student program segment against the debugging graph. This match starts at the pit-node found in the previous match process. If the student segment is matched with the pit-node or a variation node of the pit-node (by passing a V link), then it is equivalent to the ideal segment (i.e. it is correct). If the student segment is

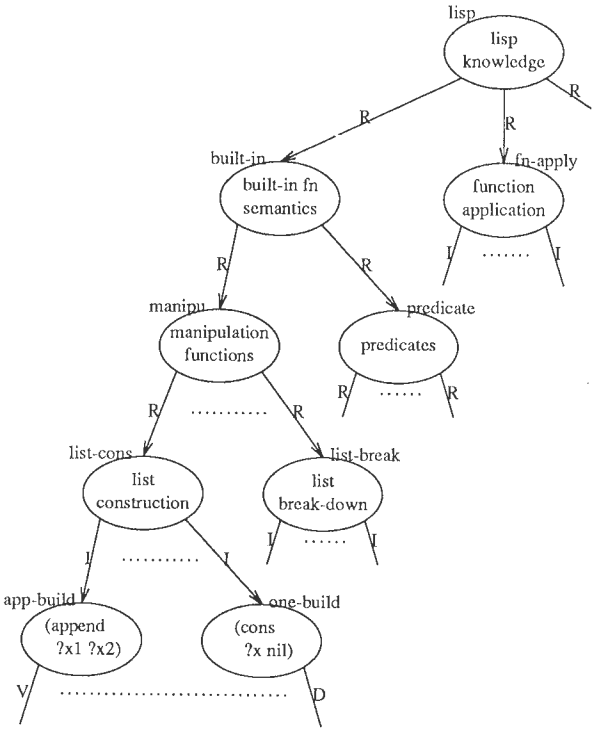



Fig. 3. The Top Level of the Debugging Graph

matched with a deviation node of the pit-node (by passing a D link), then the student segment has an expected kind of deviation of the ideal segment (i.e. a common bug is found).

A program construct stored in the debugging graph is normally a template represented a one-level function, (e.g. (cons ?x1 ?x2) ), while a program segment is usually composed of a number of nested functions. Thus, debugging has to be carried out in several recursive loops intended to unravel the nestings. The interpreter matches one level of the ideal segment against the graph each time; then it matches the corresponding level of the student segment, producing a debugging result for that level. After that, it recurs, matching segments inside the outer segments just dealt with. This recursive process is carried on until the student segment is exactly the same as the ideal segment or the innermost level of the student segment has been analyzed. This analysis method is called the "level by level" method. It is straightforward to extend this level by level analysis method to deal with some templates which are themselves nested (e.g. (append (list ?x1) ?x2) ).

Sometimes, a discrepancy in the student segment can't be explained by the match of the current level but can be explained by an outer level which has been analyzed before. In this case, back-up occurs. The analysis of the outer level is redone. An example of back-up is shown in [Huang 87].

Since the debugging graph stores only a limited number of variations and deviations of each program construct, it could fail to be matched with the student segment. In this case, the debugging graph is not able to explain the discrepancy between the student segment and the ideal segment. Then the

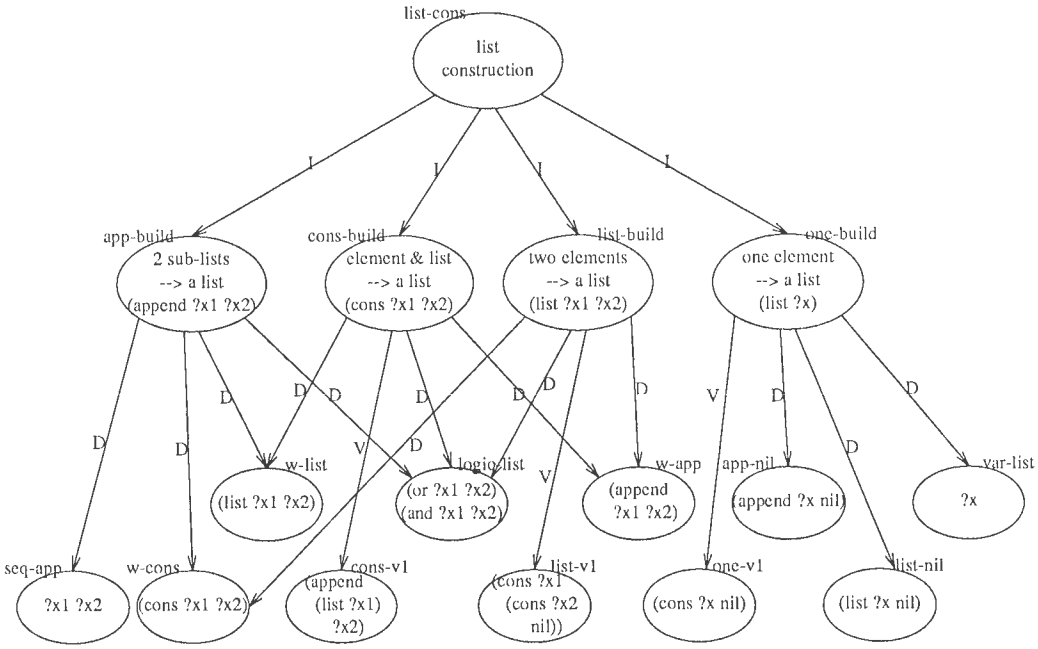



Fig. 4. The Sub-Graph for List Construction

theorem prover can be called to analyze this pair of program segments.

### 3.1.2. Using the Theorem Prover for Debugging

When the debugging graph cannot explain the difference between the student segment and the ideal segment, a theorem prover can be invoked for debugging. To date, unlike the debugging graph, the theorem prover hasn't actually been implemented in the system. In what follows it will be discussed conceptually.

Debugging by the theorem prover would be realized by proving functional equivalence between the ideal program segment and the student's program segment with Boyer-Moore logic [Boyer 79] which is a precise formalism for expressing properties of inductively constructed data objects and recursive programs. If the student segment is functionally equivalent to the ideal segment, it would be considered correct, otherwise it would be considered buggy. The bug would be located by recursively replacing a piece of the student program by a piece of the ideal program, until the modified student program is also buggy.

The approach of debugging programs with a Boyer-Moore logic theorem prover has also been used in TALUS [Murray 87]. However, proofs carried out by the theorem prover of the language expert should be much faster than proofs in TALUS since the debugging graph has already analyzed most parts of the program segments, usually leaving only small parts of them to look at. Also, the theorem prover would be called only if the debugging graph could not explain the discrepancy, so it is called less often than the one in TALUS.

### 3.1.3. An Example

In this section, the debugging process for the action part of the second conditional case of the student program in Fig. 2 is presented to demonstrate how the debugging graph is used to find language bugs. Assume this student segment is already correctly paired with its corresponding ideal segment, the action part of the third ideal conditional case (how to pair them is discussed in Section 3.2). Using the "level by level" method, the debugging process is divided into a number of steps shown in Fig. 6, where segment I is from the ideal program and segment S is from the student program.

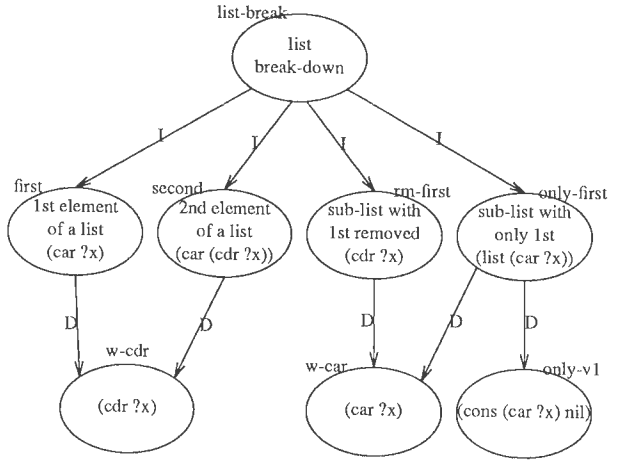



Fig. 5. The Sub-Graph for List Break-Down

```
Step a
I:   (append (cons (car people) (descendants (car people)))
             (descendants (cdr people)))
S:   (append (append (list (cdr people))
                         (cdr (descendants (car people))))
             (descendants (cdr people)))
Step b
I:   (cons (car people) (descendants (car people)))
S:   (append (list (cdr people))
             (cdr (descendants (car people))))
Step c
I:   (car people)
S:   (cdr people)
Step d
I:   (descendants (car people))
S:   (cdr (descendants (car people)))
Step e
I:   (descendants (cdr people))
S:   (descendants (cdr people))
```

Fig. 6. An Example of Debugging by the Language Expert

At step a, the interpreter first matches the ideal segment against the debugging graph. The match starts from *lisp,* goes through a R link to *built-in,* then *manipu, list-cons,* and finally passes an I link, arriving at the pit-node *app-build* (see Fig. 3) which is satisfied that the outer level of the ideal program segment "(append (cons ....) (descendants ....))" matches the template "(append ?x1 ?x2)". Then it starts to analyze the student segment. Since the student segment also matches the template in *app-build,* it is immediately proved correct.

The interpreter recurses one level to step b at which it tries to match the first argument of the ideal segment "(cons ....)" against the first argument of the student segment "(append (list ....))". The path of matching the ideal segment at this step is similar to the one at step a except that now the pit-node is *cons-build.* The student segment, however, doesn't match the pit-node this time. Thus, the interpreter tries V-type out-links of *cons-build.* The student segment passes a V link and matches the *cons-v1* node (see Fig. 4). Thus, this level is recognized as correct.

Now, the analysis is at step c where the first arguments of the ideal and student programs at step b are considered. After going through several R links and an I link, the ideal segment, "(car people)", matches the pit-node *first* in the sub-graph for list break-down (in Fig. 5). Since the student segment, "(cdr people)", doesn't match the pit-node, an attempt is made to match *w-cdr* via a D link. The match succeeds, and a "cdr for car" error is found (this information is associated with the D link).

The debugging process goes on to step d where the second argument of the ideal and student programs from step b are considered. The ideal segment eventually matches the pit-node *app-build* after passing several links, but the student segment fails to match any child of *app-build* and *app-build* itself. So, the match backs-up to the root of the debugging graph, the *lisp* node. The interpreter tries to use the next sub-graphs of *lisp* to test if there are any function application errors in the student segment. None of these kinds of errors is found. The interpreter would finally call the theorem prover. The theorem prover would prove that the student segment is not equivalent to the ideal segment, and thus is buggy.

At step e (which considers the second arguments of the templates found at step a), the student segment is identical to the ideal segment, so it is immediately proved correct. At this stage, debugging for the action part of the third case is done. There have been two bugs detected in this student program segment: a "cdr for car" error (found using the debugging graph) and the use of "(cdr (descendants (car people)))" when "(descendants (car people))" is needed (found using the theorem prover).

### 3.2. Case Splitting and Case Identification

The above discussion on finding language bugs is based upon an assumption that the student program is a small segment without conditional branches. When the program segment is a conditional function such as a *cond* or an *if* (in what follows only *cond's* are discussed since *if's* are similar), the debugging mechanism cannot be directly applied since the order of the conditional cases could differ. Instead, the *cond* function being debugged has to be split into conditional cases and then each conditional case is paired with the corresponding conditional case of the ideal program segment. In the language expert, this is done by a case splitting process and a case identifying process. Dealing with *cond's* is especially important when the program segment is a whole function definition because most recursive Lisp functions are composed of *cond's.*

The case splitting process splits a program into conditional cases, dividing a case into a test part and an action part, attaching an identifier to each case. Then, it sends its products to the case identifying process. If the program has nested *cond's* inside the *cond* function, each case of the nested *cond's* can also be viewed as a case in the outer *cond.* The case splitting process replaces these nested cases by their functionally equivalent cases in the outer *cond.* The test part of a created case is the conjunction of the test in the outer *cond* and a test in the nested *cond.* Handling nested *cond's* in this way has the advantage that it enables the language expert to analyze different implementations for a strategy with one stored ideal program even if some implementations have nested *cond's* while others haven't.

As an example, Fig. 7 shows a solution for the *hard-find-b* problem and the output generated by the case splitting process (where the **hard-find-b** problem requires the student to write a program which takes a list as an argument and which returns another list containing the B's found anywhere in the input list). This program contains a nested *cond.* The first element in each output case (e.g, i2) is the identifier of this case.

The case identifying process is more complicated than the case splitting process. Identifying a case of the student program is realized by pairing this case with the corresponding case of the ideal program. This pairing process is a process of finding the best matched ideal case with the student case, so it is based upon the match results generated by the debugging process described in Section 3.1.

Five grades are used for measuring the match:

(1) **full match** ---- which means that two conditional cases are entirely matched in both the test part and the action part, that is, no error is found in the conditional case of the student program;

(2) **good match** ---- which means that one part (either the test part or the action part) of the student case is entirely

```
(defun hard-find-b (l)
  (cond ((null l) nil)
        ((atom (car l)) (cond    ((eq (car l) 'B)
                                  (cons 'B (hard-find-b (cdr l))))
                                 (t (hard-find-b (cdr l)))
                        ))
        (t (hard-find-b (append (car l) (cdr l))))
  )
)
```

(a) The Input Program

```
((i1 (null l) nil)

 (i2 (and (not (null l)) (atom (car l)) (eq (car l) 'B))
   (cons 'B (hard-find-b (cdr l))))

 (i3 (and (not (null l)) (atom (car l)) (not (eq (car l) 'B)))
   (hard-find-b (cdr l)))

 (i4 (and (not (null l)) (not (atom (car l))))
   (hard-find-b (append (car l) (cdr l))))
)
```

(b) The Output from the Case Splitting Process

Fig. 7. An Example of the Case Splitting Process

matched with the ideal case, while another part deviates from the ideal case, but the deviation is recognized by the debugging graph, not by the theorem prover; that is, although this part is buggy, the bug is a common bug;

(3) **half match** ---- which is similar to "good match" except that it implies that at least one bug in the student program is not recognized by the debugging graph but by the theorem prover.

(4) **minimal match** ---- which means that there are bugs in both the test part and the action part of the student case, but at least one of the two parts has no bug which cannot be recognized by the debugging graph.

(5) **no match** ---- which is the worst case in which both the test and the action have uncommon bugs that cannot be recognized by the debugging graph.

According to the above five grades, the language expert can pair conditional cases of the student program with conditional cases of the ideal program, even if there are bugs in both the test and the action of a conditional case. For the pair of programs shown in Fig. 2, for example, the language expert pairs the first student case with the first ideal case, the second student case with the third ideal case, and the third student case with the second ideal case.

Note that although another automated debugging system, TALUS, also pairs cases of the student program with cases of the ideal program, it seems only to work when all test parts of the student program are correct. If the test part of a student case is buggy, TALUS may not be able to find an ideal test that is functionally equivalent to this student test, which would result in a failure of the case pairing, and would further disrupt debugging for this conditional case.

From the above discussion, it can be seen that there is no necessity to have a separate debugging process after the case identifying process. All debugging results have already been generated during this process. In fact, debugging results for the correctly paired cases are outputs of the language expert, while results for the incorrectly paired cases are cleaned up by the system at the end of the case identifying process. Therefore, debugging is also automatically done once case identification is done.

## 4. EMPIRICAL EVALUATION

This section gives an empirical evaluation based on performance results when the language expert was tested on 65 student programs. Each of these programs was used to solve one of the three problems: the descendants problem, the **deep-find-b** problem (which requires the student to write a program to determine whether the atom B occurs anywhere in a list or not) and the hard-find-b problem. Table 1 summarizes the data used for the evaluation. These data were collected over the last two years from computational science students who were solving these problems as assignments in a programming language course. The students were first-time Lisp programmers, but had experience in some other programming languages. For each student, the first version, the final version and an intermediate version of the syntactically correct program were chosen for analysis.

| | progs. | buggy progs. | total bugs | lang. bugs |
|---|---|---|---|---|
| desc | 8 | 5 | 14 | 10 |
| d-f-b | 27 | 16 | 22 | 11 |
| h-f-b | 30 | 18 | 29 | 20 |
| total | 65 | 39 | 65 | 41 |

Table 1. Summary of Data Analysis

Table 2 illustrates performance in detecting and identifying language bugs. Here, "detecting" a bug means that the language expert has found out that the program is buggy, and located a bug. "Identifying" a bug means that it has not only found the buggy code, but has also identified what the exact nature of the bug is. These performance results show the good debugging ability and the robustness of the language expert. Although the theorem prover is absent in the implemented system, all bugs in the 65 tested programs are detected (which is not surprising because the current system treats program segments which are not recognized by the debugging graph as bugs). More than 75% of the 41 language bugs are identified by the debugging graph. The rate of false alarms (i.e., bugs detected that weren't really there) in the tested programs is only 9.2%. These excellent results could be improved still further once the theorem prover is connected to the system.

Performance in case identification is even more encouraging. The results are shown in Table 3. Here, the column with title "unpaired progs" summarizes programs on which the language expert fails to identify at least one conditional case, although it correctly identifies other conditional cases. The column with title "incorrect paired" summarizes

| | bugs | detected | identified | false alarms |
|---|---|---|---|---|
| desc | 10 | 10 (100%) | 9 (90%) | 0 (0%) |
| d-f-b | 11 | 11 (100%) | 9 (81.8%) | 4 (14.9%) |
| h-f-b | 20 | 20 (100%) | 13 (65%) | 2 (6.7%) |
| total | 41 | 41 (100%) | 31 (75.6%) | 6 (9.2%) |

Table 2. Performance on Detecting and Identifying Language Bugs

programs on which the language expert has made mistakes in case identification; that is, some conditional cases in the students' programs are incorrectly paired with conditional cases in the ideal programs which in reality do not correspond to them. It can be seen that almost 90% of programs had all their conditional cases correctly identified. For the rest, most of them had more than half of their conditional cases correctly identified. Unidentified conditional cases were recorded and reported. Only one program (1.5% of the 65 programs) had conditional cases incorrectly paired with the ideal program. System failures (bugs not identified, false alarms, unpaired cases and incorrect case pairing) can be attributed to:

---- *Absence of the theorem prover in the current implemented system.* The correct program variations which are not recognized by the debugging graph are now inappropriately treated as bugs, producing false alarms. Most of these variations could be recognized by the theorem prover.

---- *The incompleteness of knowledge about common bugs stored in the debugging graph.* This incompleteness results in the failure to identify some common bugs.

---- *Existence of strategy bugs in students' programs.* If the SJ/Diag level of the SCENT advisor is hooked up, the pollution caused by strategy bugs can be filtered out by pairing the student program with the "ideal program" of the corresponding buggy strategy.

---- *Absence of task knowledge and strategy knowledge.* The language expert sometimes needs these two kinds of knowledge to determine the correctness of a program [Huang 87], but they are currently not available.

---- *Inadequate handling of bad-cond's.* A bad-cond is a *cond* clause with some flaws (e.g. missing the outermost pair of brackets). A very high percentage of false alarms and unpaired cases in the empirical evaluation is caused by bad-cond's. It seems that a better handling of them could greatly reduce the rate of system failures.

| | programs | unpaired progs. | incorrect paired |
|---|---|---|---|
| desc | 8 | 1 (12.5%) | 0 (0%) |
| d-f-b | 27 | 5 (18.5%) | 1 (3.7%) |
| h-f-b | 30 | 1 (3.3%) | 0 (0%) |
| total | 65 | 7 (10.8%) | 1 (1.5%) |

Table 3. Performance on Case Identification

## 5. CONCLUSIONS

### 5.1. Summary of Contributions

This research has created an independent language level for the automated programming advisor SCENT. Separating the process of recognizing program constructs (at the language level) from the process of identifying strategies (at the SJ/Diag level) enables more correct identification of bugs at the two conceptual levels since now a bug at one level is not incorrectly recognized as a bug at the other level. The separation also allows a more general knowledge base in the system, since now the strategy level can be more language-independent and the language level can be more task-independent. The communication between the language level and other levels, of course, requires an intelligent control mechanism, which is realized by the blackboard component in SCENT [Ng 86].

The approach of combining a knowledge-based debugger and a theorem prover to find language errors and program equivalence has been used in the language expert. This approach ideally enables the language expert to have good generality and wide coverage (from the theorem prover), and to have high efficiency and the ability to recognize the nature of bugs (from the knowledge base). Using this approach, the system can also easily provide information about how far a student program segment is from an ideal program segment, thus supporting the case identifying process.

Knowledge-based debugging has been implemented in a debugging graph. Empirical evaluation has shown that the debugging graph has achieved high performance in detecting and identifying language bugs, as well as determining language level program equivalence.

The language expert has dealt with a challenging problem in Lisp program debugging ---- conditional case identification, which is not dealt with in some current Lisp debuggers (e.g., PHENARETE [Wertz 82] and the LISP TUTOR [Anderson 85] [Reiser 85]) and not handled well in others (e.g., TALUS [Murray 86] which seems to only identify conditional cases without bugs in the test parts). The language expert uses a specific procedure to accomplish case identification, which allows it to handle Lisp programs with bugs in both test parts and action parts of their conditional cases. The empirical evaluation has shown that case identification in the language expert is very successful.

### 5.2. Problems and Future Directions

Some problems with the implemented system and limitations of the current model have been found. These problems and limitations as well as more ambitious targets suggest future directions of the research.

The pure "level by level" method for matching the debugging graph against programs doesn't seem to be flexible enough to handle the variety of students' programs, since some program constructs might vary at several function application levels from their equivalent constructs. Handling these program constructs requires a more sophisticated matching method.

Match patterns in nodes of the current debugging graph are consistent with the "level by level" match method. If knowledge for program constructs affecting several function

application levels is included in the knowledge base, the number of variations of these more complicated constructs could be very large, and thus a more general representation for match patterns would be needed.

Handling programs with side-effects has been a concern of research into program debugging for years, but a satisfactory way of dealing with side-effects has not been found. This research has been focussed on side-effect free, conditional, recursive programs, and hasn't dealt with the side-effect problem. This problem, undoubtedly, will continue to motivate more research in the future.

In spite of problems mentioned above, this research has achieved a considerable success in using a new approach to finding language errors and program equivalence in an automated advising system. It has shown the potential for combining knowledge-based and theorem proving techniques to help automated debugging.


### Acknowledgements

We would like to thank the rest of the SCENT research group, in particular Jim Greer, Rick Bunt, Bryce Barrie, Paul Pospisil, and Chris Stang for their input to this research. The financial support of the Natural Sciences and Engineering Research Council of Canada and of the University of Saskatchewan are gratefully acknowledged.


## REFERENCES


[Adam 80] A. Adam and J. Laurent, "LAURA, A System to Debug Student Programs", *Artificial Intelligence 15*, 1980, pp. 75-122.

[Anderson 85] J. R. Anderson and B. J. Reiser, "The Lisp Tutor", *BYTE*, April, 1985, pp. 159-175.

[Barrie 88] B. Barrie, *A Model for Strategy Representation and Recognition for an Intelligent Tutoring System*, Master's Thesis, Department of the Computational Science, University of Saskatchewan, in progress.

[Boyer 79] R. Boyer and J. Moore, *A Computational Logic*, Academic Press, Inc. Orlando, Florida, 1979.

[Escott 88] J. Escott, *Similarity-Based Problem Solving and Student Modelling in a Programming Domain*, Master's Thesis, Department of Computational Science, University of Saskatchewan.

[Huang 87] X. Huang, *Finding Language Errors and Program Equivalence in an Automated Programming Advisor*, Master's Thesis, Research Report # 87-13, Department of Computational Science, University of Saskatchewan, September, 1987.

[Johnson 85] W. L. Johnson, *Intention-Based Diagnosis of Errors in Novice Programs*, Ph.D. Thesis, Yale University, May, 1985.

[Lukey 80] F. J. Lukey, "Understanding and Debugging Programs", *Int. J. Man-Machine Studies, 12*, 1980, pp. 189-202.

[McCalla 86] G. I. McCalla R. B. Bunt and J. J. Harms, "The Design of the SCENT Automated Advisor", *Computational Intelligence, vol. 2, no. 2*, 1986, pp. 76-92.

[McCalla 88] G. I. McCalla and J. E. Greer, "Intelligent Advising in Problem Solving Domains", *Proceedings of the Intelligent Conference on Intelligent Tutoring Systems (ITS '88)*, June, 1988, Montreal, (to appear).

[Murray 87] W. R. Murray, "Automatic Program Debugging for Intelligent Tutoring Systems", *Computational Intelligence, vol. 3, no. 1*, February, 1987, pp. 1-16.

[Ng 86] T. H. Ng, *Dynamic Planning of Blackboard Focus Shifts in an Automated Debugging System*, Master's Thesis, Research Report 87-3, Department of Computational Science, University of Saskatchewan, March, 1987.

[Pospisil 88] P. Pospisil, *Diagnosing Strategy Errors in SCENT*, Master's Thesis, Department of Computational Science, University of Saskatchewan, in progress.

[Reiser 85] B. J. Reiser, J. R. Anderson and R. G. Farrell, "Dynamic Student Modelling in an Intelligent Tutor for LISP Programming", *Proceedings of the 9th IJCAI Conference*, Los Angeles, Calif., August, 1985, pp. 8-13.

[Ruth 76] G. R. Ruth, "Intelligent Program Analysis", *Artificial Intelligence 7*, 1976, pp. 65-85.

[Shapiro 81] D. G. Shapiro, *Sniffer: a System that Understands Bugs*, AI Memo #638, MIT, 1981.

[Wertz 82] H. Wertz, "Stereotyped Program Debugging: an Aid for Novice Programmers", *Int. J. Man-Machine Studies 16*, 1982, pp. 379-392.

# KNOWLEDGE ACQUISITION TECHNIQUES FOR KNOWLEDGE-BASED SYSTEMS

Mildred L. G. Shaw
Dept of Computer Science
University of Calgary
Calgary, Alberta
Canada T2N 1N4

## ABSTRACT

Growing recognition of the significance of knowledge-based computing systems has focused attention on processes of knowledge acquisition and transfer. Commercial applications of expert systems are being impeded by the knowledge engineering bottleneck and have led to the development of rapid prototyping tools. These have proved practically useful but the range of application of existing tools is limited, and it is not clear what limitations are inherent in the prototyping tools. This paper introduces the concept of knowledge support systems as the integration of interactive knowledge acquisition systems and expert systems shells. A prototype knowledge support system that has been implemented is described with examples of some of the knowledge acquisition and application tools provided. It includes knowledge acquisition and transfer through interactive repertory grid elicitation; exchange techniques for understanding and agreement between experts; and different forms of analysis that have been widely used for comparing data between experts and for rapid prototyping of expert systems. An evaluation model for knowledge support systems is presented, and the results of initial validation studies are reported showing the extent to which experts agree with each other and with themselves at a later date. Preliminary results indicate that the measures used discriminate the differences within and between experts.

**KEYWORDS:** knowledge acquisition, knowledge support systems, rapid prototyping tools, validation.

## INTRODUCTION

Much successful work has been done in the area of knowledge representation, but knowledge acquisition and transfer has long been thought of as the bottleneck in the process of building expert systems. It was possible that knowledge engineering might develop as a profession on a par with systems analysis and programming, and that an initial shortage of skilled knowledge engineers would cause problems which would be overcome eventually as the profession developed. However, this scenario now appears less and less likely. There is certainly a shortage of knowledge engineers and problems in developing applications, but doubts have been cast on the notion that human labor is the appropriate solution to the knowledge engineering problem.

The technology is now in a mass-market situation where many organizations see the need for expert systems. This has led to a growth in demand that is far more rapid than the growth in supply of trained and experienced knowledge engineers. In addition, the role of the knowledge engineer as an intermediary between the expert and the technology is being questioned not only on cost grounds but also in relation to its effectiveness. Knowledge may be lost through the intermediary and the expert's lack of knowledge of the technology may be less of a detriment than the knowledge engineer's lack of domain knowledge. Full exploitation of the potential of expert systems depends on the development of rapid prototyping systems directly usable by experts with the knowledge engineer acting only as manager, not intermediary.

## PERSONAL CONSTRUCT PSYCHOLOGY

Kelly developed a systemic theory of human cognition based on the single primitive of a construct, or dichotomous distinction. For an individual, constructs are:

> "transparent templets which he creates and then attempts to fit over the realities of which the world is composed." (Kelly 1955)

He proposes that all of human activity can be seen as a process of anticipating the future by construing the replication of events:

> "Constructs are used for predictions of things to come, and the world keeps rolling on and revealing these predictions to be either correct or misleading. This fact provides a basis for the revision of constructs and, eventually, of whole construct systems." (Kelly 1955)

Hence his psychological model of man is strongly epistemological and concerned with the way in which man models his experience and uses this model to anticipate the future. The anticipation may be passive as in prediction, or active as in action.

Kelly developed his theory in the context of clinical psychology and hence was concerned to have techniques which used it to by-pass cognitive defenses and elicit the construct systems underlying behaviour. This is precisely the problem of knowledge engineering. His repertory grid (Shaw 1980) is a way of representing personal constructs as a set of distinctions made about elements relevant to the problem domain. In clinical psychology this domain will often be personal relationships and the elements may be family members and friends. In the development of expert systems the elements will be key entities in the problem domain such as oil-well sites or business transactions.

Repertory grids have been widely used: in clinical psychology (Shepherd & Watson 1982); to study processes of knowledge acquisition in education (Pope & Shaw 1981); and to study decision making by individuals and groups in management (Shaw 1980). PLANET (Shaw 1982) is an integrated suite of programs that operationalizes Kelly's work and may be used for the interactive elicitation and analysis of repertory grids. These programs have been widely used internationally in clinical psychology, education and management studies (Shaw 1981), and in knowledge engineering for expert systems (Shaw & Gaines 1983).

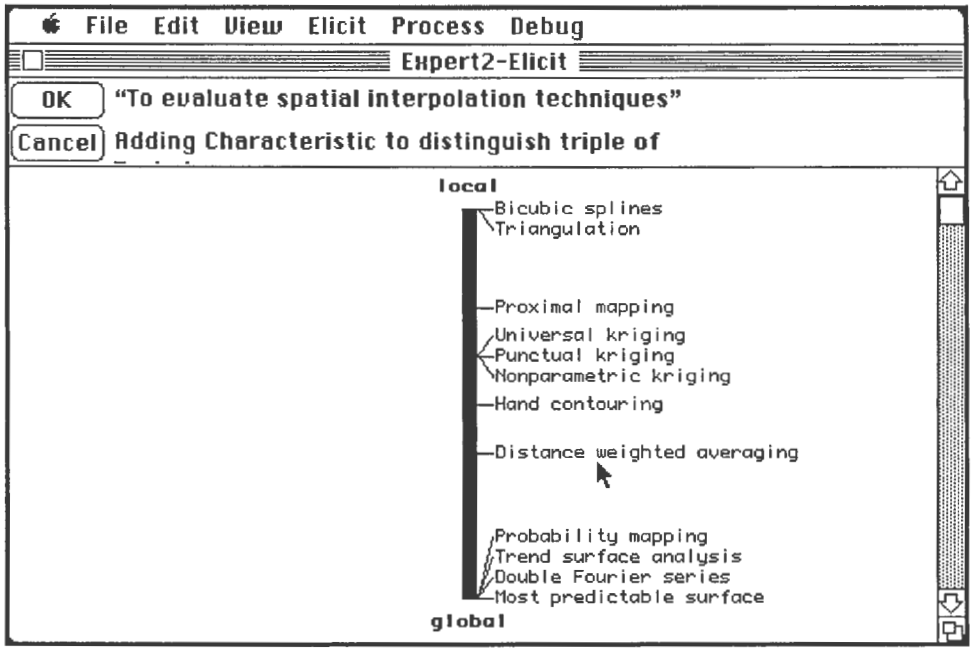



**Figure 1. A KSS0 screen showing the construct local — global.**

Kelly's personal construct psychology is important because it develops a complete psychology of both the normal and abnormal, which has strong systemic foundations. In the long term these foundations may be more important to knowledge engineering than the techniques currently based on them. However, this paper concentrates on the repertory grid as a technique for eliciting information from an expert.

## REPERTORY GRIDS

A repertory grid is a two-way classification of data in which events are interlaced with abstractions in such a way as to express part of a person's system of cross-references between his personal observations or experience of the world (elements), and his personal constructs or classifications of that experience.

The elements are the things which are used to define the area of the topic, and can be concrete or abstract entities. For example, in the context of expertise about metal joining they might be types of rivet, or in expertise about medical diagnosis they might be symptoms. Before choosing the set of elements, the user must think carefully about the area of the topic and relate the elements to his purpose. The elements should be of the same type and level of complexity, and span the topic as fully as possible. It is usual to start with about six to twelve elements. The universe of discourse is determined by the elements which must be significant to the person in the context of the particular problem.

The constructs are the terms in which the elements are similar to or different from each other. Each construct therefore has two poles, each of which has a meaning with respect to its opposite. Any construct or dimension of thinking which is important to the subject is a valid construct. For example, to distinguish between people by saying that x and y are **blue-eyed** whereas b and c are **brown eyed** may be trivial, and not concerned with the important qualities of x, y, b, and c. However, if you are an eye specialist concerned with prescribing tinted contact lenses, this may be a significant construct. Thoughts and feelings, objective and subjective descriptions, attitudes and rules-of-thumb all constitute valid constructs if they contribute to the conceptual structures of the expert. The verbal description of the construct and the labelling of the poles need not be a publically agreed meaning in the outside world, but only a memory aid to the thinking process. The mapping of the elements onto the constructs produces the two-dimensional grid of relationships.

Constructs are elicited either in triples — in what significant way is one element similar to another, and in the same way different from a third; in pairs — in what significant way do these two elements differ; or just as they come to the mind of the expert. An example is shown of elements rated on the construct **local—global** using direct manipulation in the area of spatial interpolation techniques to produce contour maps, in Figure 1.

Figure 2 shows how the techniques are dragged on to the construct bar from the left. Both the construct labels and the technique names can be edited at any time. The placing of the techniques should be continued until the expert feels that their relative positions captures the idea he had in mind.

The mapping of the elements onto the constructs produces the two-dimensional grid of relationships which can be represented as a numeric data structure as shown in Figure 3. This structure may be viewed as a component of a database in entity-attribute form (Chen 1980): a repertory grid has elements as entities, constructs as attributes and allocations of elements to poles of constructs as values. For example, **Bicubic splines** is rated a 1 on the construct **local—global** specifying **local**, whereas **Probability mapping** is rated a 9, specifying **global**.

## KNOWLEDGE SUPPORT SYSTEMS

Knowledge support systems can be seen as integrated interactive knowledge acquisition systems and expert systems shells. They are seen as having a broad scope, encompassing both aids to knowledge acquisition and transfer, knowledge representation, and support of human knowledge processes (Shaw & Gaines 1987). It is desirable that the tools in a knowledge support system are domain independent and directly applicable by experts without intermediaries. They should be able to access a diversity of knowledge sources including text, interviews with experts, and observations of expert behaviour and be able to encompass a diversity of perspectives including partial or contradictory input from different experts. Moreover, the knowledge support system should be able to encompass a

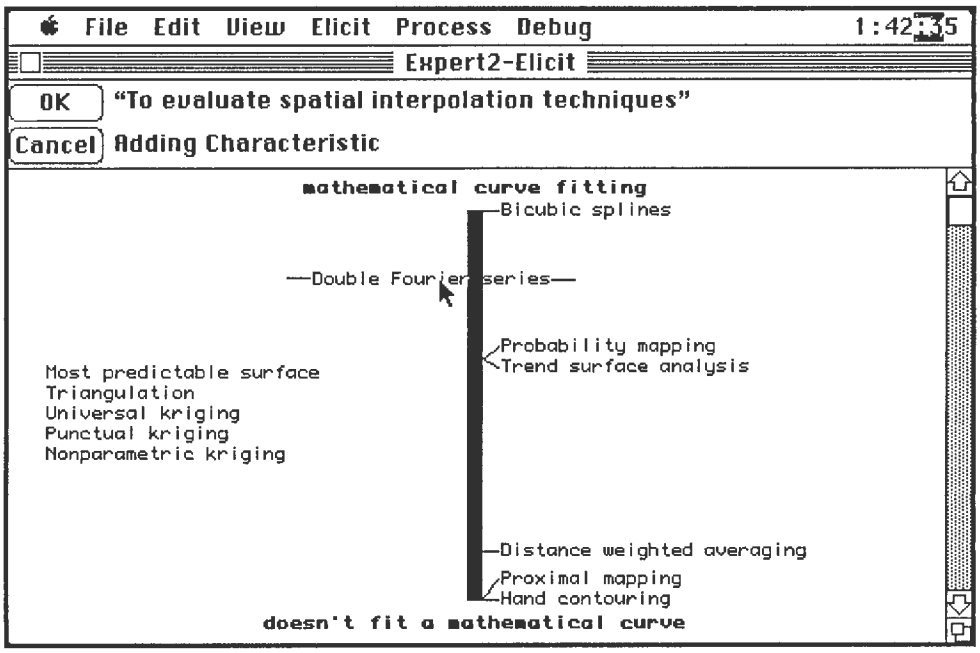



**Figure 2. A KSS0 screen showing direct manipulation of elements.**

Entities: 12, Attributes: 18, Range: 1 to 9, Purpose: To evaluate spatial interpolation techniques

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| qualitative and quantitative | 1 | 8 | 8 | 6 | 9 | 1 | 9 | 8 | 8 | 9 | 4 | 4 | 4 | 1 | quantitative |
| local | 2 | 9 | 9 | 6 | 3 | 5 | 1 | 9 | 9 | 1 | 4 | 4 | 4 | 2 | global |
| autocorrelation not considered | 3 | 1 | 1 | 5 | 4 | 7 | 2 | 3 | 1 | 2 | 9 | 9 | 9 | 3 | autocorrelation considered |
| doesn't honour data points | 4 | 2 | 1 | 3 | 5 | 9 | 9 | 1 | 1 | 9 | 8 | 7 | 7 | 4 | honours data points |
| multiple variables considered | 5 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 1 | 9 | 9 | 9 | 9 | 5 | usually one variable considered |
| mathematical curve fitting | 6 | 4 | 4 | 8 | 9 | 9 | 1 | 2 | 4 | 9 | 5 | 8 | 8 | 6 | doesn't fit a mathematical curve |
| nonparametric | 7 | 9 | 8 | 6 | 1 | 1 | 3 | 6 | 8 | 3 | 8 | 8 | 1 | 7 | parametric |
| interval or ratio data | 8 | 9 | 1 | 1 | 5 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 | nominal data |
| requires periodicities | 9 | 6 | 6 | 9 | 9 | 9 | 9 | 1 | 6 | 9 | 9 | 9 | 9 | 9 | doesn't require periodicities |
| doesn't fit a trend | 10 | 9 | 9 | 1 | 1 | 8 | 1 | 7 | 9 | 1 | 6 | 1 | 1 | 10 | fits a trend to the data |
| heavy computing load | 11 | 7 | 6 | 7 | 8 | 9 | 4 | 4 | 5 | 3 | 1 | 2 | 3 | 11 | no computing load |
| assumes isotropic surface | 12 | 1 | 1 | 4 | 3 | 8 | 9 | 1 | 1 | 9 | 7 | 6 | 6 | 12 | assumes anisotropic surface |
| not as susceptible to clusters | 13 | 8 | 8 | 6 | 9 | 3 | 4 | 7 | 8 | 5 | 1 | 2 | 2 | 13 | estimates susceptible to clusters |
| doesn't incorporate geologic model | 14 | 2 | 2 | 3 | 1 | 9 | 1 | 2 | 2 | 1 | 6 | 6 | 6 | 14 | incorporates geologic model |
| interpretive | 15 | 9 | 9 | 5 | 9 | 1 | 7 | 9 | 9 | 7 | 3 | 4 | 4 | 15 | representative |
| not very important | 16 | 4 | 5 | 2 | 1 | 9 | 1 | 1 | 6 | 7 | 8 | 9 | 9 | 16 | very important |
| not very effective | 17 | 4 | 5 | 3 | 1 | 9 | 2 | 4 | 6 | 7 | 9 | 8 | 8 | 17 | very effective |
| not widely used | 18 | 3 | 8 | 7 | 5 | 9 | 3 | 3 | 4 | 6 | 2 | 2 | 1 | 18 | widely used |

1 Probability mapping
2 Trend surface analysis
3 Distance weighted averaging
4 Proximal mapping
5 Hand contouring
6 Bicubic splines
7 Double Fourier series
8 Most predictable surface
9 Triangulation
10 Universal kriging
11 Punctual kriging
12 Nonparametric kriging

**Figure 3. A conventional repertory grid of spatial interpolation techniques. A 1 represents the left construct pole and a 9 the right.**

diversity of forms of knowledge and relationships between knowledge and be able to present knowledge from a diversity of sources with clarity as to its derivation, consequences and structural relations. Users of the knowledge support system should be able to apply the knowledge in a variety of familiar domains and freely experiment with its implications. It should make provision for validation studies and as much of the operation of the knowledge support system as possible should be founded on well-developed and explicit theories of knowledge acquisition, elicitation and representation. As the overall knowledge support system develops it should converge to an integrated system.

Clearly, these are general requirements and may not apply in particular cases. For example, some domain dependence may be appropriate for efficiency in specific knowledge support systems; some human intervention may be helpful or necessary when an expert is using a knowledge support system. However, in general these requirements capture the key issues in knowledge support system design.

PLANET (Shaw 1982, Shaw & Gaines 1986) has been developed and widely used for some years for knowledge elicitation. This paper briefly reviews the techniques involved and describes the most recent developments of a comprehensive integrated rapid prototyping system, KSS0, which attempts to satisfy the above design requirements for a knowledge support system.

## RELATED WORK

ETS and Aquinas (Boose 1984, Boose & Bradshaw 1987) are related quite closely to PLANET and KSS0, having been developed in response to the initial success of the PLANET system. KSS0 and Aquinas are integrated prototyping systems providing knowledge acquisition tools encompassing a diversity of forms of knowledge and relationships between knowledge. KSS0 can access a wide range of knowledge sources including text, interviews with experts, and observations of expert behaviour. Aquinas can present knowledge from multiple sources with clarity as to its derivation, consequences and structural relations. Both systems can encompass a diversity of perspectives including partial or contradictory input from different experts. Users of these knowledge acquisition tools are able to apply the knowledge in a variety of familiar domains and freely experiment with its implications. These systems offer the capability to expedite the prototyping stage of complex knowledge-based systems, motivating the experts to be closely involved in all aspects of the system development by giving them a supportive and comprehensible environment. All four systems use repertory grid techniques to elicit the expert's conceptual structures, and use different versions of ENTAIL (Gaines & Shaw 1986) to produce rules for an expert system shell.

MOLE (Eshelman, Ehret, McDermott, & Tan, 1987) is an expert system shell that has been used effectively to build heuristic classification systems. It interviews domain experts, exploits assumptions of exhaustiveness and exclusivity to determine the most likely candidates among competing hypotheses, and has many methods for analyzing and refining the knowledge base for consistency and adequacy. It distinguishes between three types of knowledge—covering, differentiating, and combining knowledge—in its problem-solving, and uses a system of "support values" to represent uncertainty.

CSRL (Bylander & Mittal, 1986) is another successful tool with a similar problem-solving approach, although it is difficult for domain experts to learn and use. Gruber and Cohen (1987) have developed a system called MUM that embodies architectural principles that facilitate knowledge acquisition. It allows experts to specify local evidence combining knowledge; however it is limited to a symbolic representation of uncertainty.

An innovative "learning apprentice" program, ODYSSEUS, has been developed by Wilkins (1987) to help experts refine and debug knowledge bases for the HERACLES heuristic classification shell. However, its use is appropriate only during the knowledge acquisition "end-game"; that is, it requires that a reasonable knowledge base has already been created.

The following section describes the work on KSS0, a knowledge support system that draws on many concepts and techniques for knowledge engineering to begin to encompass some of the requirements discussed above, attempting to relate them all through the theoretical base of personal construct psychology, and build a workbench of tools around a common database.

## KSS0: A KNOWLEDGE SUPPORT SYSTEM

KSS0: Knowledge Support System Zero consists of a: knowledge base; various analytical tools for building and transforming the knowledge base; and a number of conversational tools for interacting with the knowledge base. The KSS0 implementation is written in Pascal and currently runs on a network of Macintosh workstations. The KSS0 structure is best understood by following sequences of activity that lead to the generation of a rule base and its loading into an expert system shell.

A typical sequence is text input followed by text analysis through TEXAN which clusters associated words leading to a schema from which the expert can select related elements and initial constructs with which to commence grid elicitation. The resultant grids are analyzed by ENTAIL which induces the underlying knowledge structure as production rules that can be loaded directly into an expert system shell (Gaines & Shaw 1986).

An alternative route is to monitor the expert's behaviour through a verbal protocol giving information used and decisions resulting and analyze this through ATOM which induces structure from behaviour using a search over a model space ordered by complexity and goodness of fit, and again generates production rules (Gaines 1977). A version of ATOM is incorporated in KSS0 that takes a set of sequences of arbitrary symbolic data and generates a set of production rules that will reconstruct it. These can be loaded into the expert system shell to give a simulator of the behavioural system.

These two routes can be combined. KSS0 attempts to make each stage as explicit as possible to the expert, and, in particular, to make the rule base accessible as natural textual statements rather than technical production rules. The expert system shell being used in KSS0 currently is Nexpert (Roy 1986) which gives a variety of textual and graphical presentations of the rule base enabling the expert to see the impact of different fragments of knowledge.

The group problem-solving component of KSS0 is particularly important because it goes beyond the stereotype of an **expert** and **users**, and allows the system to be used to support an interactive community in their acquisition and transfer of knowledge and mutual understanding. The SOCIO analysis allows members of a community to explore their agreement and understanding with other members, and to make overt the knowledge network involved (Shaw 1980, 1981, 1988). Much expertise only resides within the social context of cooperating individuals and requires elicitation across the group. The SOCIO analysis program supports group elicitation techniques in which the construct systems of a number of experts are compared. Grids are elicited separately but then exchanged in two ways: an expert can place elements on a colleague's constructs from his own point of view, and the analysis system then allows him to explore their agreement; or he can attempt to place them from his colleague's point of view and hence explore his understanding. SOCIO supports several users in seeing the relationship of their points of view to those of others; exploring

differing terminology for the same constructs; becoming aware of differing constructs having the same terminology; extending their own construct systems with those of others; providing others with constructs they have found valuable; and exploring a problem-solving domain using the full group resources.

The KSS0 implementation is an initial prototype offering a workbench with minimal integration of the knowledge base, but each of the tools has already proven effective, and their combination is proving very powerful in stimulating experts to think of the knowledge externalization process from a number of different perspectives.

## VALIDATION OF A KNOWLEDGE SUPPORT SYSTEM

A model for evaluation of a knowledge support system and different types of validation has been discussed by Shaw and Woodward (1988). There are two distinct aspects of validation, those concerned with objective aspects of the system and those concerned with subjective aspects of the system. These distinctions represent the various aspects of operation of knowledge support systems. The objective aspects are those concerning the performance of the expert system or knowledge base, elicited from the expert or experts, against objective criteria. A knowledge support system supports the knowledge processes of the expert, and allows her to develop a system which actually performs effectively in the real world, rather than merely confirming the opinion of the expert.

The subjective aspects relate to the way in which the knowledge acquisition system acquires the knowledge that the expert has. It is clearly impossible to separate the knowledge acquisition tool from the expert, so the term **knowledge acquisition system** is used to mean the knowledge acquisition tool, **and** the expert:

**knowledge acquisition system = knowledge acquisition tool + expert**

Figure 4 shows the model developed for this study. Only the subjective aspects of validation were investigated.

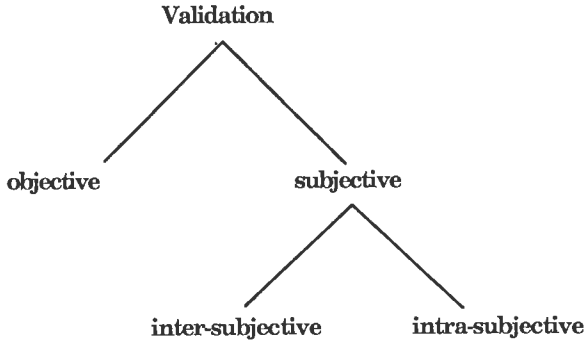



**Figure 4. The evaluation model for the study**

It is now thought to be essential to have a group of experts involved in building a knowledge base, so it was necessary to see if there was mutual consistency and/or construing across experts. Also, whether there was any consistency of an expert over a time interval, both in the terminology used, and how it was used. Finally, an investigation was made as to whether the analysis performed by the system on each expert's knowledge was seen as correct, or if any of it was incorrect.

A measure of diachronic reliability (Chow, 1987) appears most suitable for evaluating the **replication** of knowledge support systems. This measure reflects the degree of similarity, over time, of the calculated output. A knowledge support system reflects a high degree of replication in the sense that the output of the knowledge support system, the results of the algorithms, reflect a high degree of similarity over time. Strictly speaking, this approach to replication is strongly influenced by the reliability of the user of the knowledge support system. However, the knowledge support system can be viewed as an aid to the user which helps the user maintain consistency over time given the same task demands. Also, knowledge support systems are not used in isolation but are used by individuals so any measure of a knowledge support system must be done with those who will use the system. If replication measures are low, the system cannot be considered consistent or helpful, unless this can be seen to be a product of the raw data from the experts.

Although **replication** is a possible requirement of a knowledge support system, other basic requirements need attention. The knowledge support system must also demonstrate various forms of **consistency**. In this sense, consistency refers to the extent to which the knowledge support system produces what it is expected to produce. At the subjective level of evaluation, consistency is defined by two measures. The first is **intra-subjective consistency**. Based on the concepts of content and face validity (Travers, 1969; Campbell & Stanley, 1963), this type of consistency is defined as the extent to which the knowledge support system output is correct as judged by the expert. Knowledge support systems produce output which has a high degree of similarity with what is apparently expected. The absence of this form of consistency indicates that the system may be performing its calculation incorrectly and/or that the output bears little resemblance to what would be expected by the designer, by the user of the tool or by the domain expert. This result has serious implications for any subsequent use of knowledge support system output.

Another, complementary, measure of consistency at this level of evaluation is that of **inter-subjective consistency**. Based on the concept of **concurrent validity** (Chow, 1987; Travers, 1969) this type of consistency is a measure of prediction of events which occur at roughly the same time. If two or more individuals, with similar experience in an area, produce similar output using a knowledge support system, then the system can be said to afford a degree of inter-subjective consistency. The extent to which two or more people produce the the same output represents the degree of consistency.

However, inconsistency among experts is not unexpected as experts may well not agree. In many areas, each expert has a unique perspective on the topic and often believes that there is only one way that the topic should be considered. This is not necessarily an undesirable situation, as long as the system is able to reflect this. If a task is carefully defined, specifically outlined and agreed upon as important, then a certain degree of similarity can be expected in the outcome of the knowledge support system. This suggests the selection and development of a set of test cases with specific task requirements. These test cases can then be used to determine the degree of inter-subjective consistency demonstrated by various experts, while using the system. In the same way as for replication, the knowledge support system can be seen as an aid to users.

## THE RESULTS

KSS0 has been evaluated against the model and the results reported in the two domains of spatial interpolation techniques to produce contour maps and in trouble-shooting and maintenance of valves for oil and gas pipelines. Some preliminary results are described on validation experiments to show the extent to which this system can replace standard interviewing techniques.

Two separate studies were completed using specific task demands from these two separate and quite different domains. The first study involved three researchers in a university geography department each of whom is an expert in the field of spatial interpolation techniques to produce contour maps. In the second study, a trainer and an engineer in the field of gas pipeline valve maintenance provided their expertise. These two

individuals were chosen because they were both considered knowledgeable in the area, but also because they brought a different background to the field: one from a field-operator perspective, the other from that of an engineer. The trainer had been involved in training gas pipeline operators in proper maintenance of pipeline valves for 15 years and the engineer owned a company which sold valves. Each group elicited and exchanged grids on the first occasion, then 2 to 3 months later elicited new constructs and ratings using the old set of elements, and elicited new ratings using the old set of elements and constructs. Details of the design are given in Shaw and Woodward (1988).

Two measures were used in this study, **consistency-with** another and **construed-by** another. **Consistency** is measured using **exchange grids**. Do the experts see the topic in a similar way at the same time? By exchanging elements and constructs they are able to view the topic from the perspective of the other. To do this involves an understanding of what the other's elements and constructs actually **mean** in the domain, or **can I construct a point of view which makes sense of them?** Not only that, but **do I agree with the other's construction of the topic, having understood the perspective of the other?** So **consistency** is defined as the degree of match between one expert's ratings of his/her own elements and constructs and another expert's ratings of the first expert's elements and constructs. The patterning of the elements on a particular construct is matched against the patterning of the same elements on that same construct in the other grid.

**Consistency over time** can also be measured in this way but using the **same** expert's grid on the two occasions with the same element and construct labels as before, but new ratings. Only elements and constructs with the same labels in both grids can contribute to this measure of consistency. Operationally, this measure was calculated using the EXCHANGE measure based on the matching algorithm that is described in Shaw (1980) and used in FOCUS, CORE and SOCIOGRIDS. It ranges from 100 for perfect match to 0 for maximum reversal. This measure is picking out the differing use of terminology in the two grids. The following formula was used:

$$\textbf{G consistency-with G' at criterion} = 100 \times \frac{\text{(number of constructs in G matched greater than criterion by } \underline{\textbf{same}} \text{ constructs in G')}}{\text{(number of constructs in G)}}$$

**Construing** is akin to consistency. The entity labels must be the same in both grids, but separate sets of constructs can be used. In this case, the measure picks out the constructs in the second grid with the best match to the patterning of the elements on the one being considered in the first grid. In this case, the same construct may be chosen more than once in the second grid if it matches more closely than any other to more than one construct in the first. Again, the same matching algorithm was used. This measure is picking out the best match between the constructs in the two grids, regardless of terminology. The following formula was used:

$$\textbf{G construed-by G' at criterion} = 100 \times \frac{\text{(number of constructs in G matched greater than criterion by } \underline{\textbf{any}} \text{ constructs in G')}}{\text{(number of constructs in G)}}$$

Operational definitions for the various construing and consistency measures were used to evaluate the knowledge support system. Intra-subjective understanding was defined as the degree of construing of the earlier grid by the later one. Those constructs which were matched on the two grids at a criterion of 80 were used for this calculation, and this was calculated for each expert. The results ranged from 63 to 86 in the first study, and 83 to 100 in the second. This difference in values between the two studies may reflect the fact that the three experts in the first study are academic researchers at a university, and hence tended to be more provisional, exploratory and accepting of different ways of considering the problem. They were at no time fixed in their views of the topic. In the second study, however, the experts were professionals using their expertise in the field, and hence much more decisive and sure of what they were doing and how it should be done.

Intra-subjective agreement was defined as the degree of consistency between one expert's ratings of his/her own elements and constructs in the earlier grid with his/her ratings of the elements and constructs in the later grid. These scores ranged in the first study from 79 to 94, indicating that the experts were using their terminology in much the same way from one occasion to the other and that KSS0 was able to reflect this consistency. In the second study both were 67, indicating some difference in the use of terminology.

Inter-subjective agreement was defined as the degrcc of consistency between one expert's ratings of his/her own elements and constructs in the first grid with another expert's ratings of the first expert's elements and constructs; that is the extent to which the experts agree. These scores ranged in the first study from 8 to 33.showing that the experts disagree in their terminology and in how they view the topic quite extensively. This can also be seen by inspecting the raw data. In the second study the scores range from 13 to 100. Here it is interesting to note that the trainer could use the terminology of the engineer totally, but the engineer could only use a small part of the trainer's terminology.

Inter-subjective understanding was defined as the degree of construing of each expert's grid by another expert. These values are much higher showing that the experts have similar distinctions about the topic even though they differ greatly in how they use the terminology. In the first study these values range from 31 to 71, and in the second from 67 to 100. This suggests that KSS0 shows a facility for allowing experts to compare their understanding and to determine a level of consistency. Whether the threshold criteria are high enough to produce the required level of knowledge-base performance or too high to reach the criterion of completeness is a matter for further study.

Finally, intra-subjective perspective consistency was defined as the degree of consistency of the rules produced by the ENTAIL algorithm with what the expert apparently expected. The ratings of these statements were then compared to the entailment output from the earlier grid. These show whether the expert finds the rules meaningful, and rates the rules as correct or significant in the same way as the ENTAIL algorithm (Gaines & Shaw 1986, Shaw & Gaines 1987). They range from 82 to 92, showing a high level of expected rules appearing in the KSS0 output. This indicates a good basis for using the ENTAIL produced rules as input to an expert system shell.

There are many more ways in which the data are being processed, especially in terms of the rules derived from the various grids. At this stage the similarities and differences between experts and within experts over a 2 to 3 month time interval have been investigated. Clearly, the content of these similarities and differences can be identified and related to external criteria.

## CONCLUSIONS

Growing recognition of the significance of knowledge-based computing systems has focused attention on processes of knowledge acquisition and transfer. Commercial applications of expert systems are being impeded by the knowledge engineering bottleneck and have led to the development of rapid prototyping tools. These have proved practically useful but the range of application of existing tools is limited, and it is not clear what

limitations are inherent in the prototyping tools, what arises from the shells and their knowledge representation and inferencing procedures, and what arises from our lack of understanding of the underlying processes of knowledge acquisition and transfer.

This paper has introduced the concept of knowledge support systems as integrated interactive knowledge acquisition systems and expert systems shells. A prototype knowledge support system that has been implemented on a network of Macintosh computers has been described with examples of some of the knowledge acquisition and application tools provided. It includes knowledge acquisition and transfer through interactive repertory grid elicitation; exchange techniques for understanding and agreement between experts; and different forms of analysis that have been widely used for comparing data between experts and for rapid prototyping of expert systems. Such systems offer the capability to expedite the prototyping stage of complex knowledge-based systems, motivating the experts to be closely involved in all aspects of the system development by giving them a supportive and comprehensible environment.

An evaluation model for knowledge support systems was presented, and the results of initial validation studies reported. Measures have been put forward which tease out differences in terminology between experts, and show the content of this difference. To a knowledge engineer, it seems that if experts talk about their topic of expertise in the same terms then they mean the same thing, and if they talk in different terms they mean different things. However, it can be seen from the results of this study that the knowledge engineer, or even another expert, may be mistaken. The experts in the first study were clearly accustomed to disagreeing on terminology, a fact which may not always be made explicit. The consistency measure used for inter-subjective agreement showed up the divergence in use of terminology between the experts (8 to 33), but when used for one expert alone for intra-subjective agreement gave much higher values (79 to 94). This measure, then, shows a good range of discrimination in the two cases within experts and between experts. This is not so apparent in the second study where the experts were professionals using their skills and knowledge every day in an industrial situation. A similar rationale can be applied in the construed-by measures of inter- and intra-subjective understanding.

In the first study there was clearly inconsistency among the experts, and this was reflected by the results from the knowledge support system. This was therefore a good population for study, as it highlighted a controversy and/or disagreement which would be likely to occur between experts and users of an expert system, even if it did not occur among the experts themselves.

## ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. The KSS0 system is made available by the Centre for Person Computer Studies. I would like to thank my colleagues Brian Gaines and Brian Woodward who were jointly involved with me in conducting this research.

## REFERENCES

Boose, J.H. (1984). Personal construct theory and the transfer of human expertise. **Proceedings AAAI-84**, 27-33. California: American Association for Artificial Intelligence.

Boose, J.H. & Bradshaw, J.M. (1987). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies, 26**(1), 3-28.

Bylander, T. & Mittal, S. (1986). CSRL: A language for classificatory problem-solving and uncertainty handling, **AI Magazine** (August).

Campbell, D.T. & Stanley, J.C. (1963) **Experimental and Quasi-Experimental Designs for Research.** Chicago:Rand McNally and Company.

Chen, P.P., Ed. (1980). **Entity-relationship approach to systems analysis and design**. New York: North Holland.

Chow, S.L. (1987). **Experimental Psychology: Rationale, Procedures and Issues**. Calgary: Detselig Enterprises.

Eshelman, L., Ehret, D., McDermott, J. & Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool. **International Journal of Man-Machine Studies, 26**(1), 41-54..

Gaines, B.R. (1977). System identification, approximation and complexity. **International Journal of General Systems, 3**, 145-174.

Gaines, B.R. (1987) Rapid prototyping for expert systems. Oliff, M. (Ed). **Proceedings of International Conference on Expert Systems and the Leading Edge in Productions, Planning and Control.** pp. 213-241. University of South Carolina.

Gaines, B.R. & Shaw, M.L.G. (1981). New directions in the analysis and interactive elicitation of personal construct systems. Shaw, M.L.G., Ed. **Recent Advances in Personal Construct Technology**. pp. 147-182. London: Academic Press.

Gaines, B.R. & Shaw, M.L.G. (1986). Induction of inference rules for expert systems. **Fuzzy Sets and Systems**, 8(3), 315-328 (April).

Gruber, T. & Cohen, P. (1987). Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. **International Journal of Man-Machine Studies, 26**(2), 143-159.

Johnson, P.E. (1986).Cognitive models of expertise. **Symposium on Expert Systems and Auditor Judgment.** University of Southern California.

Kelly, G.A. (1955). **The Psychology of Personal Constructs**. New York: Norton.

Landfield, A. (1976). A personal construct approach to suicidal behaviour. Slater, P., Ed. **Dimensions of Intrapersonal Space: Volume 1**. pp. 93-107. London: John Wiley.

Pope, M.L. & Shaw, M.L.G. (1981). Personal construct psychology in education and learning. Shaw, M.L.G., Ed. **Recent Advances in Personal Construct Technology**. pp. 105-114. London: Academic Press.

Roy, J. (1986). Expert systems in Nexpert. **MacTutor, 2**(2), 48-51 (February).

Shaw, M.L.G. (1980). **On Becoming a Personal Scientist**. London: Academic Press.

Shaw, M.L.G. (1981). **Recent Advances in Personal Construct Technology**. pp. 31-44. London: Academic Press.

Shaw, M.L.G. (1982). PLANET: some experience in creating an integrated system for repertory grid applications on a microcomputer. **International Journal of Man-Machine Studies, 17**(3), 345-360.

Shaw, M.L.G. (1988). An interactive knowledge-based system for group problem-solving. **IEEE Transactions on Systems, Man and Cybernetics,** (to appear).

Shaw, M.L.G. & Gaines, B.R. (1980). Fuzzy semantics for personal construing. **Systems Science and Science**. pp. 146-154. Kentucky: Society for General Systems Research.

Shaw, M.L.G. & Gaines, B.R. (1983). A computer aid to knowledge engineering. **Proceedings of British Computer Society Conference on Expert Systems,** 263-271 (December). Cambridge.

Shaw, M.L.G. & Gaines, B.R. (1986a). Interactive elicitation of knowledge from experts. **Future Computing Systems, 1**(2), 151-190.

Shaw, M.L.G. & Gaines, B.R. (1986b). Techniques for Knowledge Acquisition and Transfer. **Proceedings of the Workshop on Knowledge Acquisition for Knowledge-Based Systems,** November, Banff, 39-0 — 39-13.

Shaw, M.L.G. & Gaines, B.R. (1987). Techniques for knowledge acquisition and transfer. **International Journal of Man-Machine Studies, 27**(3), 251-280.

Shaw, M.L.G. & Woodward, J.B. (1988) Validation in a knowledge support system: consistency and.construing with multiple experts. **International Journal of Man-Machine Studies,** in press.

Shepherd, E. & Watson, J.P., Eds. (1982). **Personal Meanings**. London: John Wiley.

Travers, R.M.V. (1969) **An Introduction to Educational Research (Third Edition).** MacMillan: London.

Wilkins, D.C. (1987). Knowledge base debugging using apprenticeship learning techniques. **International Journal of Man-Machine Studies,** 27(3), 281-294.

# Extracting Rules from Data with Exceptions

Toshiharu Sugawara
NTT Software Laboratories
3-9-11, Midori-cho, Musashino-shi, Tokyo, 180, Japan
Phone: +81 422 59 3585

## ABSTRACT

In this paper a method of extracting typical properties from a set of data containing exceptions is discussed. An observational function classifies a sample, removes the minority elements, and extracts a general rule based on the remaining elements. Furthermore, the extracted rule is improved by statistical estimation when new data are added. This technique is applied to a system for aiding the construction of knowledge bases.

Keywords: Knowledge aquisition, Default rule, Learning

## §1. Introduction

Extracting a general characteristic or a rule from a set of facts is the primary motivation behind this research. In real situations, a set of facts often contains elements which are exceptions to possible general rules. Nevertheless the emphasis is placed on extracting typical properties that characterize the set. For example, the general rule "birds typically fly" might be extracted from a set of facts containing "canaries fly," "sparrows fly," "penguins do not fly," "swallows fly," and so on. This may be considered as an extraction of default rules and such an extracted rule is a common-sense rule.

A number of studies on theories and application systems for extracting a general rule from a set of facts have been made ([2, 4 and 9]). For example, in M. Gold's study, an inductive inference machine (Turing machine) is employed. All rules (or functions) are enumerated within it. The machine verifies each rule with the facts entered from an external source, and outputs a rule which is completely consistent with them as a guess. References [10 and 12] are well-known systems that have applied this theory. Unfortunately, none of these methods or systems are capable of extracting a rule if there is an exception in the data entered. This is a serious shortcoming that renders all these systems ineffective for most practical applications.

Exceptions here are defined as data that are proportionally small compared to the total data within an actually observed range, or those that constitute a relatively small proportion compared to the majority elements. It is sage to say "birds typically fly" because we know after observing the number of birds which do not fly is relatively small in proportion to those that do fly. Default rules that are used as examples today can all be explained from this perspective.

Observations are not finished in a specified amount of time. There is always a possibility that a new fact may be entered from an external source. This view point is the same as in Gold's inductive inference model. In other words, the extracted rules may be modified when a new fact is entered. Therefore, the rule extracting process is infinite.

In this paper, a specific property rule is extracted by the following method. Functions $f_1, \ldots, f_n$ are prepared to observe the properties of facts (these functions are called observational functions), and a set of facts $\{d_1, \ldots, d_n\}$ (this is called a sample) is classified by the value of $f_i$. However, the number of $f_i$ calculations here must be sufficiently small. Through this process, exceptions are eliminated by the constraints of the cardinality of sets and the value of $f_i$ is estimated from the remaining data. This method may not produce rules that precisely define every aspect of a set, but it meets the requirements of most practical situations by providing appropriate observational functions.

Points that need to be clarified for the extraction of default rules are as follows:

(1) Extracting rules,
(2) Identifying exceptions,
(3) Maintenance of extracted rules.

(1) is an identification process to determine if any default rule can be extracted when a set $\{d_1, \ldots, d_n\}$ is classified by $f_i$. (2) is the issue of to what degree minorities should be treated as exceptions and (3) is the issue of how to modify a rule when new data are added when the sample becomes biased and the extracted rule loses generality. These issues, illustrated with examples, will be the primary focus of this paper.

We have applied the methods based on these issues to develop a system to aid the construction of frame-based knowledge bases (KB-CAS). Normally, knowledge base construction starts with the creation of a small-scale knowledge base first. Then data are added to this step by step. KB-CAS uses the initial knowledge base as a sample and extracts rules or similarities; for example, rules regarding slot values, reverse pointers and IS-A hierarchy. The extracted rules assist in the construction of the knowledge base and improve the efficiency of the construction process.

## §2. Related Works

From a logical viewpoint, default logic introduced by Reiter [11] is one of the notable works for representing common-sense rules. Default rules are expressed as follows:

$$\frac{A(x) : \mathrm{M}B(x)}{C(x)}$$

This means if $A(x)$ is true and $\neg B(x)$ cannot be derived, then $C(x)$ is true. Especially in normal case, where $B = C$, it has been proved that a consistent extension of default theory exists. There are other studies which support this position, such as non-monotonic logic [6] and circumscription [5].

In Variable Precision Logic, introduced by Michalski and Winston [8], rules containing exceptions are expressed by Censored Production Rules [14], which are formed as follows:

$$S \Rightarrow D \lfloor C \quad r \qquad (1 \geq r \geq 0.5)$$

This means "if S then D unless C," and if S, the possibility that C is true is $r$ and D is true is $1 - r$. Exceptions are described by C. If "S is true and C is unknown," D can be derived using inference rules which are defined by variable precision logic, so this rule employs a default rule using probability. The following is an example from their paper.

S: it is Sunday
D: john works in the yard
C: he reads a book
$r >> 0.5$

In variable precision logic, this rule means "if it is Sunday, John usually works in the yard."

The approach in this paper, unlike in these studies, is intended to extract rules and mainly to modify previously extracted rules. For example, after the rule extraction process, if C is observed many times, then C is no longer an exception. In an extreme case, instead of C, D becomes an exception. It is natural that the rule is updated to

$$S \Rightarrow D \vee C \quad r' \qquad \text{or} \qquad S \Rightarrow C \lfloor D \quad r''$$

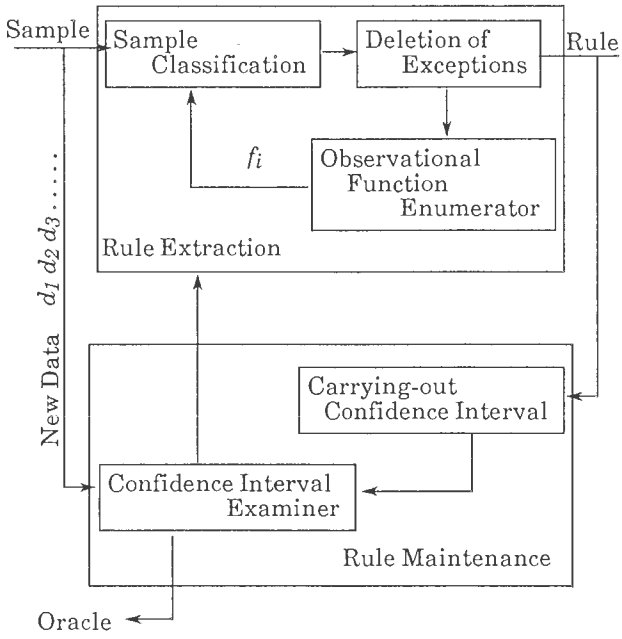



Fig. 1. Rule Extraction Model

The main problem is to decide when to rewrite or abandon the rules.

Conceptual clustering is also related to this study ([3 and 7]). It is a method for producing classification schemes from observed objects. If an unknown object can be observed, the class to which it may belong can be determined. However the extracted default rules in this study are used as follows: the class which a object belongs to is known, but its properties cannot be observed. Nonetheless the properties that it may have can be inferred with a high degree of probability.

## §3. Rule Extraction Model

A rule extraction model is shown in Fig. 1. When a sample is entered, the appropriate observational function is enumerated and the sample is classified according to the value of the function. The minority part is deleted as an exception and the remaining portion is extracted as a rule. Meanwhile, the confidence level of the extracted rule is computed using a statistical method (interval estimation).

When new data are entered, the confidence interval examiner asks an oracle. whether the rule is still applicable to the new data. Depending on the result given, the confidence of the rule is tested by the examiner. If necessary, it is referred back to the rule extraction section for extraction of a new rule. This process is described in some detail below.

## §4. Rule Extraction Method

### Definition

Assume that $A$ is a set and $S$ is defined as a finite subset of $A$, and is hereafter referred to as "the sample." Let $f : A \rightarrow V$ denote an observational function, and let $f(S)$ be the image of $S$ by $f$. For $v \in f(S)$, we will express $f^{-1}(v)$ as $\{s \in S : f(s) = v\}$. We denote $A(x)$ if $x \in A$. In this paper, the default rule is expressed as follows:

$$A(x) \Rightarrow B(x),$$

where $A$ and $B$ are sets. This expression means that if $A(x)$ then usually $B(x)$.

Classifying $S$ by the value of $f$ means deriving a set

$$K = \{f^{-1}(v) | v \in f(S)\}$$

A sequence of $K$ elements sorted in the order of magnitude of cardinal numbers from the largest will be set as $D_1, \ldots, D_u$. Here, the first m $D_i$ $(m \leq u)$ will be retained, and the remaining sections are discarded as exceptions according to the specified method (this method is discussed in Section 5). Consequently, the following default rule can be extracted:

$$A(x) \Rightarrow D_1 \cup \ldots \cup D_m(x).$$

If $A(x)$ and $D_1 \cup \ldots \cup D_m(x)$ are not derived contradictions, then we can assume $D_1 \cup \ldots \cup D_m(x)$. This corresponds to Reiter's normal default rule.

### Example 1

Assume that the partial logic circuit shown in Fig. 2 can be expressed by the frames shown in Fig. 3-1 and Fig. 3-2. Suppose that $A$ is a set of individual frames whose parent frame is CONNECTION (in complete knowledge base) and $S$ is a set of individual frames whose parent frame is CONNECTION in the knowledge base that represents only Fig. 2. Let $f_p$ denote the observational function on A and let $f_p(d)$ be the parent frame of frame $d$ indicated by the value of slot OUTPUT for any $d \in A$. In this case,

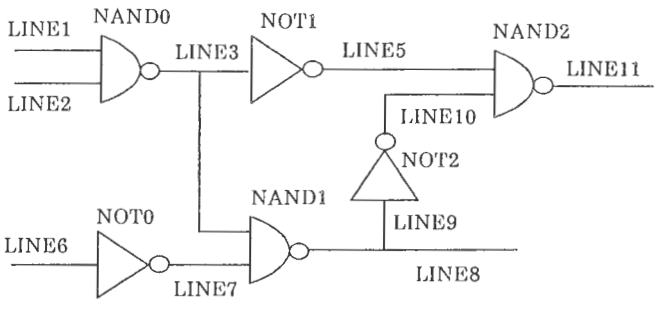



Fig.2. An Example Logic Circuit

$$f_p(S) = \{\text{NAND}, \text{NOT}\}$$

and $K$, $D_1$ and $D_2$ can be expressed as

$K = \{$ {LINE1, LINE2, LINE4, LINE5, LINE7, LINE10},
{LINE3, LINE6, LINE9} $\}$
$D_1 =$ {LINE1, LINE2, LINE4, LINE5, LINE7, LINE10}
$D_2 =$ {LINE3, LINE6, LINE9}

When all sets through $D_2$ are used, the value of $f_p(a)$ can be estimated as NAND or NOT for $a \in A$. In other words, the value can be estimated by "the value of slot OUTPUT in an individual frame, that represents a connection, is an individual frame of either NAND or NOT."

### Example 2 (Reverse Pointer)

A slot whose value is another frame denotes a pointer for that frame. Assume $A$ and $S$ are the same as those in example 1. Suppose $f_{rp}$ is a observational function on $A$ and $f_{rp}(d)$ is a set of slots whose value is $d$, in the frame

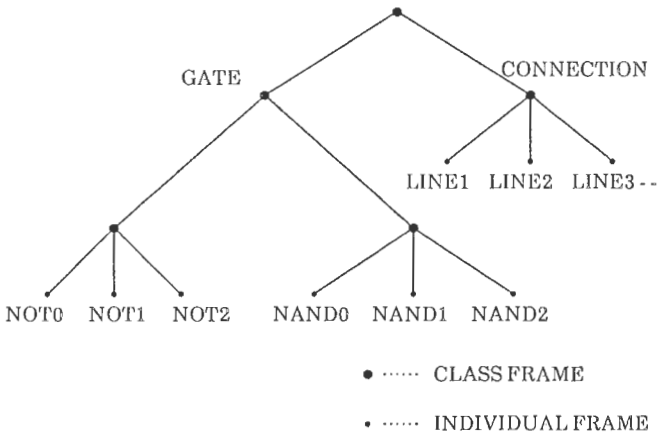



Fig.3-1. Hierarchical Structure Representation of the Logic Circuit

NOT0: Individual
Slots are:
INPUT: LINE6
OUTPUT: LINE7
. . . . .

LINE3: Individual
Slots are:
INPUT: NAND0
OUTPUT: NOT1
. . . . .

NAND0: Individual
Slots are:
INPUT1: NAND0
INPUT2: LINE2
OUTPUT: LINE3, LINE4
. . . . .

Fig. 3-2 Frame Representation Examples

indicated by the value of slot OUTPUT in frame $d$. In Fig. 2,

$$f_{rp}(S) = \{\{\text{INPUT}\}, \{\text{INPUT1}\}, \{\text{INPUT2}\}\}$$

and

$$D_1 = \{\text{LINE3}, \text{LINE6}, \text{LINE9}\},$$
$$D_2 = \{\text{LINE1}, \text{LINE4}, \text{LINE5}\},$$
$$D_3 = \{\text{LINE2}, \text{LINE7}, \text{LINE10}\},$$
$$K = \{D_1, D_2, D_3\}.$$

When all sets through $D_3$ are used, the value of $f_{rp}(a)$ can be estimated as {INPUT}, {INPUT1} or {INPUT2} for $a \in A$. In other words, the value can be estimated as "the reverse pointer of pointer OUTPUT for an individual frame that represents a connection is one of INPUT, INPUT1 and INPUT2."

### Domain Refinement

When the domains of observational functions $f$ and $g$ are both $A$, $A$ may be further refined by an observational function. Set

$$f : A \to V_f \qquad \text{and} \qquad g : A \to V_g$$

. Assume $D_1, \ldots, D_m$ are derived from $g$ by an estimation method similar to that used above. In this case, $A$ is almost equal to $D_1 \cup \ldots \cup D_m$. Derive $D_{i1}, \ldots, D_{ik}$ for

$$f|_i : D_i \to V_f$$

by limiting $f$ to each $D_i$. When $g^{-1}(v_i) = D_i$, the meaning of the estimation is "when $g(d) = v_i$, $f(d)$ is one of $f(D_{i1} \cup \ldots \cup D_{ik})$." Function $g$, used to refine the domain, is called a refinement function and the estimation derived in this way is called "estimating the value of $f$ derived by refining $g$."

### Example 3

Let's estimate the value of $f_{rp}$ by refining $f_p$. From example 1, $f_p$ may be refined as

$$D_1 = \{\text{LINE1}, \text{LINE2}, \text{LINE4}, \text{LINE5}, \text{LINE7}, \text{LINE10}\}$$
$$D_2 = \{\text{LINE3}, \text{LINE6}, \text{LINE9}\}.$$

Estimating the value of $f_{rp}$ for each $D_i$, we will have

$$D_{11} = \{\text{LINE1}, \text{LINE4}, \text{LINE6}\}$$
$$D_{12} = \{\text{LINE2}, \text{LINE7}, \text{LINE10}\}$$
$$D_{21} = \{\text{LINE3}, \text{LINE6}, \text{LINE9}\}.$$

This means "the reverse pointer of pointer OUTPUT for an individual frame that represents a connection is: (1) either INPUT1 or INPUT2 when the parent frame of the value of pointer OUTPUT is NAND, and (2) INPUT when the parent frame of the value of pointer OUTPUT is NOT," where the value of pointer OUTPUT means the value of slot OUTPUT.

Domains and ranges of observational functions and pairs of functions to refine domains are stored in the observational function enumerator section in advance, and by comparing the entered sample and domains, appropriate observational functions will be enumerated.

## §5. Extraction of Rules and Elimination of Exceptions

For any set $P$, $|P|$ denotes the cardinal number of $P$. Suppose $|S|$ is sufficiently large. If a rule can be found using the method described above in Section 3, what is observed first is the phenomenon that $|f(S)|$ is considerably smaller than $|S|$. This is because when $|S|$ is divided into segments using function $f$, many of the segments concentrate around the first several $D_i$, as shown in Fig. 4. Here, $D_1, \ldots, D_u$ is a sorted sequence described in Section 4. When this occurs, there is a gap in terms of size between the exceptional section and the rest. Furthermore, as the size of $|S|$ becomes large, the gap can be expected to widen. Therefore, it is possible to discard the minority section when a gap is identified by determining the size of the gap in advance. In KB-CAS, for the purposes of this study, the gap is determined to be 10% of $|S|$.

The second phenomenon noted is that when setting what is left after eliminating the exceptions from $S$ as $S'$, as shown above, $|f(S')|$ becomes considerably smaller than $|\text{Range}(f)|$, which denotes the cardinal number of the range of function $f$. This is a particularly important property when $|\text{Range}(f)|$ is small to start with. For example, when $|\text{Range}(f)| = 2$ and when the division of $S$ into segments by $f$ is obtained, unless $|D_1| >> |D_2|$, it cannot said there is a rule. In KB-CAS, $|f(S')|$ is required to be less than 2/3 of $|\text{Range}(f)|$.

When $|S|$ is small, the value of the gap described above is not correct. If assuming $|S| = 10$, the value of the gap is 1 and the gap has no meaning. Moreover, if $|S|$ is small, the probability that sample $S$ is a balanced set is low and there would be no value in carrying out a detailed analysis. For the sake of convenience, those $|D_i|$ whose values are larger than 20% of $|S|$ are retained and the rest are discarded as exceptions. Even if a correct rule is not extracted at this point, the extracted rule can be improved by the rule maintenance mechanism, to be discussed later.

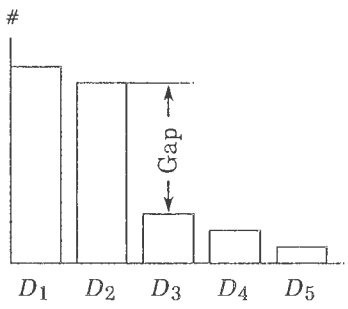



Fig.4. Graph of # of $D_i$

## §6. Rule Maintenance

### Problems

A rule extracted from a sample must be verified using data entered at a later stage. However, the result may not be what was originally desired. These errors are thought to occur because:

(1) The sample is biased toward elements with specific properties.
(2) The new data are confined to data with a specific property.

In order to avoid recurrences of errors, there needs to be a process whereby information is obtained from the creator* of the knowledge base, in the case of KB-CAS, indicating that the result of the application of the rule is not correct. The rule must then be either rewritten or abandoned. It is assumed that a set of data subject to rule extraction contains exceptions and there is a possibility that errors also. It is, therefore, necessary to determine to what extent errors should be tolerated and at what point the extracted rule should be rewritten.

The method proposed here assigns a confidence value, using statistical estimation, to rules extracted from a sample using statistical estimation. This method involves estimating the confidence by setting a confidence coefficient in advance and carrying out an interval estimation.

### Interval Estimation ([13])

In statistics, there is a procedure for extracting a sample from a set, carrying out observation on the sample, and estimating an unknown parameter $p$ $(0 \leq p \leq 1)$ from an observed value $x$. The confidence of the estimated value becomes important here. An effective way of determining the confidence is to find the estimated value in an interval with a sufficient range, such as

$$l(x) \leq p \leq u(x).$$

By setting the confidence coefficient as $0 \leq \alpha \leq 1$, and if

$$\Pr\{l(x) \leq p \leq u(x)\} \geq \alpha,$$

then the interval $[l(x), u(x)]$ is called the *confidence interval* of confidence coefficient a. Here, $\Pr\{A\}$ is the probability that event $A$ occurs. Thus, the confidence interval also represents an estimation of the range in which the true value of parameter $p$ falls from the sample. How the confidence interval is derived is explained in the appendix.

### Application to Rule Maintenance

The confidence interval is determined for the following two points to conduct an estimation.

(1) The probability $(P_R)$ that the extracted rule provides the correct answer,
(2) The proportion $(P_i)$ of $D_i$ to the total when the sample is classified by the observational function.

Every time new data are entered, a check is carried out to see whether the probability of the rule is still within the range of the confidence interval or not, and if the probability falls outside the range, the rule is re-extracted. In this case, (1) is a means to gauge the fairness of the rule, and the purpose of (2) is to check to see if $D_i$ is an exception or not. In fact, (2) converges to the correct rule as new data are entered when the sample contains biased data and when the rule for such data is in the majority in terms of that particular sample but is a minority in terms of all samples. In KB-CAS, the confidence coefficient is set at 80%.

### Example 4

Assume that in a knowledge base for animals, the frames regarding birds only contain $S$ = {swan, sparrow, penguin, canary, ostrich}. Let $A$ be a set of frames regarding birds that are entered into the entire knowledge base, and $f : S \rightarrow \{\text{fly}, \text{not-fly}\}$ be the observational function whose value $f(s)$ is the value of slot FLY of frame $s$ for $s \in S$, so $f$ observes whether a bird flies or not. In this case,

$$D_1 = f^{-1}(\text{fly}) = \{\text{swan}, \text{sparrow}, \text{canary}\}$$
$$D_2 = f^{-1}(\text{not-fly}) = \{\text{penguin}, \text{ostrich}\}$$

This means that "birds fly or do not fly" and thus is a tautology. In this case, $|\text{Range}(f)| = |f(D_1 \cup D_2)| = 2$ and no rule will be extracted. The confidence interval for each can be calculated as (cf. appendix)

$$1 \leq P_R \leq 1$$
$$0.41 \leq P_1 \leq 0.78$$
$$0.21 \leq P_2 \leq 0.59$$

Furthermore, when new data "duck" is entered ($|S| = 6$), duck will belong to $D_1$ but

$$|D_1|/|S| = 4/6 \leq 0.78$$

and there is no change to the rule. When four more data elements on birds that fly are entered, that value of $|D_1|/|S|$ exceeds the range of confidence interval. This time the rule is re-extracted. $S$ = {swan, sparrow, canary, penguin, ostrich, duck, swallow, pigeon, owl, hawk}, and $D_1$ and $D_2$ can be expressed as follows:

$$D_1 = f^{-1}(\text{fly}) = \{\text{swan}, \text{sparrow}, \text{canary}, \text{duck}, \text{swallow}, \text{pigeon}, \text{owl}, \text{hawk}\}$$
$$D_2 = f^{-1}(\text{not-fly}) = \{\text{penguin}, \text{ostrich}\}.$$

In this case, $D_2$ is treated as an exception and the rule "birds typically fly" is extracted.

* In this case, this creator is an oracle of the observational functions

## §7. Application to a System for Aiding the Construction of Knowledge Bases

The practical applications for expert systems are rapidly expanding. To become truly practical, however, expert systems require many frames, resulting in their knowledge bases becoming extremely complex and large. It is expected that the cost of constructing knowledge bases and the number of errors during data entry will continue to increase.

When the structure of frame-based knowledge bases is examined, it is found that there is a series of structures that are similar among the structures for inter-frame relationships and the values of slots. It is, thus possible to assist in the construction of knowledge bases by using an initial knowledge base as a sample of the large knowledge base. Rules can then be extracted, based on the sample knowledge base, which can be used to verify new knowledge data. This should improve the efficiency of knowledge base construction and cut down on the number of entry errors. This system is called KB-CAS.

Currently, the rules observed in KB-CAS are as follows.

1) Rules regarding slot values
   * Is the slot value constant?
   * If the slot represents a pointer for another frame, what sort of frame is this pointer for?

2) Rule regarding relationships
   * When a slot represents a pointer, what is its reverse pointer?

3) Rule regarding multiple frames
   * Detect slots with the same value among frames that are linked with a certain relationship.

After a sample knowledge base is given, construction of frames and storing of slot values are carried out semi-automatically based on the extraction rules, which will improve the construction of the knowledge base thereafter. For example, to connect NAND3 to LINE8, in Fig. 2, NAND3 must first be set into the slot OUTPUT in frame LINE8. From the rule extracted in example 1, we know that NAND3 is an individual frame that represents NOT or NAND. Therefore, KB-CAS creates frame NAND3 by referring this question to the knowledge base creator (in actual situations it is clear without inquiry that NAND3 is an individual frame of NAND, from a pattern generalization program [1] and a simple string pattern match program). Furthermore, based on the rule in example 3, LINE8 should be stored in slot INPUT1 or INPUT2 of frame NAND3. After an inquiry by the creator, LINE8 is stored in either INPUT1 or INPUT2 as a slot value.

## §8. Summary

This paper has described a method for extracting a property or a rule from a set of data (sample) that contains exceptions, and for rewriting or deleting a rule after verifying the validity of the rule using newly entered data and employing statistical techniques. In short this is a method of extracting a default rule from a set of facts.

It has been confirmed that in the case of small-scale experimental knowledge bases, KB-CAS is an effective tool. In the next stage of research, KB-CAS will be applied to a large-scale experiment to verify the validity of the rules extracted and the rule maintenance method. Moreover, the statistical value of the threshold gap and the confidence coefficient discussed in sections 3 and 4 will be further refined in a future study.


## Acknowledgements

I would like to thank Fumio Hattori, Toru Ishida and Jun-ichi Akahani for their constructive comments on this paper.


## REFERENCES


[1] Angluin, D. (1980). "*Finding Pattern Common to a Set of Strings*," J. Comput. Syst. 21, pp.46-62.

[2] Dietterich, T.G. and Michalski,R.S. (1983). "*A comparative review of selected methods for learning from examples*," in Machine Learning: An artificial intelligence approach, R.S.Michalski, J.G.Carbonell, and T.M.Mitchel(Eds.), Tioga, Palo Alto, Calif., pp.41-81.

[3] Fisher, D.H. (1987). "Knowledge Acquisition via Incremental Clustering," Machine Learning 2, pp.139-172.

[4] Gold, M. (1967). "*Language identification in the limit*," Inf. and Control 10, pp.447-474.

[5] McCarthy, J. (1986). "*Application of Circumscription to Formalizing Common-Sense Knowlege*," Artificial Intelligence 28, pp.89-116.

[6] McDermott, D.V. and Doyle, J. (1980). "*Non-monotonic Logic i*," Artificial Intelligence 13, pp.41-72.

[7] Michalski, R.S. and Stepp, R.R. (1983). "*Learning from Observation: Conceptual Clustering*," in Machine Learning: An artificial intelligence approach, R.S.Michalski, J.G.Carbonell, and T.M.Mitchel(Eds.), Tioga, Palo Alto, Calif., pp.41-81.

[8] Michalski, R.S. and Winston, P.H. (1986). "*Variable Precision Logic*," Artif. Intell. 29, pp.121-146.

[9] Mitchel, T.M. (1978). "*Version Spaces: An approach to concept learning*," Stanford CS Rep. STAN-CS-78-711.

[10] Nix, R. (1984). "*Editing by example*," Proc. of the 11th ACM Symposium on Principles of Programming Languages, ACM, New York, pp.186-195.

[11] Reiter, R. (1980), "*A Logic for Default Reasoning*," Artificial Intelligence 13, pp.81-132.

[12] Shapiro, E.Y. (1981). "*Inductive inference of theories from facts*," Tech. Rep. 192, Dept. of Comp. Sci., Yale Univ., New Haven.

[13] Hoel, P.G., Port, S.C. and Stone, C.J. (1971). "*Introduction to Statistical Theory*," Houghton Mifflin.

[14] Winston, P.H. (1982). "*Learning by augmenting rules and accumulating censors*," MIT AI Lab., AIM-678, Cambridge, MA.

## APPENDIX

If we call the probability of the extracted rule deriving a correct result $p$, the probability of it deriving an incorrect result is $1-p$ and the probability of the rule deriving the correct result will be subject to a binomial distribution. Since binomial distributions are discrete, a randomized confidence interval is employed instead of the normal confidence interval. This is derived using the following method. Among ($|S|$=)n number of samples, when there are $x$ number of cases where the extracted rule derives the correct result, the largest integer $l$ will be found which satisfies the condition

$$\sum_{x-l\le k\le x+l} \binom{n}{k} x^k (n-k)^{n-k}/n^n \le \alpha$$

Here, $\binom{n}{k} = 0$ ($k<0$ or $k>n$) is set. Next, $p$ is given by the following linear equation

$$\left\{ \binom{n}{x-l-1} x^{x-l-1}(n-x)^{n-(x-l-1)}/n^n + \binom{n}{x+l+1} x^{x+l+1}(n-x)^{n-(x+l+1)} \right\} \rho + \sum_{x-l\le k\le x+l} \binom{n}{k} x^k (n-k)^{n-k}/n^n = \alpha$$

($0 \le p \le 1$). In this case, the randomized confidence interval can be expressed as

$$\max\left(0, \frac{x-l-\rho}{n}\right) \le p \le \min\left(1, \frac{x+l+\rho}{n}\right) \quad (1)$$

When a computer is used to calculate the randomized confidence interval, there is the disadvantage that the computation time becomes large when the number of samples, $n$, becomes large. This is due to the large number of factorial calculations and calculations of large numbers. To offset this disadvantage, the confidence interval will be made it closer to the normal distribution. In other words, according to the Central Limit Theorem, the larger the observed value of $n$, the closer the distribution of $x/n$ is to a normal distribution. Thus, the computation of the confidence interval will be carried out by making the distribution closer to a normal distribution. The method of computation is complex and will not be described here. However, when the confidence coefficient is set to $\alpha$, the following confidence interval will be obtained.

$$\frac{x+\kappa_\alpha/2 - \sqrt{\kappa_\alpha^2 x - \kappa_\alpha^2 x^2/n + \kappa_\alpha^4/4}}{n+\kappa_\alpha^2} \le p \le \frac{x+\kappa_\alpha/2 + \sqrt{\kappa_\alpha^2 x - \kappa_\alpha^2 x^2/n + \kappa_\alpha^4/4}}{n+\kappa_\alpha^2} \quad (2)$$

where constant $\kappa_\alpha$ is a value that can be found from the table of normal distribution, such as $\kappa_{0.95} = 1.96$. The size of the calculation steps for (2) does not depend on $n$. When $n$ is small, the confidence interval derived from the above formula tends to become wider compared to the randomized confidence interval. Therefore, the selection of either (1) or (2) involves a trade-off between accuracy and computation time. In KB-CAS, formula (2) is used when $n$ exceeds 50.

# Search Strategies for Finding Partial Answers in Large Knowledge-Bases

Jiawei Han
*School of Computing Science*
*Simon Fraser University*
*Burnaby, B.C. V5A 1S6*
*CANADA*

Lawrence J. Henschen
*Department of EE/CS*
*Northwestern University*
*Evanston, Illinois 60208*
*U.S.A.*

Wenyu Lu
*Computer Science Department*
*St. Cloud State University*
*St. Cloud, Minnesota 56301*
*U.S.A.*

## ABSTRACT

*Database search techniques are designed for retrieving all the answers to queries in large databases; while most AI search techniques are designed for finding partial answers in relatively small databases. However, many knowledge-based applications require to find partial answers in large knowledge-bases. We examine important search heuristics for finding partial answers to queries in knowledge-based systems and propose integrated search methods which integrate the set-oriented database search techniques with AI best-first or depth-first search techniques. We demonstrate that by using the proposed heuristics and the integrated search methods, the query processing cost can be considerably reduced while the quality of answers can be improved.*

## 1. Introduction

An important distinction between relational database systems and PROLOG programming systems concerns search requirements. Database users usually require to find *all* the answers to a query, such as retrieving all the students registered in a course. PROLOG users often require to find only *one or a few* answers to a query, such as finding the next move in a game. Different search requirements demand different search strategies. Database systems apply set-oriented search to finding all the tuples in a data relation satisfying search arguments expressed in relational algebra or relational calculus; while PROLOG systems apply depth-first search and backtracking to finding one answer, and the answer to be found is often dependent on the order of facts and rules stored in PROLOG databases and the order of predicates in the query.

Such diverse search requirements are natural in many applications. However, queries in many knowledge-based applications are unlike these two extremes. For example, to book a flight from Chicago to Vancouver, a customer seldom requires to find all the possible combinations but only a few good ones. To distinguish such different search requirements, we call conventional database queries **find-all queries** but queries for finding partial answers in a knowledge-base **find-some queries**.

The existing search techniques in relational or logic programming systems may be able to process *find-some* queries in large knowledge-bases. However, it is not easy for them to do it elegantly or efficiently. For example, we may use database techniques to exhaustively search for all the possible answers and then toss away most of them, which could be very costly. Moreover, searching for all the answers is difficult in complex recursive databases, especially those containing cyclic data because the testing of termination conditions could be very complex [7]. On the other hand, PROLOG's tuple-oriented depth-first search and backtracking method is inappropriate for searching large knowledge-bases [Zani84]. Its order-dependent characteristic also places burdens on PROLOG programmers in arranging appropriate orders or using extra-logic features, such as "cut", to control processing flow, which weakens the declarative philosophy of logic programming.

Such observation motivates us to study the integration of two search methods: database set-oriented search and AI depth-first or heuristic search. The *integrated search methods* should have the features of both set-oriented and depth-first or best-first search. Such search methods can be outlined as follows: At each choice point, instead of selecting one path as in PROLOG, a set of paths are to be selected and explored in parallel. However, unlike searches in database systems which explore all the search space, some of the space is purposely left unexplored, which could be portions of databases or disjunctions of rules, as long as enough answers can be found. The set-oriented backtracking can be initiated for further exploration on the unexplored portions of search space if more answers are needed until the entire search space has been explored.

In principle, the integrated search methods are appropriate for many knowledge-based applications. We confine our discussion to query processing in deductive database systems [4]. The results can be easily extended to logic programs containing substantial amount of facts [10, 15] or some other domains.

Like most researchers in deductive databases, we assume that a deductive database contains an extensional database (*EDB*) consisting of a set of base relations and an intentional database (*IDB*) consisting of a set of deduction rules in *safe* (*range restricted*) Horn clause [4]. Moreover, we assume that the size of IDB is much smaller comparing with EDB relations (*data intensive assumption*). Our notational convention is almost the same as in PROLOG [3], except that the upper/lower cases for variables and constants are reversed: an identifier starting with a lower case letter represents a variable, and starting with an upper case letter a constant or a function name.

Similar to the methods studied in the processing of *find-all* queries in deductive databases, we propose to separate the

processing into two phases: the compilation of IDB and the query processing using the compiled IDB and EDB [8, 9]. Since such method separates deductive search from expensive database search, set-oriented processing and global optimization can be explored on the compiled results to reduce total processing costs.

Since the compilation process of find-some queries is similar to that of find-all queries [1, 2, 7, 8, 14], our discussion emphasizes the dynamic aspects of compilation and query processing. Section 2 studies the general heuristics on compilation and query processing. Section 3 discusses the integrated search methods. Section 4 is on the search in multiple directions. We conclude our discussion in Section 5.

## 2. General Heuristics on Compilation and Processing

Like most studies where general principles are extracted from the analysis of specific examples, we examine two application-oriented examples.

**Example 1.** The *same generation* problem [2]. The predicate *same-generation cousin (SG)* is defined by the following rules.

$$SG(x, y) :- SIB(x, y). \quad (1)$$

$$SG(x, y) :- PA(x, u), PA(y, v), SG(u, v). \quad (2)$$

where the predicate $SG(x, y)$ indicates that $x$ and $y$ are the same-generation cousins, $PA(x, y)$ is an *EDB* predicate indicating that $y$ is $x$'s parent, and $SIB(x, y)$ is a non-recursive *IDB* predicate defined by $PA$. The compiled formula is

$$\bigcup_{k=0}^{n} PA^k \; SIB \; CH^k. \quad (3)$$

where $PA^k$ indicates that there are $k$ $PA$'s forming a well formed join expression, and $CH$ is defined by

$$CH(x, y) :- PA(y, x).$$

The query, *find some persons who are John's same-generation cousins and who were born in 1960's*, is in the form

$$?- SG(John, y), BirthYear(y, birth\_year),$$

$$birth\_year \geq 1960, birth\_year < 1970. \quad (4)$$

**Example 2.** The *air-flight* problem [6]. The *connected-flight*, *Flights*, is defined as

*Flights* ([*flight_no_list*], *Departure* (*dep_city*, *dep_time*),

*Arrival* (*arr_city*, *arr_time*)).

which consists of the following two rules.

*Flights* ([*f* | *fL*], *Departure* (*d_c*, *d_t*), *Arrival* (*a_c*, *a_t*)):–

*Single_flight* (*f*, *Departure* (*d_c*, *d_t*), *Arrival* (*i_c*, *i_at*)),

*Flights* ([*fL*], *Departure* (*i_c*, *i_dt*), *Arrival* (*a_c*, *a_t*)).

*Flights* ([*f*], *Departure* (*d_c*, *d_t*), *Arrival* (*a_c*, *a_t*)) :–

*Single_flight* (*f*, *Departure* (*d_c*, *d_t*), *Arrival* (*a_c*, *a_t*)).

Notice that the definition contains simple function symbols, e.g., Departure (dep_city, dep_time), and simple list manipulation functions, e.g., [flight_no_list], which can be implemented by extending the techniques developed in deductive database research [12]. Obviously, the compilation of this set of rules results in a transitive closure of the base relation *Single_flight*. The query, *find some (connected) flights from Chicago to Vancouver*, can be written as

$$?- Flights(flight\_list, Departure(Chicago, d\_time),$$

$$Arrival(Vancouver, a\_time)). \quad (5)$$

Most inquiries related to these IDB predicates are *partial* queries because people are seldom interested in finding *all* the *remote* cousins of John in Ex. 1 or *all* the flights from Chicago to Vancouver via *tiny* airports in Ex. 2. The problem becomes how to efficiently find a *small* set of *good* answers.

We propose the following heuristics.

**Heuristic 1. (Cluster-Oriented Compilation).** *The rule cluster-oriented compilation of an IDB may reduce unnecessary search of large rule bases.*

A query is usually relevant to only one or a few IDB predicates, and an IDB predicate is defined (directly or transitively) by one or a few rules. We define a **cluster** of an IDB predicate P to be a set of rules which directly or indirectly define the predicate P. Grouping rules into clusters avoids unnecessary search on a large body of irrelevant rules and clarify the relationships among rules [7]. A rule cluster is usually much smaller than an EDB relation. Therefore, the cost of storage and access of compiled rule clusters is much smaller than that of EDB relations.

Since a rule directly defining a predicate $P$ may indirectly define another predicate $R$, it may belong to both the P- and R-clusters. For example, in Ex. 1, the rule defining *SIB* belongs to both *SIB* and *SG* clusters. We treat a rule belonging to several clusters as multiple copies of the rule, each copy belonging to one cluster, thus the optimization on one cluster, such as the elimination of an intermediate predicate, will not have "side-effects" on other clusters.

**Heuristic 2. (Application of User or Expert- Specified Constraints).** *Expert knowledge or user search requirements are important constraints to reduce search space and generate knowledgeable answers.*

For example, in Ex. 2, a ticket agent (expert) may tell us (i) most customers like to choose direct flights or flights with fewer stops, (ii) it is preferable to have each single flight flying in the direction closer to the global *Flights* direction, and (iii) it is preferable to fly via big airports rather than tiny airports. Such heuristics usually play two roles, *reducing search space* and *improving the quality of answers*. These *search constraints* should be specified by experts as knowledge rules or by user in their queries.

**Heuristic 3. (Starting at the Most Selective Points).** *Performing selection first and starting at the most selective points makes the search focus on the set of relevant facts and reduces the size of intermediate relations.*

This heuristic has been popularly used in both conventional database and deductive database query processing [2, 8, 13]. The selective points are often provided by query constants. When several constants are available, the most selective point should be distinguished by comparing the selectivities of the constants, e.g., *John* is usually more selective than 1960s in Ex. 1.

**Heuristic 4. (Controlling the Depth of Recursion).** *Search confined to shallow levels of recursion often produces more efficient and knowledgeable answers.*

Recursion in a large knowledge-base often produces huge sets of answers with the problems of inefficiency, meaningless answers and non-termination. An intersting observation is that search confined to shallow levels of recursion often produces efficient and knowledgeable answers. For example, in the connected_flight problem, flights with fewer stops often generate preferred answers. Also, in the same-generation problem, people would seldom like to trace John's "remote" cousins to centuries ago.

We should recognize that deep recursion is necessary in solving some intersting problems. Nevertheless, shallow recursion has its obvious advantages: (i) the expensive search on deep recursion can often be saved, (ii) the complicated termination testing can often be omitted, and (iii) the answers thus derived are often expected to represent knowledgeable answers. In many cases, this heuristic can be taken as default unless explicitly deactivated when a user likes to examine deeper levels.

**Heuristic 5. (Selective Search of Disjunctions in a Compiled Formula).** *Search can sometimes be confined to one or a few disjunctions of compiled formulas and leave other disjunctions untouched.*

For example, in the *same-generation* example, suppose the parent rule *PA* is defined by two relations *Father* and *Mother* disjunctively. Search may follow one relation or even interleave between the two as long as enough answers can be found. If some set of disjunctions is known more important than others, the more important disjunction(s) should be explored first. The confinement of recursion to shallow levels can also be viewed as a preference at the selective search of the disjunctions of compiled formulas because different depths of recursion form a union in the compiled formula.

**Heuristic 6. (Selective Search of Portions of EDB Relations).** *Selective search can also be performed on portions of EDB relations based on different criteria, such as the importance of the data, user preference, and the closeness and easiness in accessing the data, etc.*

The importance of the data can be determined based on some data priority rules defined by experts or the database administrator or based on the statistics of queries on databases, such as the frequency of reference. Priority rules are usually defined by users or experts based on their knowledge, e.g., the size and classes of the airports in the air-flight problem. If such knowledge are frequently used in search, indexing and clustering should be performed on the database to cluster data with similar priorities together to reduce I/O accessing cost. Such ordering often saves processing cost and helps producing knowledgeable answers. Selective search can also be based on the closeness or easiness in accessing databases, e.g., data in the same index or data pages.

**Heuristic 7. (Tuning the Search Preference Based on the Size of Expected Intermediate Relations).** *Search preference should also be tuned based on the size of the expected intermediate relations which is usually determined by the size of the joining relations and join selectivity.*

For example, if it produces only a tiny intermediate relation by joining with a portion of relation $B$, it is suggested to search more portions of $B$ to obtain more intermediate results. On the contrary, if a huge intermediate relation is to be generated, more constraints should be incorporated to reduce the size of expected intermediate results. The goal of such a dynamic adjustment is to obtain sufficient answers while reducing I/O and other processing costs.

**Heuristic 8. (Multiple-Directioned Parallel Search).** *Search starting at different starting points and proceeding in multiple directions parallelly may save processing cost and speed up the processing.*

More details on this heuristic will be analyzed in Section 4.

The heuristics discussed above are often interrelated. For example, the cluster-oriented compilation should cluster deduction rules with the associated search constraints in compilation. However, the use of these search constraints should be dynamically adjusted based on the size of the intermediate results. Since some heuristics, such as *cluster-oriented compilation* and *starting at the most selective point*, are shared in both find-all queries and find-some queries, our discussion focuses on those special in find-some queries and studies their roles in developing *integrated search methods* and *multi-directioned search.*

## 3. The Integrated Search Methods

The major difference in search strategies between find-some and find-all queries is at the dynamic and heuristic aspects of search. Since set-oriented database search techniques have been proven superior to PROLOG's tuple-oriented search in handling large amount of data [16], our study follows the *set-oriented* philosophy. Nevertheless, the search for find-some queries should be different from those in conventional databases where the "breadth-first" search philosophy is adopted. We propose an integration of database set-oriented search with AI depth-first and best-first search.

To simplify our discussion, we first assume that the query to be processed consists of a conjunction of extensional predicates (data relations), and each data relation is clustered and partitioned into several segments (nodes, in our discussion) according to some preference rules and/or their physical adjacency. Among many possible combinations of database and AI search methods, we analyze three typical ones: *set-oriented depth-first search, set-oriented best-first search*, and *set-oriented mixed search.*

In these approaches, "set-oriented" indicates that search proceeds within each node (segment) in set-oriented manner, much like conventional database processing. However, they adopt different philosophy at searching among different nodes. In the set-oriented depth-first search, the selection of nodes is performed in depth-first manner. In the set-oriented best-first

search, the selection is done in best-first manner. While in the set-oriented mixed search, the selection is a mixture of depth-first and best-first ones: best-first among nodes within each data relation but depth-first among different data relations. We study and compare these approaches.

### 3.1. The Set-Oriented Depth-First Search

**Example 3.** We study the processing of find-some query on the compiled formula $\sigma_{x=a}\, A(x, y), B(y, w), C(w, z)$, which can be described as: performing selection on relation $A$ using a set of values denoted by $a$, then performing join on $B$ and then on $C$ to derive the set of $z$ values. Suppose each relation is partitioned into several nodes. For example, $A_1, A_2$ for $A$, $B_1, B_2, B_3$ for $B$, and $C_1, C_2, C_3$ for $C$.

Suppose that there is no specification on the preference of nodes, and the search simply follows the partitioned order, i.e., $B_1 < B_2 < B_3$, where $B_1 < B_2$ indicates that $B_1$ is selected before $B_2$.

The set-oriented depth-first search proceeds as follows. We set up $n$ node pointer queues for the $n$ relations to be processed, each has three pointers: *current* (pointing to the current node of the queue), *front* (pointing to the front of the queue) and *rear* (pointing to the rear of the queue). At initialization, each *current* points to the *front* of its queue. The process proceeds until enough answers have been found or all the candidates are processed. The database searcher processes the nodes pointed to by the *current* pointers, and these pointers are adjusted in depth-first manner: if the *current* in the deepest queue does not reach its *rear*, move it one step forward; otherwise, reset it to the *front*, and set one step forward the *current* of the queue next to the deepest queue, and so on.

In our example, the complete search sequence should be

| | | | |
|---|---|---|---|
| (1) $A_1B_1C_1$ | (2) $A_1B_1C_2$ | (3) $A_1B_1C_3$ | (4) $A_1B_2C_1$ |
| (5) $A_1B_2C_2$ | (6) $A_1B_2C_3$ | (7) $A_1B_3C_1$ | (8) $A_1B_3C_2$ |
| (9) $A_1B_3C_3$ | (10) $A_2B_1C_1$ | (11) $A_2B_1C_2$ | (12) $A_2B_1C_3$ |
| (13) $A_2B_2C_1$ | (14) $A_2B_2C_2$ | (15) $A_2B_2C_3$ | (16) $A_2B_3C_1$ |
| (17) $A_2B_3C_2$ | (18) $A_2B_3C_3$ | | |

**Figure 1.** The *Complete* Search Sequence of Ex. 3 by Set-Oriented Depth-First Search

The algorithm is summarized as follows.

**Algorithm 1. The Set-Oriented Depth-First Search.**

**Input** : A knowledge-base consisting of (i) a set of *EDB* relations, $P_1, P_2, ..., P_n$, with each relation $P_i$ partitioned into a set of nodes (segments); and (ii) a compiled $R$-cluster:

$R :- P_1, P_2, \cdots, Pn.$

**Output** : A set-oriented depth-first search plan for the find-some query $?-R(a, z)$.

**Data Structure**: We set up $n$ queues holding the node pointers of $n$ data relations (Q[i] for relation i, and Current[i], Front[i] and Rear[i] for the current, front and rear pointers of Q[i]).

```
BEGIN
    FOR i := 1 TO n DO Current[i] := Front[i];
    WHILE NOT ( enough_answer OR exhausted ) DO
        BEGIN
            Process_Currents;
            IF enough_answer
            THEN return(True)
            ELSE Reset_Currents(n);
        END; { While }
    IF NOT Enough_Answers
    THEN The problem is unsolvable.
END.

PROCEDURE Reset_Currents(i : integer);
BEGIN
    IF Current[i] ≠ Rear[i]
    THEN Current[i] := Current[i] + 1;
    ELSE IF i > 1 THEN
     BEGIN
          Current[i] := Front[i];
          Reset_Currents(i – 1)
     END
    ELSE exhausted := True
END. { Reset_Currents } □
```

### 3.2. The Set-Oriented Best-First Search

Different from the first method, the set-oriented best-first search assumes that there is a total ordering of preference among the segments (nodes) of the data relations to be searched. At any selection point, we always select the most preferred ones from among all the candidates.

**Example 4.** We solve the same problem presented in Ex. 3 using the set-oriented best-first search method. We assume that there exists a total ordering of preference for all the nodes to be searched:

$$A_1\#1 < B_1\#2 < C_1\#3 < C_2\#4 <$$
$$B_2\#5 < A_2\#6 < C_3\#7 < B_3\#8.$$

The best-first rule indicates that at each selection point, the most preferred candidates should be selected. Therefore, the complete search sequence should be:

| | | | |
|---|---|---|---|
| (1) $A_1B_1C_1$ | (2) $A_1B_1C_2$ | (3) $A_1B_2C_1$ | (4) $A_1B_2C_2$ |
| (5) $A_2B_1C_1$ | (6) $A_2B_1C_2$ | (7) $A_2B_2C_1$ | (8) $A_2B_2C_2$ |
| (9) $A_1B_1C_3$ | (10) $A_1B_2C_3$ | (11) $A_2B_1C_3$ | (12) $A_2B_2C_3$ |
| (13) $A_1B_3C_1$ | (14) $A_1B_3C_2$ | (15) $A_2B_3C_1$ | (16) $A_2B_3C_2$ |
| (17) $A_1B_3C_3$ | (18) $A_2B_3C_3$ | | |

**Figure 2.** The *Complete* Search Sequence of Ex. 4 by Set-Oriented Best-First Search

The algorithm is summarized as follows.

**Algorithm 2. The Set-Oriented Best-First Search.**

**Input** : the same as Algorithm 1 except there is a total ordering of preference for the nodes to be searched;

**Output** : A set-oriented best-first search plan for the find-some query $?-R(a, z)$.

**Data Structure** : A processing stack *STK* (initialized to empty), with each stack element containing two fields: a node *Node* and its associated lists *Ls*. *EDB* relations are represented by a collection of lists EDB_Ls with the i-th list *Li* containing all the nodes (partitions) of relation $P_i$.

**BEGIN**
PROCESS(The most preferred unprocessed node in EDB_Ls)
**IF NOT** *Enough_Answers*
**THEN** *The problem is unsolvable.*
**END.**

**PROCEDURE** PROCESS(Candidate_Node);
**BEGIN**
Push_Stk(Candidate_Node, Candidate_Ls);
{ Candidate_Ls is calculated before pushing, and is the collection of the candidate lists of Ls of STK_TOP or EDB_Ls (when STK is empty) except the list containing the Candidate_Node. Note: The candidate lists contain only the nodes with preference number not greater than the Candidate_Node, and all the nodes pushed onto the STK are marked *unprocessed.* }
**IF** the associated list of the STK_TOP = ϕ **THEN**
**BEGIN**
**IF** Length of STK = n { Length of the formula to be processed }
**THEN** Generate and process the subformula consisting of the set of STK nodes;
Pop the STK and mark the Candidate_Node *processed* in the STK_TOP or EDB_Ls if Empty_STK;
**END;**
**WHILE NOT** Empty_STK **AND** All the nodes of Ls of the current STK_TOP are *processed* **DO**
Pop the STK and mark the Candidate_Node *processed* in the STK_TOP or EDB_Ls if Empty_STK;
**IF NOT** (*Enough_Answers* **OR** *Exhausted_Search*) **THEN**
{ *Exhausted_Search* is TRUE if there is no unprocessed node in EDB_Ls }
PROCESS (The most preferred unprocessed node in the Ls of the current STK_TOP or in EDB_Ls if Empty_STK);
**END**; { PROCESS } □

The algorithm can be illustrated by watching a fragment of the execution of Ex. 4. At the moment when the last unprocessed node $B_3$ is at the bottom of the stack and $A_2$ is the next to the bottom, the stack becomes: ('*' is the *processed* mark.)

| Stk_Node | Candidate_Lists |
|---|---|
| $C_1$#3 | ϕ |
| $A_2$#6 | { $C_1$#3 , $C_2$#4 } |
| $B_3$#8 | { *$A_1$#1 , $A_2$#6 } { *$C_1$#3 , *$C_2$#4 , $C_3$#7} |

The subformula generated is $A_2B_3C_1$. The next step (with $C_1$ marked *processed*) is

| Stk_Node | Candidate_Lists |
|---|---|
| $C_2$#4 | ϕ |
| $A_2$#6 | { *$C_1$#3 , $C_2$#4 } |
| $B_3$#8 | { *$A_1$#1 , $A_2$#6 } { *$C_1$#3 , *$C_2$#4 , $C_3$#7} |

The subformula generated is $A_2B_3C_2$. The next two steps generate $A_1B_3C_3$ and $A_2B_3C_3$, and the process terminates.

### 3.3. The Set-Oriented Mixed Search

The set-oriented mixed search is a balance of the above two strategies. Since the search of the first method does not apply preference rules, more data segments may have to be explored with less knowledgeable answers expected to be found. The second method assumes that there is a total ordering for all the nodes in the relations to be searched. However, such a total ordering may not be available in many applications. It is more practical to assume an ordering of preference for the nodes in each data relation but not a total ordering for all the nodes in the database. Based on such an assumption, we present another integrated search method, *the set-oriented mixed search.*

In the set-oriented mixed search, the selection of a node (segment) is based on a best-first rule within each relation but a depth-first rule among different data relations, that is, select a more preferred node within each relation being searched. If there have not been enough answers, the (continued) search backtracks at the last (deepest) selection point, and so on until enough answers are found or all the search space has been explored.

**Example 5.** We solve the same problem of Ex. 3 based on the assumption that the nodes are partitioned and ordered as in Ex. 3 based on the preference rule(s) rather than being done randomly as in Ex. 3 or based on physical clustering. Moreover, we assume that the reorganization of data relations, such as sorting or clustering, based on such preference rules is also performed. Under such assumptions, the generated search sequence and the algorithm are the same as those presented in Ex. 3. However, the implications are different because the former implies a simple set-oriented depth-first search while the latter implies a mixture of depth-first and best-first search.

### 3.4. A Comparison and Discussion of the Three Methods

The three integrated methods share some common properties.

(1) They all adopt set-oriented search philosophy in searching each data segment, which is appropriate for large knowledge-bases. The set-oriented search reduces the expensive I/O processing cost and the chances of backtracking.

(2) Since they are designed for finding partial answers, the methods examine only portions of data relations at a time, which indicates that the search may occasionally have to go back to the previous choice point to continue searching the unexplored portions of the search space. Therefore, backtracking, though could be considerably reduced by set-oriented search, is not eliminated.

(3) Considering all the combinations of different segments, the search in all three methods are complete and non-redundant, i.e., all the portions of the knowledge-base to be searched are reachable but no portion will be searched more than once.

(4) The principles developed here, though only on simple conjunctive formulas, are applicable to complicated rule clusters, such as formulas containing disjunctions and recursion.

However, the three methods have their different characteristics.

(1) The first method may involve more backtracking and/or produce less knowledgeable answers because the search is not guided by preference rules. The second method may not be applicable since the total ordering of all the nodes to be searched is not always available. However, when applicable, it is expected to produce more knowledgeable answers with less backtracking. The third method is a compromise of the previous two. It only requires the ordering within each relation, which is easy to be satisfied in many applications.

(2) Although there are no redundant subformulas generated, the second method may still suffer from redundant processing of portions of subformulas. For example, in Ex. 4 the subformulas (1) $A_1B_1C_1$, (2) $A_1B_1C_2$ and (9) $A_1B_1C_3$ share a subformula $A_1B_1$. If the intermediate results $A_1B_1$ is not saved in the processing of (1) and (2), it has to be reprocessed for (9). It is costly to save all the intermediate relations. Such a problem will not occur in Methods 1 and 3. Therefore, the second method may cost more than the other two when backtracking occurs more frequently and/or the partitions are too fragmented. In these cases, we should modify the best-first search strategy for different knowledge-bases and queries, e.g., giving higher preference to later processed relations, selectively saving intermediate results, or even switching to the mixed search method.

Our study of the three typical integrated search methods does not exclude other integrated search methods. Variations of the three methods are possible. For example, if preference information is available only in some of the searched relations, we may order the applicable ones using preference rules but select others randomly or according to the physical clustering. The resulting algorithm will be a mixture of the first and the third methods.

Another interesting issue is the judgement of "enough answers". The enoughness can be predefined, such as predefining the required number of answers, however, for most interactive systems it is better to be judged by users interactively. A user may first request a set of answers, and repeatedly request more if not satisfied with the existing set of answers. The algorithms presented above are also applicable to such queries, and in such cases, the flag of "enough_answers" should be interactively set by users.

## 4. The Control of Search Directions

*Starting from the most selective point* is an important heuristic in the determination of search directions. Since a query often contains several query constants, search may start at multiple points and proceed in multiple directions.

Different from the discussion in [11] which concerns the selection of application directions of rules between forward chaining and backward chaining, our study of search directions is based on the data intensive assumption (large amount of data and small amount of relevant rules), hence the backward chaining of the application of rules is always suggested, i.e., starting at query constants (goals) rather than at large amount of facts.

We first study multiple directioned search for non-recursive queries. The study in query optimization in conventional database systems [13] indicates that *selections should be performed before joins to reduce the size of relations to be joined.* For the same reason, proceeding from multiple selective points also benefits non-recursive deductive query processing for both find-all and find-some queries. For find-some queries, the search may be confined to portions of data relations or disjunctions of compiled formulas. In such cases, it is important to balance the width and depth of the search to reduce I/O cost and control the size of intermediate relations. Obviously, it is a highly dynamic process. We have

**Conclusion 1.** *Multiple directioned search is suggested for non-recursive find-some queries.*

We then examine multi-directioned search in recursive query processing. We consider three cases based on different roles of constants in recursive rules, Case 1 : *constants within a recursive rule definition*, Case 2: *constants outside a recursive predicate*, Case 3: *constants in a recursive predicate.*

Case 1, *constants within a recursive rule definition*, is often formed by introducing search constraints which may contain some constants. For example, In Ex. 2, the constraint, *flying via large airports only*, may introduce some constants ("the judgement of large") in the recursive rule. In this case, performing selection first reduces the size of data relations to be iteratively worked on. For example, performing selection on $A$ using $c$ in $B$ for the recursive rule

$$R(x, y) :- A(x, w), R(w, y), B(w, c). \tag{8}$$

results in an equivalent recursive rule on a smaller *EDB* relation $A'$

$$R(x, y) :- A'(x, w), R(w, y). \tag{9}$$

where $A'(x, w) :- A(x, w), B(w, c)$. Therefore, we have

**Conclusion 2.** *Selection and simplification should be performed within the definitions of a recursive rule to reduce the size of the relation(s) to be iteratively worked on.*

For case 2, *constants outside a recursive predicate*, the processing should proceed from several directions towards the recursive predicate because a closure operation usually costs much more than conventional relational operations. For example, suppose there are query constants in $A$, $B$, $C_1$, $C_2$, $D$ and $E$, a query on the formula

$$A\ , B\ , R\ , (C_1\ ; C_2)\ , S\ , D\ , E. \tag{10}$$

where $R$ and $S$ are two recursive predicates, should be solved by proceeding in multiple directions from the provided constants towards the two recursive predicates. After such processing, the problem becomes feeding constants into a recursive predicate (case 3). Therefore, we have

**Conclusion 3.** *Processing should be performed first using query constants on the non-recursive predicates of a recursive query to reduce the starting size of the recursive processing.*

There is no sharp distinction in multi-directioned search between find-some and find-all queries in the first two cases, but the situation is different in handling case 3.

For a recursion involving a single compiled chain as in Ex. 2, two search directions can be considered: *from Chicago "forward"* or *from Vancouver "backward"*. For find-all queries, searching from both directions is not as efficient as

searching in one direction because even when there are some meeting points, the search must proceed until all the traversals from Chicago have gone to the west of Vancouver, or vice versa, or the most lag-behind one of one direction has passed over the most lag-behind one of the other, which are more complicated and usually involve even larger space than the single-directioned search.

However, multi-direction search is encouraged in find-some queries because search may terminate as long as enough meeting points have been found, and by starting from two end points, search usually terminates much earlier than that in one direction only. Therefore, we have

**Conclusion 4.** *Search from both ends of a chain in recursive query processing is usually beneficial in find-some queries, but not so in find-all queries.*

## 5. Conclusions

We studied the distinguished features in the processing of an interesting and important class of queries, *find-some queries*, in large knowledge-based systems. Most previous work in deductive database research is on find-all queries [1, 2, 5, 7, 8, 15], however, we feel that find-some queries represent a more practical and frequently used class of queries in knowledge-based applications.

Find-some queries share some general characteristics with the find-all queries in the deductive database query processing, such as *cluster-oriented compilation* and *set-oriented processing*. However, find-some queries have their own characteristics which suggest *heuristics-guided and integrated search methods*. We studied some important heuristics in query processing and three search methods integrating DB and AI search techniques. Our analysis shows that by incorporating heuristics and adopting integrated search methods, the query processing cost can be considerably reduced while the quality of answers can be improved.

The study also shows that the best search strategy for a find-some query is usually data- and query-dependent and should often be determined dynamically. More quantitative and performance studies are needed to clearly identify the relative importance of different heuristics and the merits of each integrated search method, which is also a topic in our future research.


## Acknowledgements

The work was supported in part by the U.S. National Science Foundation under Grant DCR-860-8311 and the President's Research Grant and a research grant of Centre for System Science of Simon Fraser University in Canada. The authors wish to thank Ghassen Z. Qadah and Ning Zhuang of Northwestern University for their helpful discussions.


## References


1. F. Bancilhon and R. Ramakrishnan, An Amateur's Introduction to Recursive Query Processing Strategies, *Proceedings of 1986 ACM-SIGMOD Conference on Management of Data*, Washington, DC, May 1986.
2. F. Bancilhon, D. Maier, Y. Sagiv and J. D. Ullman, Magic Sets and Other Strange Ways to Implement Logic Programs, *Proceedings of 5th ACM Symposium on Principles of Database Systems*, Cambridge, MA, 1986.
3. W. Clocksin and C. Mellish, *Programming in Prolog, 2ed.*, Spring-Verlag, 1984.
4. H. Gallaire, J. Minker and J. Nicolas, Logic and Databases : A Deductive Approach, *Computing Survey 16(2)*, , 1984.
5. G. Gardarin, Magic Functions : A Technique to Optimize Extended Datalog Recursive Programs, *Proceedings of the 13th International Conference on Very Large Data Bases*, Brighton, England, Sept. 1987.
6. J. Han, Pattern-Based and Knowledge-Directed Query Compilation in Recursive Data Bases, *Computer Science Department Technical Report No. 629 (Ph.D. Dissertation)*, University of Wisconsin at Madison, Dec. 1985.
7. J. Han and L. J. Henschen, Handling Redundancy in the Processing of Recursive Database Queries, *Proceedings of the 1987 ACM-SIGMOD Conference on Management of Data*, San Fransisco, CA, May 1987.
8. L. J. Henschen and S. Naqvi, On Compiling Queries in Recursive First-Order Databases, *J. ACM 31(1)*, , 1984.
9. C. Kellogg, A. O'Hare and L. Travis, Optimizing the Rule-Data Interface in a Knowledge Management System, *Proceedings of the 12th International Conference on Very Large Data Bases*, Kyoto, Japan, Aug. 1986.
10. L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, 1986.
11. R. Treitel and M. R. Genesereth, Choosing Directions for Rules, *Proceedings of 1986 AAAI Conference* , Philadelphia, PA, 1986.
12. S. Tsur and C. Zaniolo, LDL: A Logic-Based Data-Language, *Proceedings of the 1986 VLDB Conference*, Kyoto, Aug. 1986.
13. J. D. Ullman, *Principles of Database Systems, 2nd ed.*, Computer Science Press, 1982.
14. J. D. Ullman, Implementation of Logical Query Languages for Databases, *ACM Transactions on Database Systems 10(3)*, , 1985.
15. D. Warren, Efficient Processing of Interactive Relational Database Queries Expressed in Logic, *Proceedings of the 7th International Conference on Very Large Data Bases*, Cannes, France, 1981.
16. C. Zaniolo, Prolog : A Database Query Language For All Seasons, *Proceedings of the 1st International Workshop on Expert Database Systems*, Kiawah Island, SC, Oct. 1984.

# A Rule-Based Framework for Controlling a Robotic Workcell

M.E. Malowany A.S. Malowany

Computer Vision and Robotics Laboratory
Department of Electrical Engineering
McGill University
Montréal, Québec, Canada

## Abstract

A hierarchical approach to performing assembly and repair tasks within a robotic workcell is presented. A rule-based paradigm is chosen for realizing the upper levels of the processing hierarchy, and an implementation is given for one of these levels, called the Task level. New extensions of the Task-level system include a non-monotonic inference engine, the use of state variables permitting a static database size, the use of non-blocking functions for altering states, a parallel-processing capability enhanced by the use of non-blocking functions, and a rule set for a typical task (installing a component) which implements adaptive task performance so the goal can be reached from a variety of initial workcell states. The programming of the system was carried out in Franz LISP on a Sun 3 workstation running UNIX BSD version 4.3. Testing of the Task-level system with a simulated robotic workcell is described and implications of the test outcomes upon the feasibility of the approach are discussed.

*Keywords:* expert system, robotics, task-level programming.

## 1. Why a Rule-Based, Hierarchical Approach ?

Programming robots to perform manufacturing tasks the way humans do, with flexibility and intelligence, is a complex problem. It involves planning, executing, and monitoring activities in the robotic workcell. The use of a hierarchical or partitioned approach to this problem is attractive since the complexity is reduced by breaking the task into several simpler levels or parts. The parts may even be run in parallel on different machines. Such a hierarchical structure for robot control was first presented by Albus and Evans [1] in 1975, although the first realistic application was not reported until 1984 by Haynes et al [2]. The hierarchy is illustrated in the block diagram of Figure 1. The rule-based paradigm is one of the simplest ways of realizing the high-level, device-independent aspects of directing the workcell (such as action sequencing and resource sharing) in order to achieve a flexible, easily-updated, and portable front-end environment for users. These high-level functions are the province of the Problem and Task level processors in the hierarchy of Figure 1.

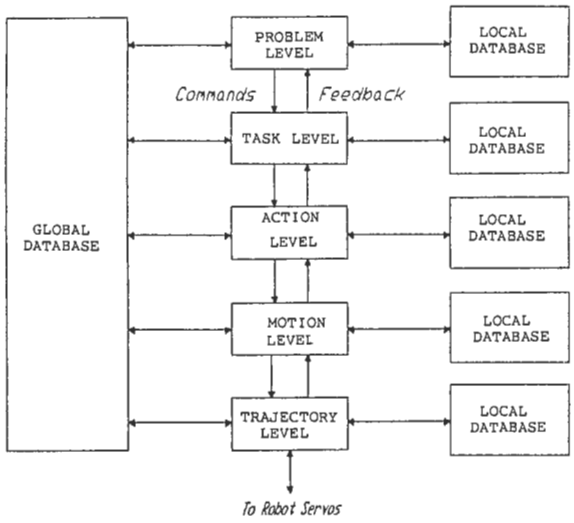



**Figure 1** General Structure for a Hierarchical Robot Control System

In this section, the approaches taken by three systems which use hierarchical or partitioned methods in robot control are summarized. Two of these systems (SAGE and WRAP) work together as a team, and the third system fits into the hierarchy of Figure 1 and served as an inspiration for the new Task-level system which is the subject of this paper. These examples illustrate the advantages of the approach as well as a variety of implementations, and share an emphasis on the concepts of state variables, real-time task performance, and parallel processing.

Separating the plan generation from the run-time execution of a task can be viewed as a form of partitioning the workcell control problem, especially if the run-time execution shares part of the decision burden by being adaptive. This will involve deciding what is to be done in the workcell and in what order at the plan generation level, then deciding at run-time how the list of actions and sequencing constraints is to be mapped onto the workcell, considering its present state.

Two systems which fit together to form such a description are the SAGE (Sequence Analysis by Graphical Evaluation) framework of Freedman et al [3] and the WRAP (Workcell ReAl-time Programming) environment of Carayannis et al [4]. Additional details on these systems may be found in Freedman et al [5].

SAGE is a facility which allows plans to be defined for repetitive jobs in the workcell and then automatically generates the corresponding minimum-time task sequence with the aid of a workcell model based on the theory of Petri Nets. SAGE's processing takes place in the planning domain and is off-line. SAGE was implemented as set of programs written in C-Prolog. It accepts input to define a model for the job to be done in the workcell, performs a structural analysis to ensure a solution exists, searches the problem task space (in which a node is the set of concurrent events at a given time) for cycles, and from these cycles builds task sequences from which the minimum-time member is selected.

WRAP oversees the actual performance of tasks in the robotic workcell in real time. WRAP is hierarchical, accepting input from an off-line top level (such as SAGE) called the Strategy level and processing this input with a Tactics level program which communicates to a Virtual Device level. The Virtual Device level isolates the Tactics level from the particularities of the workcell elements. A table of state variables is used to maintain information for sequencing steps within tasks. Each task is described by a "block", which resembles a rule in a rule-based system in that each block contains three parts: pre-conditions about the workcell which specify when the task is to begin execution, activities which instruct the Virtual Device level to do something in the physical workcell, and state changes which specify modifications to the state table to reflect the effects of the activities. The WRAP environment is implemented as a set of concurrently running processes on a set of hosts realizing true parallelism. The source code is written in YACC and in "C" running under UNIX 4.3 BSD. RCCL on a MicroVax II under UNIX was used for the virtual devices.

Unlike the above companion systems, the Task Request Interpreter (TRI) formulation of Klepko [6] fits into the hierarchical scheme of Albus and Evans as shown in Figure 1 and is truly rule-based. It is a Task-level system specific to assembly and repair of printed-circuit boards within a robotic workcell. It uses multi-tasking under UNIX to realize concurrency. In the next section, we outline an approach based on Klepko's TRI. Another system which addresses the problem of assembly in a multi-robot, multi-sensor environment is the NNS system developed by Alami [7]. This system is similar to the revised TRI in several respects. NNS is hierarchical, featuring a two-level hierarchy in which a Master Module (MM) controls the lower-level Specialized Modules (SMs). The Master Module is implemented in a form of LISP. Each Specialized Module runs on a separate processor and recognizes a set of commands from the Master Module that are implemented as Remote Procedure Calls. The NNS supports a feature for realizing concurrency called a "non-wait call" that is similar to the non-blocking function of the revised TRI. They are similar in that both the non-blocking function and the non-wait call return before their respective actions are completed.

## 2. Hierarchy and Commands

Several aspects of the hierarchy, commands, and rule-writing strategy of Klepko's original TRI carry over to the new Task-level system, the revised TRI, which is the subject of this paper. These common aspects are discussed in this section, followed by a short discussion of the differences between the two systems.

Both the TRI and the new system fit into the hierarchy of Albus and Evans at the Task level as shown in Figure 1. The Task level and the levels below it in the hierarchy are intended to run in real time. Only the Problem level is off-line. The Problem level is typically an automatic plan generator dealing in primitives just above the level which must interact with the real world. In the case of the TRI, the output of the Problem level is assumed to be a set of Task-level commands. These commands form a fixed Task-level vocabulary as listed in Figure 2. Being an interpreter, the TRI accepts task commands one at a time from the Problem level and breaks each command down into commands suitable for the next level below, the Action level. The set of Action-level commands are shown in Figure 3. Further processing at the Action, Motion, and Trajectory levels is device-dependent and is procedural rather than knowledge-based.

```
INSTALL (component-x)
ALIGN (component-x)
REMOVE-SOLDER (location-x)
SOLDER (location-x)
REVERSE (component-x)
REMOVE (component-x)
```

**Figure 2** The Task-Level Commands of the TRI

```
MOVETO (location-x, robot-y)
PICKUP (thing-x, robot-y)
PUTDOWN (thing-x, robot-y)
ACTIVATE (tool-x)
```

**Figure 3** The Action-Level Commands of the TRI

Note that the Task-level commands supported by the TRI are object-oriented rather than tool-oriented or robot-oriented in the way that the Action-level commands are. The Task-level commands specify what is to be done to the printed-circuit board but not how it is to be done. Specifically, which robots are to perform the task or what tools are to be used are not specified by the Task-level command. Assignment of tools and robots to the task is decided in real time by the TRI depending on the current state of the workcell as modelled in the database. Then the "how" of the task is synthesized by the TRI in terms of Action-level primitives. How such assignment and synthesis decisions will be made at execution time is embeded in the expert knowledge of the rules.

The extent to which the system can handle a diversity of initial states is determined by how many possibilities the expert rule-writer foresaw and incorporated into the rules. For example, the TRI uses a workcell model containing multiple

robots and causes them to work in parallel whenever possible for reasons of efficiency. However, if there is only one working robot (the others are broken), the rules are written so that they will instruct the single robot to achieve the task by itself. The new system handles a variety in the initial robot locations, tool locations, and robot-hand contents as well as in the number of working robots.

The rules in the previous TRI are divided into rule sets by task and by subtask. Context limiting is employed so that each task activates its own rule set. For example, the INSTALL task has a rule set distinct from that of the REMOVE task. The tasks are further divided into subtasks which may be performed nearly independently. A subtask is defined as a set of actions which can all be performed by a single robot without changing tools. For example, the INSTALL(component-x) task can be divided into 1) a board subtask, where a robot moves the circuit board between the hotplate and the board-jig using the board-grabber tool, 2) a solder subtask, where a robot dispenses solder at the component-x location on the board from the solder-paste-dispenser using the solder-paste-dispenser-holder tool, and 3) a component subtask, where a robot fetches the component-x from its feeder and places it in its location on the circuit board using the component-grabber tool. If there are as many free robots as there are subtasks, the subtasks may all start out simultaneously. Order constraints between subtasks are enforced by having the rules check state variables maintained in the database. If there are fewer free robots than subtasks, the rules will have to enforce sharing the robots among subtasks. Determining how to divide a task into subtasks and how to best share resources is treated heuristically by the human expert who writes the rules in the case of the TRI system. This contrasts with the approach taken by the SAGE/WRAP system where search algorithms for optimality are employed.

In a simulation of the TRI created for testing purposes, the Action-level-command output is printed on a terminal and sensory input is provided by the user via the terminal. That implementation of the TRI was written in Franz LISP for a VAX 11/750 computer.

The new system is essentially an extension of Klepko's TRI framework to utilize the concept of non-blocking functions rather than multiple LISP environments for realizing concurrency. In the previous TRI, the division of tasks into subtasks is mapped onto multiple processes at the Task level; multi-tasking is realized through the use of the LISP "fork" facility, creating a separate LISP environment for each process. The extension of the new system hinges on the use of non-blocking functions which allow a single LISP process at the Task level to direct multiple device controllers at the lower levels. Rather than run separate processes which each wait for every action they initiate to be completed, the new single process uses non-blocking functions to allow processing on other subtasks to take over while the rules relating to one subtask are blocked. This blocking is achieved by a state variable of the associated robot which is set to "busy" until completion of the action in the real-time environment of the workcell. This is similar to the way a semaphore is used in the dispatching of user jobs to identical hosts or print jobs to printers. Results would be the same for the new system and the previous TRI if program-execution time was negligible compared to robot-motion time which is on the order of seconds.

True parallel processing in the new system is assumed to take place at the Action level, with each robot having its own controller. In addition to these controller machines, a collision- avoidance co-processor is envisioned for the new hierarchical system and is being developed as a separate research project. This co-processor would monitor the robot trajectories and intercede before imminent collisions occur.

For simplicity, the processing of sensory feedback, or even simulated sensory feedback as implemented in the TRI, was not included in this implementation of the Task level processor. Similarly, error handling has not been incorporated yet, although this facility is essential for coping with the real-world and could be added in the future without major revision of the system.

## 3. Implementation of the Task Level

A new rule-based Task-level system for assembly and repair of printed-circuit boards in a robotic workcell has been designed and tested with a simulated workcell. The system consists of four parts, each of which was coded in LISP instead of using standard knowledge-engineering tools supplied with systems such as OPS5. The four parts are an inference engine, a database, a set of rules, and a set of application-specific functions. These embody the three components of a classical rule-based or "production" system in AI: the control or global knowledge, the data or declarative knowledge, and the operations or procedural knowledge (rules and functions.) [8]

### 3.1 Inference Engine

The inference engine was written to be general. Supplied with appropriate rules, facts, and functions, it can be used to perform a wide range of tasks. As in OPS5, the engine uses forward chaining.

Since the engine is designed to perform tasks in real time, the the state of the world will alter as the engine operates in conjunction with the physical workcell. This implies that the world model found in the database must also evolve. Hence, some way of retracting assertions (a form of non-monotonicity) is required. However, for the hierarchical system in which the engine is to operate, learning is not required. It is assumed that the actions to be performed to achieve the task are pre-defined. These requirements led to the use of a fixed-size database consisting of state variables. No variables are added to or removed from the database, but the values of the variables can be altered by the rules. This is the sense in which the engine is non-monotonic.

The engine was written assuming that it is both desirable and feasible to perform a task with the engine in real time. Unlike a task-planning system which generates a plan for future use, the engine, as a real-time system, must operate as if actions in the real world were happening in response to the firing of rules. It is assumed that physical-world actions are slow compared to the computing time required by

the engine. Hence, there is a period of time following the firing of an action-causing rule when other actions affecting the same workcell elements should not be ordered by other rules. This requires a mechanism to enforce sharing of workcell elements among rules. One could have the rules call functions which only return after the workcell elements have finished their actions (blocking functions). However, the engine was designed to utilize non-blocking functions. This attractive idea involves functions which only initiate world actions and set corresponding state variables before returning. The non-blocking-function option requires updating of the state variables at some later time either by the engine or by the rules. In this implementation, it is done by the engine. The state variables can be updated by comparing elapsed system time against heuristically pre-determined time intervals for each action or by interpreting sensory input related to the action. In this implementation, pre-determined time intervals are used for simplicity. This contrasts with WRAP, where a dedicated process is responsible for "awakening" a blocked task when the state of the world matches the pre-conditions.

Since the engine is to perform actions in the physical world, the rules need some kind of "procedural attachment" to link the action primitives to real-world actions. This has been achieved by having the engine evaluate the consequents, or right-hand sides of rules, which are written according to the syntax of LISP but may contain user-defined procedures. Function calls resulting from the evaluation of consequents manipulate the outside world, making it correspond with the world model maintained in the database.

Frequently constraints must be enforced upon the order of actions to accomplish a task correctly. For example, in pc-board repair, there must be a board put in the board-jig before solder-paste is dispensed there. A set of binary tags is maintained by the engine to facilitate the coding of rules for enforcing such order constraints. Tags are variables that indicate, by their state, whether a certain action has been performed. The set of tags allows the engine to know what it has already done and what remains for it to do. It is often desirable to fire a rule only upon the first occurrence of certain conditions. For example, it may be required in pc-board repair to have a free robot pick up the solder-paste-dispenser initially. Subsequently, the robots must be prevented from doing this each time they become free. Such a one-shot action is achieved by having a rule test for a tag-state in its pre-condition and set the tag to the opposite state in its consequent.

The control strategy of the engine is a linear-search, first-found first-fired method. The engine goes through the list of rules starting at the top, irrevocably firing the first rule whose pre-conditions are met. It then continues to check down the list until it finds another rule whose pre-conditions are satisfied. Upon reaching the end of the rule list, the engine checks all "busy" states and updates any that have expired before starting over at the top of the rule list. This continues until the task is completed or a deadlock is detected. The first-found first-fired control strategy requires the use of an agenda for the rules. This means the list of rules is ordered such that the early-firers are at the top and the late-firers are at the bottom. However, the fact that multiple passes through the list will be made obscures the actual run time order of firing. Hence, an additional mechanism is needed to supplement mere order in enforcing sequentiality. Tags, which have been discussed above, provide this additional mechanism.

The inference engine is a modified version of the system presented by Winston and Horn [9] which was monotonic and suitable for proofs. To be suitable for tasks, it had to be modified by implementing functional evaluation of rule consequents, changing the control strategy, and extending the matcher to work with the two-level-deep association-list format of the database.

### 3.2 Database

It is desirable to keep the database small in order to minimize the time required to search it. However, there is a trade-off between the amount of information encoded directly into the database and the complexity of procedures operating on the database.

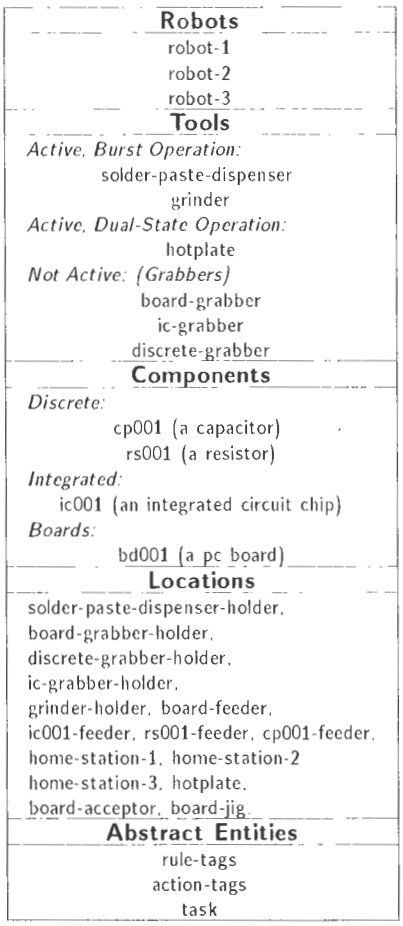



**Figure 4** Elements of the Workcell World Model

The database content is illustrated in Figures 4 and 5. Figure 4 shows the elements of the workcell world model which are encoded in the database and the categories into which these elements fall. The elements are either robots, tools, components, locations, or abstract entities (tags and task parameters.)

| Elements | Properties | Set of Possible Values |
|---|---|---|
| Robots | state | ready, busy, broken |
| | alloc | board-ops, paste-ops, grind-ops, component-ops |
| | holds | tool-x (except hotplate) |
| | location | location-w |
| | fine-loc | component-x-board-loc, no-fine-loc |
| | home | home-station-n |
| | time-stamp | seconds-N |
| | time-interval | seconds-N |
| Tools: *Active, Burst Operation* | holder | location-w (a holder) |
| | location | location-w |
| | state | ready, busy, broken |
| | time-stamp | seconds-N |
| | time-interval | seconds-N |
| Tools: *Active, Dual-State Operation* | state | ready, busy, broken |
| | time-stamp | seconds-N |
| | time-interval | seconds-N |
| | temp | cold, hot |
| Tools: *Not Active (Grabbers)* | holder | location-w (a holder) |
| | location | location-w |
| | contents | component-x, board-x, nothing |
| Components: *Discrete or Integrated* | tool | discrete-grabber, ic-grabber |
| | feeder | component-x-feeder |
| | fine-loc | component-x-board-loc |
| Locations | at | robot-y, nothing |
| | in | component-x, tool-z, bd001, nothing |
| Abstract Entities: | | |
| *Rule-Tags* | rule-n | armed, fired |
| *Action-Tags* | action-n | to-do, done |
| *Task* | component | component-w |
| | tool | tool-z |
| | fine-loc | fine-loc-w |
| | feeder | location-w (a feeder) |

**Figure 5** Properties Defined for Elements of the Workcell World Model

Figure 5 shows which properties are defined for each workcell-element category. Robots and some tools (the grinder, solder-paste-dispenser, and hotplate) are considered active elements and so must be declared busy while performing these actions. The general term "state" is used to indicate whether active elements are "ready" for action, "busy" doing an action, or "broken" and incapable of action. Since non-blocking functions are used, a mechanism is needed to decide when active elements are finished performing an action in the real world. This is achieved using elapsed system time in conjunction with initial timing data recorded in the database. Hence, each active element has a "time-stamp" indicating when its last action was started and a "time-interval" indicating how long the task takes to accomplish. It is not necessary to keep track of which Action-level command is currently being executed by an active element because the database is immediately updated to predict the state the workcell will take on upon completion of the action. The "busy" state inhibits using this predicted information until it becomes valid. Such timing information is not required for passive elements.

Figure 6 shows an example of the two-level-deep association-list format for the database. The database is contained in a list called "dB". Each member of "dB" is a list describing one element in the world model. The first element of each

```
(setq dB '(
     (object-1 ((property-1 value-1)
                (property-2 value-2)
               ... (property-n value-n)))
     (object-2 (  ...                  ))
...  (object-n (  ...                  ))))
```

**Figure 6** Format of the Database

member-list in "dB" is the object-name. The other element of each member-list in "dB" is yet another list whose elements are lists of length 2 which each give a property of the element and the associated value. The pair-lists at the deepest level of nesting enumerate all of the properties defined for the given object (as specified in Figure 5).

### 3.3 Application-Specific Functions

Each Action-level command is implemented as an action-function. The following is a general outline of what is done when an action-function is evaluated. First, a message is sent to the appropriate Action-level processor. Then the start time for the action is fetched from the real-time clock. Finally the workcell model in the database is modified to reflect the new state of the world which should result from the activities of the Action-level processor. Conservative time-interval estimates are built in to each action-function for setting the "time-interval" properties of the relevant active element in the database. This could be refined by incorporating intelligent sensor processing into the functions which would permit run-time error handling procedures.

A "pcb-repair" function was written as an even higher level of control over the general inference engine. The "pcb-repair" function is a command-interpreting loop which processes orders from the Problem level. The input is used to direct which rule set gets loaded into the "rules" variable before the engine is called. The argument of the command is also stored appropriately under the "task" object in the database. Undefined command inputs cause the loop to be started again at the top while escape from the loop is achieved by the command "exit". Currently, only the INSTALL task has been implemented with rules, but the shell is there in "pcb-repair" to deal with the other tasks as well.

### 3.4 Rules

An example of the format of the rule set is given in Figure 7. The rule set is contained in a list called "rules". Each member of "rules" is a rule which is in the form of a list with four elements, the first being the word "rule", the second being a rule name, the third being the "if" list, and the last being the "then" list. The first element of the "if" list is the word "if" while all subsequent elements are pre-conditions (no limit on the number). The first element of the "then" list is the word "then" while all subsequent elements are conse-

```
(setq rules '(
     (rule example1
           (if (pre-condition-1)
               (pre-condition-2)
           ... (pre-condition-n))
           (then (consequent-1)
                 (consequent-2)
             ... (consequent-n)))
     (rule example2 (if ...) (then ...))
 ... (rule examplen (if ...) (then ...))))
```

**Figure 7** Format of the Rule Set

quents (no limit on the number). The rule set developed for the INSTALL task consists of 47 rules in this format.

The pre-conditions of the rules must be written in a "matcher language" which describes symbolic pattern-matching operations to be performed against the database. Four main features of this language are associated with the symbols: "+", " ·", and "· " as well as the use of constant pre-conditions. The "+" tells the matcher to skip over any intervening forms. It will not inhibit a match if there are no intervening forms (i.e. an instance of nothing satisfies "+" as well as an instance of nothing). A pre-condition giving all three of the fields for the database search explicitly (i.e. object, property, and value are all constants) is a constant pre-condition and is permitted. The " ·" tells the matcher to match any atom found at this point in the pattern (wild-card) and to store the value of this atom tagged by a variable name. The construct has a list form: ( · variable-name) and the whole list form is matched by an atom. The "· " is the inverse of " ·". It causes the value of the variable stored previously to be substituted for the form: (· variable-name) before the matching is attempted.

The engine does not functionally evaluate pre-conditions but rather matches them against the database. It may be argued that functional evaluation is more efficient in the case where the value of a property is to be fetched. However, if it is the property that must be fetched given a known constraint on its value, then the symbolic pattern match is preferred since it allows a smaller database. In such a situation, functional evaluation would need reciprocal functions which would need reciprocal encodings of relationships to be added to the database.

The consequents of the rules, unlike the pre-conditions, are written in LISP since they are functionally evaluated. The consequents involve the application-specific functions, called in a manner consistent with LISP, as well as functions supplied by LISP. However, there is one non-LISP feature permitted in the consequents. It is the replace-variable form "<" from the matcher language. Before evaluation begins, these forms will be reduced to constants by the engine. An additional stipulation for the consequent of a rule is that only *one* action can be initiated involving a given active element by the entire right-hand side of the given rule. It would not make sense to initiate more than one action per element per rule because the functions are non-blocking. They return before the action is completed. The natural pre-condition that the element be in the ready state before it is commanded to perform an action would no longer be satisfied after the first action is initiated. This limitation of one action per active element per rule leads to a larger number of rules for a given task than would the blocking-function implementation.

**The first group of actions involves the solder-paste-dispenser:**

MOVETO (solder-paste-dispenser-holder, no-fine-loc, robot-y)
PICKUP (solder-paste-dispenser, robot-y
MOVETO (board-jig, component-x-board-loc, robot-y)
ACTIVATE (solder-paste-dispenser)
MOVETO (solder-paste-dispenser-holder, no-fine-loc, robot-y)
PUTDOWN (solder-paste-dispenser, robot-y)

**The second group of actions involves component-x:**

MOVETO (component-x-grabber-holder, no-fine-loc, robot-y)
PICKUP (component-x-grabber, robot-y)
MOVETO (board-jig, component-x-board-loc, robot-y)
PUTDOWN (component-x, robot-y)
MOVETO (component-x-grabber-holder, no-fine-loc, robot-y)
PUTDOWN (component-x-grabber, robot-y)

**The third group of actions involves the pc board:**

MOVETO (board-grabber-holder, no-fine-loc, robot-y)
PICKUP (board-grabber, robot-y)
MOVETO (board-jig, no-fine-loc, robot-y)
PICKUP (bd001, robot-y)
MOVETO (hotplate, component-x-board-loc, robot-y)
PUTDOWN (bd001, robot-y)
PICKUP (bd001, robot-y)
MOVETO (board-jig, no-fine-loc, robot-y)
PUTDOWN (bd001, robot-y)

**Figure 8** Sequence of Actions Needed for the INSTALL (component-x) Task

The content of the rules reflects the expert knowledge of the system. The sequence of action-functions to be evaluated is determined by the rules. The example for which a rule set has been written is the INSTALL task. The sequence of actions needed to INSTALL (component-x) is shown in Figure 8. The rule set divides this sequence among the functioning robots in the workcell (i.e. robot-y does not have to be the same robot for each action.) The rules also ensure that the actions will be performed in proper order (although they need not occur exactly as in the sequence of Figure 8.) It is assumed in the listing of Figure 8 that a board already exists in the board-jig before the first action is executed. The natural decomposition of the INSTALL task into three subtasks is also illustrated in this figure.

Recall the constraint that allows only one action per work-

```
(rule install-29
   (if (action-tags  (+(action-1 to-do)+))
       ((> robot-y)  (+(holds
                 solder-paste-dispenser)+))
       ((< robot-y) (+(alloc paste-ops)+))
       ((< robot-y)     (+(state ready)+))
       (board-jig          (+(in bd001)+))
       (board-jig        (+(at nothing)+))
 (then (moveto 'board-jig
               (value 'task 'fine-loc)
               '(< robot-y))
       (change 'action-tags
               'action-1
               'done)))
```

**Figure 9** An Install Rule

cell object among the consequents of any given rule. This constraint is responsible for the occurrence of meta-rules and following rules. The meta-rule, or initiating rule, and its following rules form a group. The group could have been expressed as a single rule if blocking action-functions had been used. The rules in such a group require many of the same pre-conditions and form a sequence of actions which follow each other closely in time. Hence, the conditions for the group to fire are found in the meta-rule which comes first in the sequence. It is usually a "big" rule with many pre-conditions. Figure 9 shows rule "install 29" which initiates the paste-dispensing group consisting of the following three rules (30 - 32.) The following rules are characterized by a pre-condition which says, in effect, "if the rule before me just fired." This keeps the sequence intact. The paste-dispensing rule group cited here implements the actions: move to the board-jig, dispense the paste, move away from the board-jig back to the robot's home, and de-allocate the robot. A brief description of the rule groups found in the INSTALL rule set is given by Figure 10. Owing to the use of the "busy" state, only one rule from a given rule group is fired on each pass through the rule set. The allocation rules are an exception, since allocation is not an action but is a database operation and so does not involve the "busy" state.

## 4. Testing

For testing purposes, a simulation approach was used where terminal input was treated as if it were from the Problem level and terminal output was considered to have been passed on to the Action level (the robot controllers). Only minor modifications to the application-specific functions would be needed to effect the actual hierarchical control of the workcell. Specifically, the interfacing of the action-functions and the highest-level control loop which interprets commands from the Problem level would be required.

The INSTALL(ic001) command was used for the testing. Figure 11 shows an excerpt of a transcript of the interactive testing session. The initial state of the workcell for these tests had three functioning robots, each with nothing in its hand

| Rule Numbers | Function |
|---|---|
| | *Singleton Rules:* |
| 1 | Make sure hotplate is on. |
| 2 | Fill task-description fields. |
| 3 | Allocate robot->board-ops initially if it has board-grabber (ideal). |
| 4 | Allocate robot->paste-ops initially if it has dispenser (ideal). |
| 5 | Allocate robot->component-ops initially if it has component-grabber (ideal). |
| 21 | Free a robot for board-ops if no board out in workcell. |
| 22 | Same as 21 except free component-ops robot instead of paste-ops. |
| 47 | Termination rule: sets indicator for engine to stop. |
| | *Rule Groups:* |
| 6 to 8 | Allocate robot->board-ops if it holds nothing, then get board-grabber. |
| 9* to 10 | Put away unwanted tool before board-ops allocation. |
| 11 to 13 | Allocate robot->paste-ops if it holds nothing, then get dispenser. |
| 14* to 15 | Put away unwanted tool before paste-ops allocation. |
| 16 to 18 | Allocate robot->component-ops if it holds nothing, then get component grabber. |
| 19* to 20 | Put away unwanted tool before component-ops allocation. |
| 23 to 28 | Fetch a board if none out in workcell (board-ops). |
| 29 to 32 | Dispense solder paste on board (paste-ops). |
| 33* to 38b | Place component on board (component-ops). |
| 39 to 46 | Move board to hotplate for heating and back to jig (board-ops). |

**Figure 10** Functional Summary of the Rules in the INSTALL Task

and located at its home station. There was no printed-circuit board out in the workcell, so one had to be fetched.

The testing was carried out on different multi-user computers running Franz LISP in the Computer Vision and Robotics Laboratory. During the debugging phase, a small four-rule test set was used on a Sun 3/160 Workstation and on a DEC VAX 750. As expected, this rule set gave fast results. An intermediate implementation with 27 rules was then tested on the Sun and on the VAX 750. The complete 47-rule version was run on a DEC VAX 780. The processing time required by the various rule sets was influenced by the multi-user system loading which varies. The values for "time-interval" incorporated into the non-blocking functions ideally would be the limiting factors in the execution time of the task if computation time was negligible. The durations assumed by the five action-functions in the INSTALL task are given in Figure 12.

The observed execution times of the system during testing were somewhat too slow for real-time applications. Results for a small system (47 rules) to complete a typical task (installing a component) suggest that it is not feasible to use the this approach in real time on the multi-user VAX or SUN computers. A factor of 100 in speed improvement must be obtained before using the system in real time becomes practical. Future research directions include modifying the code

```
(Firing RULE install-24)
(PICKUP { bd001 robot-3 }
   begins at : "Fri Aug 21 08:31:50 1987")
(Firing RULE install-34)
(PICKUP { ic001 robot-1 }
   begins at : "Fri Aug 21 08:33:09 1987")
(robot-3 finished its action as of :
                "Fri Aug 21 08:34:00 1987")
(robot-1 finished its action as of :
                "Fri Aug 21 08:34:00 1987")
(Firing RULE install-25)
(MOVETO { board-jig no-fine-loc robot-3 }
   begins at : "Fri Aug 21 08:36:18 1987")
(robot-3 finished its action as of :
                "Fri Aug 21 08:37:32 1987")
(Firing RULE install-26)
(PUTDOWN { bd001 robot-3 }
   begins at : "Fri Aug 21 08:39:11 1987")
(robot-3 finished its action as of :
                "Fri Aug 21 08:40:53 1987")
```

**Figure 11** Excerpt of the Output from an INSTALL(ic001) Task Simulation

| Action | Assumed Duration *(sec)* |
|---|---|
| PICKUP | 20 |
| PUTDOWN | 20 |
| MOVETO | 20 |
| ACTIVATE | 20 *(not hotplate)* |
| | 300 *(if hotplate)* |
| DE-ACTIVATE | 200 *(hotplate only)* |

**Figure 12** Assumed Duration Times for Actions

to run on a dedicated LISP machine (such as the Symbolics Model 3670) where we hope that execution times in the one-second range can be demonstrated.

## 5. Conclusion

A rule-based Task-level system has been implemented which participates in a hierarchical approach to robotic workcell control. This control strategy has been applied to automatic assembly and repair of printed-circuit boards. A rule set performing a typical task for this application (installing a component) was formulated and tested. This implementation involved developing a non-monotonic inference engine. Three major features, namely state variables, real-time task performance, and parallel processing, were incorporated. In particular, the use of non-blocking functions enhanced the effectiveness of these three concepts.

To address the problems associated with run-time errors in the workcell, sensors must be incorporated. This qualifies as another worthy avenue for future research. For such extensions, the updating of busy states should be performed by the rules instead of the engine. This would allow more flexible handling of any run-time errors that may be sensed in the real world.

The authors wish to acknowledge the financial support of NSERC and FCAR.

## References

[1] Albus J., Evans J., "A Hierarchical Structure for Robot Control" Proc. of the 5th Int. Symp. on Industrial Robots, pp. 231-237, 1975.

[2] Haynes L., Barbera A., Albus J., Fitzgerald M., McCain H., "An Application Example of the NBS Robot Control System" Robotics and Computer Integrated Manufacturing, Vol. I, No. 1, pp. 81-95, 1984.

[3] Freedman P., Malowany A., "Sequencing Tasks within a Robotics Workcell: from Feasibility to Optimality", IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, Victoria B.C., pp. 137-140, June 1987.

[4] Carayannis G., Blais B., Malowany A.S., Levine M.D., "A Real-Time Database for a Robotic Workcell Programming Environment" IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, Victoria B.C., pp.141-144, June 1987.

[5] Freedman P., Carayannis G., Malowany A., "A Graphical Perspective on Robot Workcell Programming", to be presented at Graphics Interface '88, Edmonton Alta., June 1988.

[6] Klepko R., Malowany A., "A Rule-Based Hierarchical Robot Control System" The Summer Computer Simulation Conference, Montreal P.Q., pp. 646-652, July 1987.

[7] Alami R., "NNS: A LISP-Based Environment for the Integration and Operating of Complex Robotic Systems", Proc. of the IEEE Conf. on Robotics and Automation, pp. 901 - 907, March 1985.

[8] Nilsson N.J., *Principles of Artificial Intelligence*, Morgan Kaufmann Publishers Incorporated, Los Altos, Calif., pp. 17-18, 1980.

[9] Winston P.H., Horn B.K.P., *LISP*, Second Edition, Addison-Wesley Publishing Company, Don Mills, Ontario, pp. 253-284, 1984.

# CURVED MONDRIANS: A GENERALIZED APPROACH TO SHAPE FROM SHADING

Walter F. Bischof[1] & Mario Ferraro[2]

[1]Department of Psychology
and
Alberta Centre for Machine Intelligence
and Robotics
University of Alberta
Edmonton, AB, T6G 2E9 Canada
and
[2]Istituto di Fisica Superiore
Università di Torino
Torino, Italy

## Abstract

Most shape-from-shading methods assume that surface reflectance is constant within large image regions. This assumption is violated in natural scenes with objects made from different materials. We present a method for recovering shape and albedo under the assumptions that surfaces are smooth and albedo is piecewise constant, as would be the case if a Mondrian image was painted on a smooth curved surface. Our method is based on combining Brooks and Horn's method for shape recovery with the recovery of albedo using stochastic relaxation.

**Keywords:** shape-from-shading, Mondrian image, stochastic relaxation.

## 1. Introduction

Computational vision aims at understanding how 3-D representations of the world can be reconstructed from information contained in 2-D images. Recent research has produced a variety of methods that allow the recovery of such shape information, including, for example, shape from stereo, shape from motion, shape from texture, and shape from shading. Most of these shape recovery problems are ill-posed (Hadamard, 1923; Torre & Poggio, 1986) in that existence, uniqueness and stability of solutions may not be guaranteed in the absence of additional constraints. In the case of shape from shading, for example, Pentland (1984) restricts the space of solution surfaces to umbilical surfaces to enable unique recovery of surface orientation from local variation of image irradiance. Brooks and Horn (1985) use a weaker constraint, requiring that the solution surface maximize some global smoothness measure, but constraints on surface shape have to be propagated non-locally. Both approaches are based on the image irradiance equation (1)

$$E(x,y)=\rho\lambda\underline{n}(x,y)\cdot\underline{s} \qquad (1)$$

where E is the image irradiance, $\rho$ the surface albedo, $\lambda$ the incident flux, $\underline{n}$ the surface normal, and $\underline{s}$ the illuminant direction (sun). They require that surface reflectance, in the imaging model (1) Lambertian reflectance, remains constant over a large area, and all variations in image irradiance are attributed to variations in surface orientation. Further, the two approaches provide no means for detecting that this assumption may be violated, except in some pathological cases. Such is the case, for example, with a black area in an image with $\underline{s}=\underline{v}$, the viewing direction, where surface orientation is computed being orthogonal to the viewing direction over some extended area, a geometrically impossible inference.

The aim of this paper is to investigate the feasibility of shape inference under weaker constraints than those imposed by either Pentland or by Brooks and Horn. There is no way of separating the two effects of surface orientation and surface reflectance in the general case, since any change in image irradiance can be attributed to a change in either of the two or a combination of both. Thus a photograph can be interpreted as a flat surface with changing albedo or as an image of a curved surface with or without changes in surface reflectance characteristics. In particular we will show that surface orientation can be successfully recovered if the surface is assumed to be smooth (as in Brooks & Horn) and surface albedo is piecewise constant (as opposed to globablly constant in Brooks & Horn).

Piecewise constant surface albedo can lead to discontinuities in image

irradiance and thus the borders of "albedo patches" could be detected by either looking for discontinuities in image irradiance directly (edge detection) or by looking for large errors in predicted image irradiance after an initial surface orientation fit. As we will show later, both these approaches lead to unsatisfactory results, inferior to our scheme in which surface orientation and surface albedo are recovered simultaneously.

Our approach is closely related, on one hand to Brooks and Horn (1985) for recovering surface orientation from shading, and to that of Marroquin, Mitter and Poggio (1987) and Geman and Geman (1984) for recovering the piecewise constant albedo map using stochastic relaxation. As discussed in the next section, the two processes run in parallel, each updating iteratively the input to the other process.

## 2. Model

We want to recover the orientation $\mathfrak{n}(x,y)$ of a smooth surface satisfying the image irradiance equation for Lambertian surfaces

$$E(x,y) = \rho(x,y)\mathfrak{n}(x,y)\cdot\mathfrak{s} \qquad (2)$$

over some domain $\Omega$, where $E(x,y)$ is the image irradiance, $\rho(x,y)$ the (piecewise constant) surface albedo, $\mathfrak{n}(x,y)$ the unit normal of the surface, and $\mathfrak{s}$ the (known) direction of a distant single point source (sun) with $|\mathfrak{s}|=\lambda$, the (known) incident flux. We will now discuss in turn how to recover the surface orientation map $\mathfrak{n}(x,y)$ and the surface albedo map $\rho(x,y)$ from equation (2).

### *2.1 Recovering the surface orientation map*

As shown in Brooks and Horn, recovering the surface orientation map $\mathfrak{n}(x,y)$ can be put as a variational problem where we try to minimize (3) with respect to $\mathfrak{n}$.

$$I(\mathfrak{n}) = \iint_\Omega [E(x,y)-\rho(x,y)\mathfrak{n}(x,y)\cdot\mathfrak{s}]^2 + \alpha[\mathfrak{n}_x^2(x,y)+\mathfrak{n}_y^2(x,y)]dxdy. \qquad (3)$$

In (3) the first term in the integral captures errors in predicted image irradiance and the second term captures smoothness of surface orientation, with $\alpha$ weighting the relative importance of this term and $\mathfrak{n}_x$, $\mathfrak{n}_y$ denoting partial derivatives. In terms of regularization theory (Tikhonov & Arsenin, 1977; Torre & Poggio, 1986) the second term is called the stabilizing functional and $\alpha$ the regularization parameter. The minimization formulation (3) differs from that of Brooks and Horn in that the term for surface albedo $\rho(x,y)$ is included and that no term is included for constraining surface normals to unit vectors (which will be included in the iterative formulas). The Euler-Lagrange equation associated with (3) is

$$(E-\rho\mathfrak{n}\cdot\mathfrak{s})\rho\mathfrak{s}+\alpha\nabla^2\mathfrak{n} = \mathfrak{0} \qquad (4)$$

where

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

is the Laplacian. Using the 9-point discrete approximation for the Laplacian

$$\nabla^2 \approx \frac{1}{6\epsilon^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix} \qquad (5)$$

and solving (4) for $\mathfrak{n}$ we arrive at the following iterative estimates for $\mathfrak{n}$:

$$\mathfrak{n}_{ij}^{k+1} = \mathfrak{w}_{ij}^{k+1}/|\mathfrak{w}_{ij}^{k+1}| \qquad (6)$$

$$\mathfrak{w}_{ij}^{k+1} = \bar{\mathfrak{n}}^k{}_{ij} + \frac{3\epsilon^2}{10\alpha}(E_{ij}-\rho_{ij}\mathfrak{n}^k{}_{ij}\cdot\mathfrak{s})\rho_{ij}\mathfrak{s} \qquad (7)$$

where $\mathfrak{w}_{ij}^{k+1}$ are the unnormalized surface normals and

$$\bar{\mathfrak{n}}^k{}_{ij} = \frac{1}{20}[4(\mathfrak{n}_{i,j+1}+\mathfrak{n}_{i,j-1}+\mathfrak{n}_{i+1,j}+\mathfrak{n}_{i-1,j}) + (\mathfrak{n}_{i+1,j+1}+\mathfrak{n}_{i+1,j-1}+\mathfrak{n}_{i-1,j+1}+\mathfrak{n}_{i-1,j-1})]. \qquad (8)$$

After every iteration of the surface orientation estimation we can obtain an estimate of the surface albedo $\hat{\rho}(x,y)$ from (2):

$$\hat{\rho}(x,y) = E(x,y)(\mathfrak{n}(x,y)\cdot\mathfrak{s})^{-1} \qquad (9)$$

except at self-shadow boundaries where $\mathfrak{n}(x,y)\cdot\mathfrak{s}=0$.

## 2.2 Recovering the surface albedo map

Given some estimates of surface albedo, $\hat{\rho}(x,y)$, we want to fit a piecewise constant albedo map, $\rho(x,y)$. That is, $\rho(x,y)$ should be constant except at boundaries between different regions, and the discontinuity boundaries should be spatially continuous. This problem could again be cast in terms of a minimization problem, but it is unlikely that the associated minimization functional is convex and thus can be solved using standard variational methods.

A way to solve this problem may be provided by the technique of stochastic relaxation introduced by Geman and Geman (1984) to restore images corrupted by noise. This technique is based on an analogy between images and lattice-like physical systems: pixels are considered the nodes of the system, grey levels and the presence of edges are viewed as states of the system; the probability of a state w is determined by the Gibbs distribution

$$\pi(w) = \frac{e^{u(w)/T}}{\sum_{w\in\Omega} e^{-u(w)/T}} \qquad (10)$$

where $\Omega$ is the set of all possible states w, u(w) is an energy function--described below--and T is a parameter called "Temperature" in analogy with statistical mechanics. Because of the equivalence between Gibbs distributions and Markov random fields, it is possible to show that the a posteriori probability distribution of a state w (i.e., an estimate of the original image given the data) follows an energy distribution with energy of the form

$$U(w) = V(w)+D(d,w) \qquad (11)$$

where V is the energy due to local node interactions and D(d,w) depends on the data structure. The maximum a posteriori estimate of the original image given the data, corresponds to the minimum of U(w). In the present application the states correspond to different configurations of the albedo map, while the "data" set is given by $\hat{\rho}(x,y)$ as estimated in (9). In the discussion of the albedo map recovery we first introduce the associated energy functions and then discuss the filling process using stochastic relaxation. The goal of the albedo recovery process is to determine the "true" surface albedo at every node of a dense grid in the viewer coordinate system. Between each pair of neighbouring nodes (in a 4-neighbour- system) a line element indicates the possible presence of a discontinuity boundary (see Figure 1).

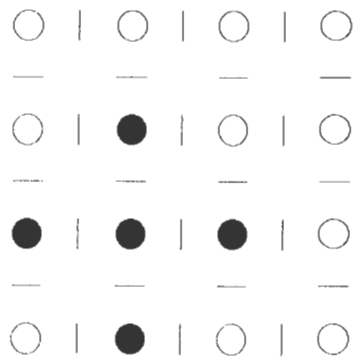


*Figure 1*
*Albedo node lattice with interstitial (horizontal and vertical) line lattice for modelling discontinuity boundaries. The black nodes indicate an "albedo node" with its four neighbour nodes.*

### 2.2.1 Energy functions

To model the piecewise constant albedo map we used a coupled node-line model (Geman & Geman, 1984; Marroquin et al. 1985) with the node process capturing penalties or differences between neighbouring albedo nodes and the line process capturing penalties on the local geometry of discontinuity boundaries.

Node process: At every node in the lattice the modelled albedo, $\rho_i$, should be as close as possible to the estimated albedo, $\hat{\rho}_i$ and the albedo value between neighbouring nodes should not differ unless there is a discontinuity boundary between them. To model the former we introduce the "albedo error" energy $D_i$

$$D_i = |\rho_i - \hat{\rho}_i|. \qquad (12)$$

To model the latter we introduce an "albedo variation" energy. Let i and j be two neighbouring nodes, $\ell_{ij}$ the line element between the two nodes, and let $C_i$ be set at all neighbours (the clique) of node i. Then the albedo variation energy $U_i$ is defined as

$$U_i = \sum_{j\in C_i} V(i,j,\ell_{ij}) \qquad (13)$$

where

$$V(i,j,\ell_{ij}) = \begin{cases} -1 & \rho_i=\rho_j,\ \ell_{ij}=\text{'off'} \\ +1 & \rho_i\neq\rho_j,\ \ell_{ij}=\text{'off'} \\ 0 & \ell_{ij}=\text{'on'} \end{cases} \qquad (14)$$

where $\ell_{ij}$ = 'on' or 'off' indicates the presence or absence of the discontinuity boundary element $\ell_{ij}$.

Line process: In modelling the spatial geometry of discontinuity boundaries isolated boundary fragments and clustered discontinuities should be highly penalized, whereas "good continuations" of boundaries should be less penalized.

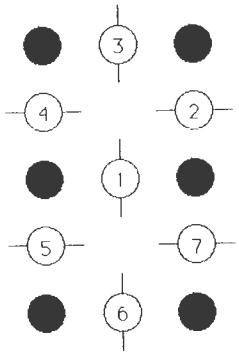



*Figure 2*
*Vertical line element (1) with six neighbours (2-7). Energy functions are defined on the two cliques (1,2,3,4) and (1,5,6,7).*

Figure 2 shows a horizontal line element (1) with its six neighbouring line elements (2-7), forming the two cliques (1,2,3,4) and (1,5,6,7). The energy functions $V_k(\ell_{ij})$ are defined for each clique with the values of $V_k(\ell_{ij})$ for all possible configurations, up to rotations, given in Figure 3.

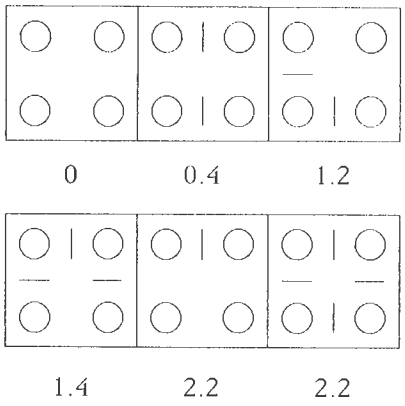



*Figure 3*
*Six possible line configurations and their associated energies.*

Isolated boundary fragments ($V_5$) and clusters of boundary fragments ($V_4$ and $V_6$) are highly penalized.

Given some estimated albedo values $\hat{\rho}_i$, we want to find the interpretation $I=(P,L)$ of albedo values $P=\{\rho_i\}$ and discontinuity boundaries $L=\{\ell_{ij}\}$ that minimize the combined energy function $E(I)$:

$$E(I) = \beta_1 \sum_i D_i + \beta_2 \sum_i U_i + \beta_3 \sum_\ell V_\ell \qquad (15)$$

where $\beta_1$, $\beta_2$ and $\beta_3$ are weighting factors.

### 2.2.2 Stochastic relaxation

The method used to minimize the energy function is based on the Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller & Teller, 1953) and on the annealing process described by Kirkpatrick, Gelatt and Vecchi (1983). It can be briefly characterized as follows. At a given temperature T a new value of $\rho(x,y)$ for each pixel is randomly chosen and the difference in energy $\Delta E=E_{new}-E_{old}$ is computed. The new value is accepted with probability

$$P = \begin{cases} \exp(\Delta E/T) & \text{if } \Delta E>0 \\ 1 & \text{if } \Delta E\leq 0 \end{cases} \qquad (16)$$

and the same process is repeated for the introduction or removal of line elements. The accepted value of $\rho(x,y)$ is then used in (7) to compute a new value of $\Omega$ which in turn is used to determine a new value of $\hat{\rho}(x,y)$, the estimated albedo, using (9). The temperature is then lowered following the schedule $T=T_0 ld(k/4)$ with k being the iteration number and ld the logarithm to the base 2. This method allows not just changes that decrease the energy but energy increasing changes as well, the latter becoming progressively unlikely as the process evolves. The process thereby avoids becoming trapped in local minima and should converge at least to states close to the global minimum (see Geman & Geman, 1984; Marroquin, Mitter & Poggio, 1987 for a discussion on the mathematical properties of stochastic relaxation process). It should be noted that in the original annealing method (Kirkpatrick et al., 1983) temperature is not changed until the average energy reaches equilibrium. We change temperature at every iteration regardless of whether equilibrium has been reached but compensate for the latter by using a conservative cooling scheme.

### *2.3 Combining the recovery of surface orientation and surface albedo*

There are several approaches on how to combine the recovery of surface orientation and albedo. Although we have chosen one particular approach, simultaneous recovery, it is useful to consider alternative approaches with their respective advantages and disadvantages.

Consider first the "albedo-surface" approach in which albedo, or at least boundaries of patches with constant albedo are recovered first. Given the model assumption of surface smoothness, discontinuities in image irradiance can only occur in two ways, at points with a discontinuity in albedo and at occluding boundaries of surfaces. The latter are assumed to be known in the model for the recovery of surface orientation. Hence boundaries of (constant) albedo regions could be detected using some edge detection mechanism. If this can be done reliably, then a mechanism for orientation recovery could be devised which independently estimates surface albedo in each of the regions. The reliability of such an edge scheme decreases in regions near the self-shadow boundary where image irradiance changes rapidly. Accordingly, our experiments using only an edge detection scheme proved not sufficient for recovering boundaries of albedo regions reliably. However, information from the edge detection mechanism could be used, for example, for initializing discontinuity line elements in the albedo recovery mechanism.

Consider next the "surface-albedo" approach in which boundaries of patches with constant albedo regions are detected after a full cycle of surface orientation recovery with a "constant albedo" model. Probable albedo boundaries can then be located using a surface discontinuity detector (Terzopoulos, 1985) or by detecting significant errors in predicted image irradiance, i.e., detecting positions where the first term in (3) has significantly large values. Given these boundaries and some appropriate albedo estimation for each region, surface orientation can then be re-estimated. One problem with this approach is that the performance of the discontinuity detectors is reduced by the previous smooth surface interpolation.

The approach we have found most successful involves the simultaneous recovery of surface orientation and surface albedo. Surface orientation recovery using the Brooks and Horn scheme is intertwined with surface albedo recovery using stochastic relaxation. In every iteration an estimate of surface albedo, $\hat{\rho}(x,y)$, produced by the orientation process is fed into the albedo process, which in turn produces an estimate of the piecewise constant albedo map, $\rho(x,y)$, used in the next iteration of the orientation process.

## 3. Experiments

Recovery of surface orientation and albedo was tested on synthetic, noise-free images of size $64^2$ and with $2^8$ intensity levels. Illuminant direction $\underline{s}$ was coincident with viewing direction and incident flux was kept constant $\lambda=|\underline{s}|=1$. Images and results of the recovery process are shown in Figure 4. The first surface (left panel) was a Mondrian-like sphere with seven patches of different albedo in the range 0.2-1.0, the second surface (middle panel) was an egg-shaped Mondrian-surface and the third surface (right panel)--used for control purposes--was a sphere with constant albedo.

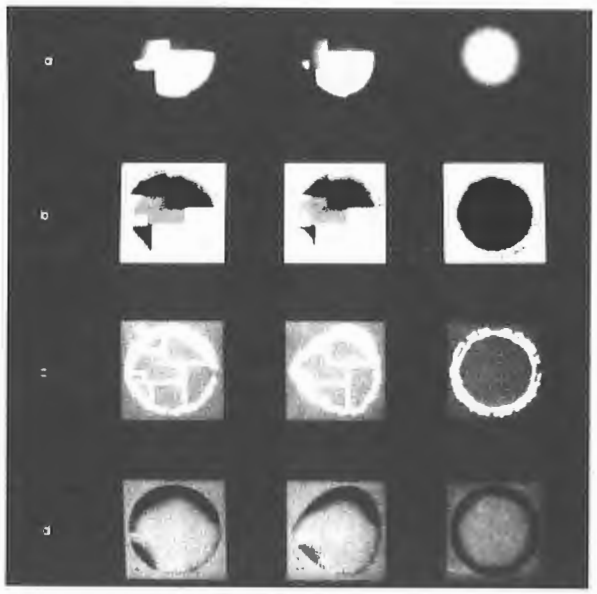


*Figure 4*
*Images and results of albedo/shape recovery. The left panel shows a sphere with piecewise constant albedo, the middle panel an egg-shaped object with piecewise constant albedo, and the right panel a sphere with constant albedo $\rho=0.6$.*
*(4a) Images*
*(4b) Recovered albedo map*
*(4c) Discontinuity boundaries found in albedo recovery*
*(4d) Error of z-component ($\underline{n}\cdot\underline{s}$) of recovered surface orientation. Grey areas indicate correct recovery, in dark areas recovered surface is too steep, and in white areas too flat.*

The images are shown in Figure 4a, the inferred albedo map in Figure 4b, the inferred discontinuity boundaries in Figure 4c, and the error of the inferred surface orientation in Figure 4d. The latter is to be interpreted as follows: in grey areas the z-components ($\underline{n} \cdot \underline{s}$) of the inferred surface normals are correct, in white areas too high (inferred surface orientation too flat) and in black areas too low (inferred surface orientation too steep).

All examples were computed using the same parameter values $\alpha$=0.3 (see (3)), $\beta_1$=0.05, $\beta_2$=19, $\beta_3$=13 (see (13)). Initial temperature was $T_o$=2 and was decreased according to the formula $T_k=T_o 1d(k/4)$ with k being the iteration number. All results shown in Figure 4 were obtained after 600 iterations.

The initial solution of the surface orientation $\underline{n}$ was correct at the occluding boundary and $\underline{n}$=[0,0,1] elsewhere. Surface albedo $\rho(x,y)$ was initialized to a random value for the first iteration and was allowed to vary in steps of 0.2. Discontinuity line elements were initialized randomly (with probability p=0.5) to 'on' or 'off'.

As can be seen from Figure 4, recovery of surface orientation and albedo was perfect except in a band of varying width near the occluding boundary. This effect was present with a relatively large range of other parameter values. The reasons for this failure will be discussed below.

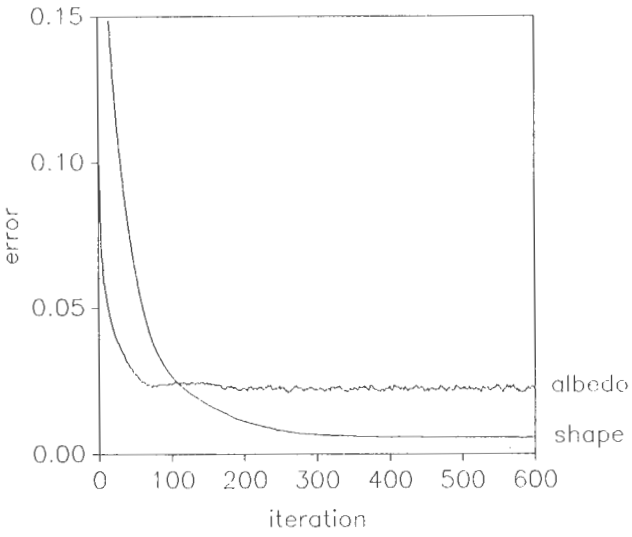



*Figure 5*
*Average error of estimatded surface orientation, $E[n_{inferred} \cdot n_{true}]$, and average error of estimated albedo, $E[\rho_{inferred}-\rho_{true}]$, over 600 iterations for egg-shaped surface with piecewise constant albedo.*

Figure 5 shows average errors of the inferred surface orientation, $E[\underline{n}_{inferred} \cdot \underline{n}_{true}]$, and average errors of the inferred surface albedo, $E[\rho_{inferred}-\rho_{true}]$ for the egg-shape Mondrian surface over iterations. They decrease exponentially, reaching an asymptote after about 200 iterations. The final errors are due to an imperfect fit near the occluding boundary, whereas there is virtually no error in shape/albedo recovery in the interior of the regions.

## 4. Discussion

The results show that our proposed method is capable of simultaneously recovering surface orientation and surface albedo for surfaces with piecewise constant albedo using shading information. This is achieved with a much higher computational effort--the results presented were obtained after 600 interations--than other known methods (about 50 iterations for the Brooks and Horn method and essentially a single iteration for Pentland's method).

The poor performance in recovering surface orientation/albedo near the occluding boundary is related to observations made by Smith (1982). The smoothing term in (3) is minimized by spheres but it does not propagate a sphere-like constraint in its discrete approximation, partially due to the fact that shape constraints are propagated only over few neighbouring nodes. The smoothing term alone (i.e., using $\alpha \to \infty$ in (3)) produces a cylinder-like shape if the initial solution is correct at the occluding boundary (see Smith, 1982, Figure 11). In our case this leads to a consistent over-estimation of surface albedo $\rho(x,y)$ near the occluding boundaries, as can be seen in Figure 4. This deficiency of the smoothing term was not apparent in Brooks and Horn (1985) as it was compensated by the first term in (3), the irradiance prediction term.

There are several possibilities to overcome this deficiency including the use of a multi-resolution approach to ensure sufficient propagation of shape constraints or the use of alternative propagators of shape constraints (Smith, 1982). We are currently working on improving our shape/albedo recovery along these lines.

In the section on combining the recovery of surface orientation with the recovery of surface albedo (section 2.3) we discussed three different approaches to combining the two mechanisms, eventually favouring the "simultaneous recovery" approach. We argued that the "albedo-surface" approach in which boundaries of "albedo patches" are

localized first using some edge detection mechanism was not reliable enough for identifying regions of different albedo. However, this is throwing out the baby with the bath water as edge information can be used to initialize and/or to constrain the introduction of discontinuity line elements in the recovery of the albedo map. The latter can be realized by introducing a data term similar to (12) for the line process. We expect an increase in efficiency of the recovery process using this information and are currently extending our method along these lines.

## Acknowledgment

W.F. Bischof was partially supported by grant 81.166.0.84 of the Swiss National Science Foundation. M. Ferraro was partially supported by the Consiglio Nazionale delle Ricerche.

## References

Brooks, M.J. & Horn, B.K.P. (1985). Shape and source from shading. Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles: Morgan, Kaufman, 932-936.

Geman, S. & Geman, D. (1984). Stochastic relaxation, Gibbs Distribution, and the Bayesian restoration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 6, 721-741.

Hadamard, J. (1923). Lectures on the Cauchy Problem in Linear Partical Differential Equations, New Haven, CT: Yale University Press.

Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by simulated annealing. Science, 20, 671-680.

Marroquin, J., Mitter, S., & Poggio, T. (1987). Probabilistic solution of ill-posed problems in computational vision. Journal of the American Statistical Association, 82(397), 76-89.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953). Equation of state calculations by fast computing machines. Journal of Physical Chemistry, 21, 1087.

Pentland, A. (1984). Local shading analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6, 170-187.

Smith, G.B. (1982). From image irradiance to surface orientation. Technical Note 273, Stanford Research Institute, Menlo Park, CA.

Terzopoulos, D. (1984). Integrating visual information for multiple sources for the cooperative computation of surface shape. In A. Pentland (Ed.) Pixels to Predicates: Recent Advances in Computational and Robotic Vision, Norwood, NJ: Ablex.

Tikhonov, A.N. & Arsenin, V.Y. (1977). Solutions of ill-posed problems, Washington, D.C.: Winston & Sons.

Torre, V., & Poggio, T.A. (1986). On edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8, 147-163.

# EXTENDING MOMENT ANALYSIS WITH DIRECTED ATTENTION TO HANDLE STRUCTURAL VARIATIONS IN CHARACTER RECOGNITION

Dale M. McNulty

Department of Information and Computer Science
University of California
Irvine, CA 92717
ARPAnet Address: MCNULTY@ICS.UCI.EDU
714-856-6360

## Abstract

A model (and instantiating program) is described which recursively combines the learning paradigms of conceptual clustering (Michalski, 1980) and learning from example to resolve the ambiguities of a real-world recognition paradigm. The model is based on neurophysiological and psychological evidence that the visual system is analytic, hierachical, and composed of a parallel/serial dichotomy (many, including Crick, 1984). Parallel processes in the model decompose the image into components and cluster the constituents in much the same way as moment analysis (Alt, 1962). However, the model extends the success of moment analysis, to handle simple alterations in stucture by serially investigating spatial relationships of component parts. The need for serial attentive resources results from a disagreement between the environment and the model on what constitutes the salient features at labeled level of the symbol. The model also overcomes other limitations of moment analysis, such as segmentation problems and difficulties matching moments.

## Keywords

Moment analysis, attention, conceptual clustering, learning from example

## 1.0 Introduction

Machine recognition is difficult in a perfect image environment, but the difficulty is compounded in a real-world paradigm where objects can, from one viewing to the next, undergo various types of transformations, including: translation, scale, rotation, and structural alterations.

The problem has been addressed from a number of viewpoints. One of these approaches, an image processing technique referred to as moment analysis, is demonstrably successful at recognizing alphabetic characters independent of translation, scale, and rotation (Alt, 1962). Unfortunately, this technique is not as successful when the domain of symbols includes simple structural variations, such as might result from occluding noise or sloppy printing technique. Additionally, the approach has potential problems of segmentation and difficulties with the moments themselves.

This paper describes a learning, recognition model, ZBT, (and its program instantiation) that extends the capabilities of the moment analysis technique by combining two documented learning paradigms: conceptual clustering and learning from examples. The basis of the model is the accumulating neurophysiological and psychological evidence that the visual system is analytic, hierarchical, and composed of both parallel and serial processing elements (Crick, 1984). ZBT simulates the biological approach by recursively combining the two learning paradigms in a two step process that zooms its attention from the most abstract form of an object to the most detailed aspects.

In the first step, ZBT incorporates the successful clustering aspects of moment analysis. Parallel, preattentive processes decompose an area of the image into meaningless, component blobs according to simple, fixed primitives. Other parallel mechanisms then compute features of the blobs that are invariant to the transformations of translation, rotation, and scale, and use these features to index into every level of the memory hierarchy looking for similar learned experiences.

In the second step, ZBT utilizes its attentional abilities to overcome the limitations of moment analysis and resolve a potential disagreement between the system and the environment. The disagreement revolves around the features that constitute the appropriate measures of similarity at the labeled level of an object. ZBT, by adopting the invariants of moment analysis, has chosen one set of features while the environment is (possibly) concerned with another set. The

second step in ZBT's process extends the capability of moment analysis by serially focusing attention on other features attempting to isolate the differences.

If the attentional step does not resolve the disagreement, ZBT recursively applies the two step process, again, searching for greater detail that might reveal a difference and, thus, define some aspect of a new concept. This recursive search creates a multi-level-indexed, is-a/part-of hierarchy that avoids the problems of exponential growth frequently associated with the extension of learning systems.

## 2.0 The Problem Domain

Briefly, the problem considered by this research is the learned recognition that confronts vision systems which must cope with imperfect perceptual environments, and, thus, potentially infinite domains. The problem has been simulated with the corresponding problem of recognizing binary, raster scanned images under certain transformations.

The following points summarize the problem domain. (A more detailed explanation can be found in McNulty, 1988).

(a) Input is a binary raster image of a target symbol selected from the English/Latin alphabet[1].

(b) The input can be accompanied by a label indicating the category, or class, of the input symbol that the teacher, or environment, prefers the system to associate with the target symbol. For example, the characters of Figure 1 would typically be labeled as members of Class A. (Category information, if provided by the teacher, is assumed to be consistent.)

(c) In the absence of an assigned label, the system's output should indicate the proposed class of the character encountered. However, with or without category information, the system should "learn" the symbol in such a manner that the learned information can be used later.

(d) The system should not learn verbatim information at every exposure. That is, it should learn incrementally (see Schlimmer & Fisher, 1986 for explanation and a justification concerning machine and biological systems).

(e) The system must be easily extensible. This implies that linear extensions should not increase learning/recognition times exponentially.

(f) The following variations in the target symbol should not impede recognition:

1) Translation: A symbol's location in the image field can vary from one experience to another.
2) Scaling: A symbol's size can also vary from experience to experience.
3) Rotation: 2D rotations of less than 45 degrees should be tolerated.
4) Linear structural alterations: Variations in structural characteristics as might result from omissions (due to noise or sloppy printing) are allowed. Figure 1 illustrates three example symbols selected as typical of one type of transformation. Object1 is a well-formed capital letter "A" composed of three straight-line strokes, labeled (only for purposes of the discussion): left side, right side, and brace. Object2 is a transformation of Object1 in which the connection between the right end of the brace and the right side has been broken. Object3 is identical to Object2 except the break between brace and side is wider.


Figure 1

## 3.0 Classification by Features - One Approach to the Problem

A somewhat successful image processing approach to character recognition is based on grouping the characters according to some set of features (Duda & Hart, 1973). Labeled "statistical recognition" by some authors, the term feature analysis is used here to include the following steps:

1) Decomposition, or segmentation, of the image into entities (see survey by Rosenfield, 1978).
2) Description of the entities in terms of features or attributes. Descriptors and the corresponding attributes they describe are usually selected for their invariant qualities over certain classes of image manipulation (Hu, 1962).
3) Classification or grouping of the objects by features (Duda & Hart, 1973).

This approach has the following qualities:

- Translational effects are negated by mapping the image (raster) data onto a symbolic data base.

[1] The domain is actually more inclusive but the initial focus, and that reported here, is on character recognition; thus, the model was able to build on the previous work of moment analysis.

- Scaling and rotation can also be negated by applying certain classification techniques on the proper invariant descriptors (discriminate analysis is an example of such a technique).
- Training (learning) and, thus, extensibility is possible.

One type of feature analysis employs moments as the invariant feature descriptors (Hu, 1962). Space restrictions prohibit a description of these computations (see Alt, 1962) but their significance as descriptors in the raster domain can be summarized by the following points:

- The moment sequence, $M_{pq}$, of a raster array, $f(x,y)$, uniquely defines $f(x,y)$ and conversely, $f(x,y)$ is uniquely determined by $M_{pq}$ (The Uniqueness Theorem) (Hu, 1962).
- The central (normalized) moments are invariant to various manipulations including: scale, translation, and rotation (Hu, 1962).

The technique of moment analysis works in this manner. First, the system is trained, one at a time, on the domain of objects and their respective classes. During training, the system groups the symbols according to the moments computed on the raster representations. After training, when exposed to a symbol in the absence of category information, the system will look through its memory for a previous, similar experience by comparing the stored moments of each class with the newly calculated moments. A match indicates that the new experience is probably of the matched class.

Alt (1962) demonstrated the usefulness of moments as descriptors by showing that it is possible to distinguish the standard 35 textual characters (26 alphabetic and 9 numeric) independent of translation, scale, and rotation. There have been other successes, including: Casey (1970, handprinted characters), and Hall, Crawford, & Roberts (1975, interpreting medical x-rays), but there are limitations to the technique (Lambert, 1969, multiple fonts, Hall, et al., 1976, matching optical scenes with radar images, and Dudani, 1977, aircraft identification). Limitations of moment analysis are generally of three sorts: (a) problems of matching features that are not invariant to certain structural alterations, (b) difficulties introduced by the segmentation phase, and (c) problems caused directly by the nature of the moments themselves.

The first two classes of problems are to be expected since The Uniqueness Theorem prescribes that different structural forms will have different moments associated with them. The problem is illustrated by Object2 whose moments will not typically be the same as those of Object1. Therefore, if the system has been trained on Object1 (i.e., exposed to the object and told that the respective class is A), then it is unlikely that an unlabeled occurrence of Object2 will be associated with the class of Object1. Further, if the system is then exposed to an unlabeled Object3 (i.e., after learning that Object1 and Object2 are both members of Class A), it is still unlikely that the system will associate the new object with either Object1 or Object2. Thus, training a moment analysis system to allow this type of structural alteration requires training the system on every minor, yet acceptable, change. This can be, at best, time consuming and, at worst, an exponential problem.

There are also potential problems of segmentation. For example, if characters are not segmented consistently from one viewing to another their moments will not match. The problem is comparable to that previously described. If the computed moments are inconsistent then the system will not consistently recognize even unaltered symbols.

Another potential problem of segmentation is typified by the letters "i" and "j". Normally, decomposition techniques do not segment these as whole letters (i.e., dot and stroke associated). The problem for the system in this case is that if segmentation does not present the descriptor with the entire symbol, the two components of the symbol must somehow be reassociated later. This puts a burden on the later stages of processing that was not intended to be part of feature analysis.

The nature of moments themselves create two additional difficulties. An analysis of Table 1 (reproduced from Wong & Hall, 1978) reveals one type. The table contains the (logarithms of) seven typical moments computed on three images - the original image and two possible transformations, one scaled and one rotated. Comparing the moments, one sees that they do not match from one image to another. The variation is caused by the digital encoding of data. That is, a scaled or rotated image can vary from the original image in the number of and locations of pixels. The result is that, contrary to their touted characteristic, moments do not necessarily match precisely from transformation to transformation, and, thus, a simple matching procedure is not possible. Moment analysis systems attempt to overcome this problem by employing different matching techniques that allow a little slack in the values (e.g., correlation), but this creates the opposite problem.

| Moment | Original | Scaled | Rotated |
|---|---|---|---|
| 1 | 6.24993 | 6.22637 | 6.25346 |
| 2 | 17.18015 | 16.95439 | 17.27091 |
| 3 | 22.65516 | 23.53142 | 22.83652 |
| 4 | 22.91954 | 24.23687 | 23.13025 |
| 5 | 45.74918 | 48.34990 | 46.13627 |
| 6 | 31.83071 | 32.91619 | 32.06803 |
| 7 | 45.58951 | 48.34356 | 46.01707 |

Table 1

If slack is tolerated in the moments then it is difficult to distinguish the slack associated with a variation between moments of similar objects, from the natural but close variation between two dissimilar objects. Additionally, as more variations in characters are learned, the individual features of dissimilar characters can overlap (e.g., distinguishing "6" from "b" and "I" from "1" across multiple fonts). Therefore, the more the system learns, the more difficult recognition can become.

In summary, moment analysis is a useful recognition tool in limited domains, but the technique depends completely on the moments to distinguish the similarities and differences of every object and its variation. Confusion between objects can result despite what appear to the human to be very simple structural differences between objects.

## 4.0 ZBT: Learning Levels of Features and Structural Detail by Directed Attention

ZBT extends the demonstrated capabilities of moment analysis and overcomes its weaknesses by augmenting it with a second step. In the first part of the process, ZBT clusters experiences invariant to scale, translation, and rotation. This much of ZBT's operation is similar to other moment analysis systems except that a simple match process is utilized instead of a more complex correlation procedure[2].

If a match is unique, then the category of the match is reported and recognition is complete. If, on the other hand, the match is not unique, or if there are labeling mismatches, then in the second part of the process, ZBT attempts to resolve the problems caused by clustering experiences by moments. It does this by focusing on structural detail in the current image that might distinguish the two structures and resolve the conflict. This part of the process reassociates the decompositions and refines the categories.

[2] Typically this range (slack value) is plus or minus 0.2% of the moment value. This range has been determined empirically for this scanning domain. As the resolution of the scanner or the size of the symbols to be recognized changes, the range must be adjusted proportionally.

ZBT can be summarized by the following functional flow:

a) In parallel, decompose the current attentive area of the image into component blobs.
b) In parallel, compute invariants of the blobs.
c) In parallel, reference memory with the invariants.
d) If the reminds are unique and there are no labeling conflicts, the recognition is complete
   else, serially gather and compare spatial relations of component blobs with those of the reminds.
e) If they match and there is no label conflict, the recognition is complete
   else, recurse.

The following discussion elaborates the previous outline description of ZBT with a simplified example of its operation. The example is composed of a sequence of four images: Object1, Object2, Object1 (again)[3], and Object3. The first three images are all labeled as members of Class A and the latter is unlabeled. Space restrictions prohibit a complete description, therefore, ZBT's operation on the first two images is summarized in outline form. (See McNulty (1988) for greater detail, justifications, and explanation of the computations.)

### 4.1 Learning An Aspect of the "A" Class

Encountering the first image (i.e. the image containing Object1), ZBT:

1) Decomposes (consistent with neurophysiological data and Julesz's, 1983, segregation based on local density) the image into a single, top-level blob that defines the "A" form.
2) Computes the invariants (moments) of the component blob.
3) Uses the invariants to index into memory for comparable stored experiences.
4) Finds no similar experiences. (Note that if an experience with similar moments had been stored, ZBT would have been reminded of it at this time.)
5) Stores the new experience in memory.

Encountering the second image (an image of Object2 labeled Class A), ZBT:

6) Duplicates steps 1-5. Notice that it is not reminded of the previous experience because the moments vary between the two objects. As discussed previously, one of the shortcomings of moment analysis is that the invariant measures of similarity at the highest level of the two objects, Object1 and Object2, do not conform to the

[3] The redundancy is required because ZBT is an incremental learning system.

grouping that a system predictive of its environment should possess. That is, the break in the structure of Object2 - as compared to Object1 - causes ZBT to cluster the two experiences separately, but the environment labels them identically. Thus, when ZBT indexes memory with the moments of the first level of the new image, it finds no learned experiences comparable to Object2.

7) Discovers that the label of the new experience is the same as a previous experience. Since Object2 has the same label as the unreminded Object1, ZBT has encountered a mismatch situation. (The attributes indicate that two distinct perceptions have been encountered, however, the experiences have been assigned the same category.)
8) Decomposes the single blob of the current experience into 3 component blobs (Figure 2) looking for something to differentiate it from the stored experience.
9) Indexes memory with the 3 component blobs.
10) Finds no similar experiences in memory. (This is because during the initial Object1 experience, ZBT had no reason to decompose the top-level.)
11) Serially attends to each component and its spatial relationship with each of its siblings. The following information concerning the relationship the brace has with the right side is recorded:

    LOCUS: 100 (the point on the blob closest to the sibling; this value is normalized as a percentage of the length of the major axis from the point of origin; in this case, the value 100 means that it is the right end of the brace[4])

    DISTANCE: 10 (the distance between LOCUS and the closest point on the sibling, again, normalized).

12) Stores the three new experiences in memory. It does this by placing them in memory as separate experiences according to their respective invariants, connecting them via two-way pointers to the higher level blob of which they are components, and interconnecting them with two-way pointers to signify the sibling relationships. Figure 3 summarizes the resultant memory.

[4] Internally, ZBT does not use terms such as "brace" or "right side." These are used in the text for clarity. ZBT's stored experiences go unlabeled until a label is assigned by the environment.

Figure 2

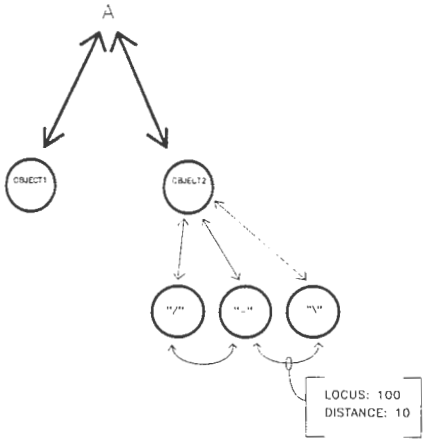



Figure 3:

At this point, ZBT has no comparable structural information for Object1. It can go no further, therefore, it terminates with knowledge of two comparably labeled experiences. Another Object1 experience will allow it to collect more information which will isolate the differences between the two experiences.

### 4.1.1 Identifying A Difference

Encountering the third image, a second occurence of Object1, ZBT is reminded of the top-level of the first Object1 experience, but it notices the labeling conflict once again. It zooms on the current image in an attempt to distinguish the two experiences. It decomposes the top-level, references memory with the three constituent blobs of Object1, and is reminded of the three constituents of Object2. It then proceeds to complete the tentative match by serially attending to the spatial relationships of the current blobs. It discovers the following relationship between the brace and the right leg:

LOCUS: 100
DISTANCE: 0

Comparing the reminded blobs with the current blobs, ZBT finds a difference in the distance value. ZBT has now detected the break in the "A" which distinguishes Object1 from Object2.

If ZBT's knowledge is to be predictive of its environment, it must associate these two experiences in memory. It does this by coalescing

the two memory structures, recording the differences in structure as cumulative statistical values. It records the maximum experienced DISTANCE value and its standard deviation (SD) from the previously stored DISTANCE value (see Figure 4).

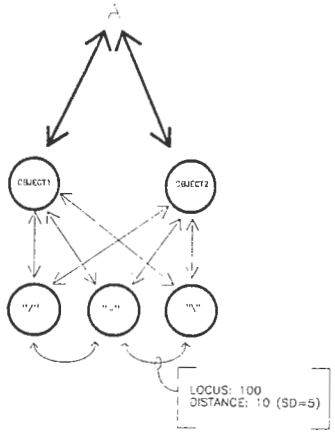



Figure 4

### 4.2 Recognizing Unlabeled Objects

ZBT begins processing the the fourth image, an unlabeled image of Object3, in much the same way as it did the third image. Because of the disparity between the moments of Object1, Object2, and Object3, it fails to associate the top-level blob of Object3 with the previous experiences. It zooms as it did before, but this time the reason is not to resolve a conflict, but instead to attempt to recognize an unlabeled image. It decomposes the top-level blob as it did before and is reminded of the three components of the second level of the coalesced experiences of Object1 and Object2. It then serially determines the spatial relationships between the current sibling blobs and compares them with those of the reminded experiences.

The comparison indicates that the DISTANCE value defining the critical relationship in Object3 exceeds the DISTANCE value of the matching remind. ZBT handles this in one of two ways. If the new DISTANCE value is less than a standard deviation from the stored DISTANCE value (i.e. less than or equal to 15), it will accept Object3 as a member of Class A. In this case, memory will be updated to reflect a new maximum value and standard deviation. If, on the other hand, the break exceeds the standard deviation (i.e. the break is larger than 15), Object3 will not be recognized as a member of Class A[5] and a new memory structure will be recorded.

## 5.0 Summary

An incremental learning model, ZBT, based on psychological and neurophysiological data was described that extends the capability of moment analysis by recursively combining the learning paradigms of conceptual clustering with learning from example to address the problem of real-world, character recognition.

Beginning with the least level of detail, primitive parallel mechanisms, invariant to translation, scale, and rotation, cluster meaningless blobs of information. The blobs are clustered, but remain unclassified until the environment or teacher provides a label (meaning). As a conceptual clustering engine, ZBT's invariant measures of similarity at any one level, may conflict with those of the teacher and, thus, negate the grouping. That is, (assuming a consistent environment), the problem is really a lack of disagreement between those attributes the environment uses to classify an object and the attributes ZBT utilizes to cluster.

Instead of redoing its clusters, ZBT refines the approximately correct grouping by decomposing the current level and serially searching for variable structural information among the components. If this does not resolve the conflict, ZBT recursively zooms on component blobs until memory is matched or it exhausts the levels of image detail.

ZBT is based on an apparent parallel/serial dichotomy in the visual system. ZBT's parallel processes, consistent with psychological and neurophysiological results, decompose the image along dimensions of contrast and line orientation. Serial focusing of attention isolates structural relationships between the components. The combination builds a hierarchy representing a distinction between two types of visual features. In the vertical direction the hierarchy varies in detail by clustering features which are invariant to the transformations of: translation, scale, and rotation. On the other hand, the horizontal direction contains information about features which are not invariant to those transformations.

[5] ZBT actually reports that there was a close match to the A category, but this aspect of ZBT's operation has not received a great deal of attention. Work has instead focused on experimentally verifiable aspects of the model. The concept formation literature has not, as yet, addressed the specific concept formation task confronting ZBT. Previous experiments have largely concentrated on the perception of well-formulated, natural concepts such as cups, bowls, birds, and animals (e.g., Labov, 1973). The statistical approach was incorporated into ZBT because of evidence presented by a number of researchers that subjects prefer an all-or-none concept formation strategy (e.g., Trabasso & Bower, 1968) and that the natural categories formed by people do not seem to have fixed boundaries (McCloskey & Glucksberg, 1978).

## Acknowledgments

I wish to thank Laura Yoklavich for her diligence and patience in proofreading this paper and others.

## References

Alt, F. (1962). Digital Pattern Recognition by Moments. In G.L. Fischer, et al. (Eds.), Optical Character Recognition. Spartan, Washington, D.C.

Crick, F. (1984). Function of the thalamic reticular complex: The searchlight hypothesis. Science, 8, 4586-4590.

Duda, R.O. & Hart, P.E. (1973). Pattern Classification and Scene Analysis. Wiley-Interscience, New York.

Hall, E.L., Wong, R.Y., Chen, C.C., Sadjadi, F., and Frei, W. (1976), Invariant Features for Quantitative Scene Analysis, Final Report. Image Processing Institute, Department of Electrical Engineering, University of Southern California, July.

Hu, M. K. (1962). Visual pattern recognition by moment invariants. IRE Trans. Information Theory, IT-8, February.

Julesz, B. (1983). Textons, the fundamental elements in preattentive vision and perception of textures. Bell Syst Tech J, 62, 1619-1645.

Labov, W. (1973). The boundaries of words and their meanings. In C.N. Bailey and R.W. Shuy (Eds.), New Ways of Analyzing Variations in English. Georgetown Press, Washington, D.C.

Lambert, P. F. (1969). Designing Pattern Categorizers with Extremal Paradigm Information. In S. Watanabe (Ed.), Methodologies of Pattern Recognition. Academic Press, New York.

McCloskey, M.E. & Glucksberg, S. Natural categoriess: Well-defined or fuzzy sets?. Memory and Cognition, 6, 462-472.

McNulty, D. M. (1988). Recognition of directed attention to recursively partitioned images. UCI Information and Computer Science Technical Report #88-08.

Michalski, R. (1980). Knowledge acquisition through conceptual clustering: a theoretical framework and algorithm for partitioning data into conjunctive concepts. International Journal of Policy Analysis and Information Systems, 4, 3, 219-243.

Rosenfield, A. (1978). Survey: Picture Processing 1977. Computer Graphics and Image Processing, 7, 211-242.

Schlimmer, J.C. & Fisher, D. (1986). A Case Study of Incremental Concept Induction. Proceedings Fifth National Conference on Artificial Intelligence, pg. 496-501.

Trabasso, T.R. & Bower, G.H. (1968). Attention in Learning. Wiley, New York.

# Speaker Normalization and Automatic Speech Recognition Using Spectral Lines and Neural Networks

Yoshua Bengio and Renato De Mori
McGill University, School of Computer Science
805 Sherbrooke Str. W., Montreal, Quebec, Canada, H3A-2K6

## Abstract

Artificial neural networks offer a new way to undertake automatic speech recognition. The Boltzmann machine algorithm and the error back propagation algorithm have been used to perform speaker normalization. Spectral segments are represented by spectral lines. Speaker-independent recognition of place of articulation for vowels is performed on lines. Results depend on the coding. The best results were obtained with coarse, multi-level coding, relative frequency and amplitude representation of spectral lines with a non-linear frequency scale. Samples were extracted from continuous speech for 38 speakers. The error rate obtained (4.2% error on test set of 72 samples with the Boltzmann machine, 6.9% error with error back-propagation) is better than results of previous experiments with the same data but using Continuous Hidden Markov Models (7.3% error on test set, 3% error on training set).

**Key Words** : Automatic Speech Recognition, Neural Networks, Boltzmann Machine Algorithm, Error Back Propagation Algorithm, Spectral Lines, Speaker-Independent Speech Recognition.

## 1. Introduction

Speaker Normalization (SN) in Automatic Speech Recognition (ASR) is known to be a difficult task.

It is known from speech analysis and perception that sonorant portions of speech spectrograms exhibit similar images when different speakers pronounce the same sound or the same sequence of sounds. Variations among speakers could be characterized by constrained variations of the frequency and energy of spectral lines as well as by insertion and deletion of lines. Such variations have been characterized by Hidden Markov Models (HMM) [14].

This paper investigates the use of multi-layer neural network models ([5][6]) for performing speaker normalization. Speaker-independent recognition of place of articulation is performed with the Boltzmann machine algorithm and the error back-propagation algorithm. The results are compared with the ones obtained with HMMs.

It will be shown that results depend on how spectral lines information is coded and that results better than the ones obtained with HMM (alone) can be obtained with a coding scheme that uses relative frequencies and amplitudes in a non-linear frequency scale derived from knowledge about ear sensitivity, as well as coarse and multi-level energy coding.

## 2. The Boltzmann Machine Algorithm

Boltzmann Machine Algorithms (BMA) (see [5] and [7] for more details and proofs) enable to learn the weights of connections involving hidden units, so as to optimize an information theoretic measure of the performance of the network. An example of structure for the network of the BMA is shown

in figure 1 ( this structure was used for our experiments with the BMA).

To obtain an output the BMA parallel network performs relaxation searches that simultaneously satisfy several weak constraints (the fixed inputs). The computation is performed by iteratively decreasing the value of a cost function which measures how well the network satisfies the constraints. That cost function is called energy, was introduced by Hopfield in 1982 [8] and is defined as follows :

$$E = - \Sigma_{i<j} W_{ij} Y_i Y_j + \Sigma_i B_i Y_i \quad (1)$$

where $Y_i$ is the output of unit **i**, $W_{ij}$ is the weight of the connection from unit **j** to unit **i** and $B_i$ is the threshold or bias for unit **i**.

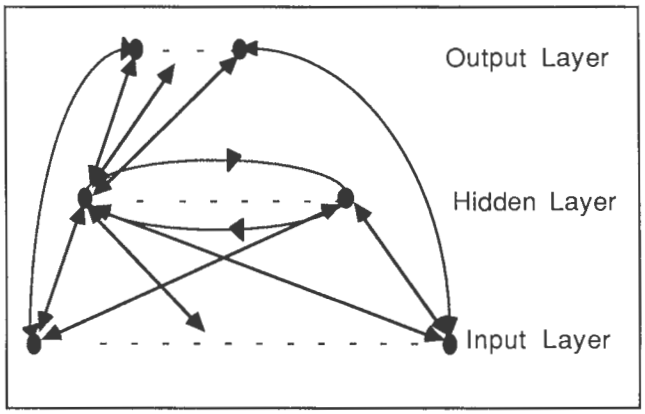



Figure 1 : *Input, Output and Hidden Layers for the Boltzmann Machine.*

Note that the connections of the network are symmetrical : $W_{ij} = W_{ji}$. The units can be divided into visible units (input/output units, connected to the environment) and hidden units. If the task of the network is to map an input domain to an output domain, the visible units can be divided into input and output units. If the task of the network is simply to complete a pattern, each visible unit can be input or output.

The BM is a stochastic network of units. They have a boolean (1 or 0) output chosen depending on the continuous weighted sum of their inputs. Each unit chooses the value 1 with a probability

$$p_k = 1 / (1 + e^{(-\Delta E_k / T)}) \quad (2)$$

where the local energy for each unit **k** is

$$\Delta E_k = \Sigma_i W_{ki} Y_i - B_k \quad (3)$$

$$(\text{i.e. } E = - \Sigma_k Y_k \Delta E_k ) \quad (4)$$

and T is parameter called temperature which is slowly decreased in the relaxation process. In an I/O cycle, the inputs (and possibly the outputs) are clamped to a certain value. The relaxation starts at a high temperature, units update their output, and the temperature is decreased according to a cooling schedule. This cooling relaxation is called simulated annealing and was introduced as a search procedure by Kirkpatrick & al. in 1983 [9].

A number of cooling schedules were proposed in the literature. The most frequently used is a fixed schedule [5][7]. A rate of cooling proven to make the system reach the best solution is logarithmic cooling ( i.e. T(t) = c / log (1+t) in Geman and Geman, 1984 [10]). Instead we chose a linearly increasing inverse temperature :

$$T(t) = \text{starting\_temperature} / (t+1) \quad (5)$$

With such a function, we obtained results identical to those obtained with the logarithmic cooling, with a gain in speed greater than 10.

To decide of a starting temperature, we computed the average energy barrier $\Delta E_k$ that each unit has to jump to make an upward move. We chose as starting temperature this average energy divided by 3. This means that at the beginning of the cooling, the average unit will have 42% probability to make an upward move in the energy (58% to make a downward move). By choosing a starting temperature which depends on the energy barriers, we ensure that the network will start at a high enough temperature even if very high energy barriers are created during the learning.

To stop the cooling (reaching thermal equilibrium), we waited for the network to stabilize. It is at this temperature that statistics about the co-occurrence of ON states are gathered, to be used in the learning phase when weights are updated. Statistics are gathered for a length of time inversely proportional to the average error of the system :

$$\text{len\_stat} = 4 / \text{sqrt} ( \text{average\_error} + 0.1 ) \quad (6)$$

This is because the learning is driven by the amount of error (discrepancy between actual output and desired output), and thus when the

error is smaller, more precise statistics are needed.

As suggested in the literature (Ackley, Hinton & Sejnowski 1985)[7], we modified the weights by a constant times the SIGN of the difference between the two probabilities $p_{ij}^+$ and $p_{ij}^-$ (the probabilities, averaged over all environmental inputs and measured at thermal equilibrium, that the units **i** and **j** are ON simultaneously, when the output units are fixed and not fixed, respectively) :

$$\Delta W_{ij} = \text{learning\_rate} \times \text{SIGN}\ (p_{ij}^+ - p_{ij}^-) \quad (7)$$

We also varied the learning rate as function of the average error as follows :

$$\text{learning\_rate} = 10 \times \text{average\_error} + 1 \quad (8)$$

## 3. The Error Back Propagation Algorithm

Figure 2 shows a model of the multi-layer perceptron to which the Error Back Propagation Algorithm (EBPA) was applied. For this model, the units are deterministic and compute a continuous output value (between 0 and 1).

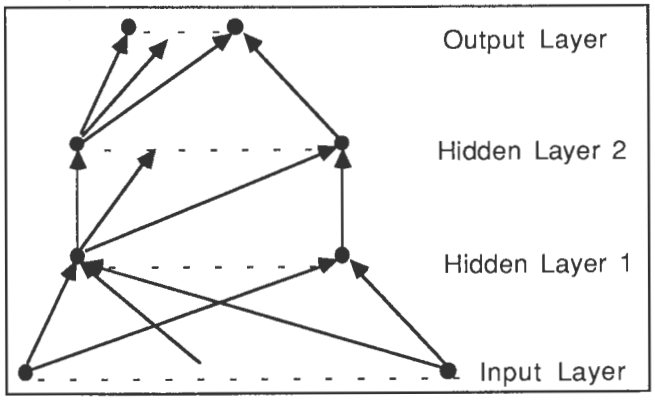



Figure 2 : *Feedforward Layers of the Error Back Propagation Algorithm.*

The basic version of this model is made of a certain number of feedforward layers of units (no recurrent connections : each unit is connected only to units in the next layer ). We implemented the algorithm as described in [6], with a momentum term in the learning rule :

$$\Delta W_{ji}\ (t+1) = m\ \Delta W_{ji}(t) + \text{learning\_rate}\ D_j\ Y_j \quad (9)$$

with Wji the weight from unit i to unit j on a layer above unit i, $Y_j$ is the output value of unit j, and $D_j$ is the derivative of the error signal as defined in [6], which is backpropagated from the output units back toward the input units. The constant **m** determines the influence of the past movements of $W_{ji}$ on the current change. We set it in our experiments to 0.95. Note that this procedure is equivalent to applying a low pass frequency filter to the landscape of the weight space, thus filtering out high curvatures which might have caused very slow convergence (with a small learning rate) or oscillations (with a large learning rate).

Note that the sigmoid function giving the output value $Y_j$ in terms of the weighted sum of inputs $X_j$, $f(X_j)=1/(1+\exp(-X_j))$ does not permit an output value $Y_j=f(X_j)$ of 0 or 1 unless $X_j$ is infinite, i.e. some weights are infinite. Thus desired output values were assigned a value close but not equal to 0 and 1, i.e. 0.9 and 0.1.

Since the network is updated deterministically, and we don't want it to learn the exact input/output pairs "by heart" (we want it to be able to generalize), we introduced some noise at the input units. We added a uniform random variable in U[-0.05,0.05] to each input value. Comparing results with or without the input noise, we observed slightly better performance when the inputs are noisy (actually best results are the same, but without noise, the error on the test set gets worse with more learning).

## 4. Application to the Recognition of Place of Articulation

The problem we chose to give to the neural nets was the recognition of the place of articulation of sonorant sounds based on spectral lines. Sounds are classified into three categories according to their place of articulation in the mouth : back position, central position or front position. The data consisted of 144 speech samples, 72 used for the training of the networks, 72 used only for testing. These samples were extracted from continuous speech, for 38 speakers (24 males and 14 females) pronouncing connectedly spoken letters and digits. A vowel (sonorant) segment in each sample was extracted and spectral lines were

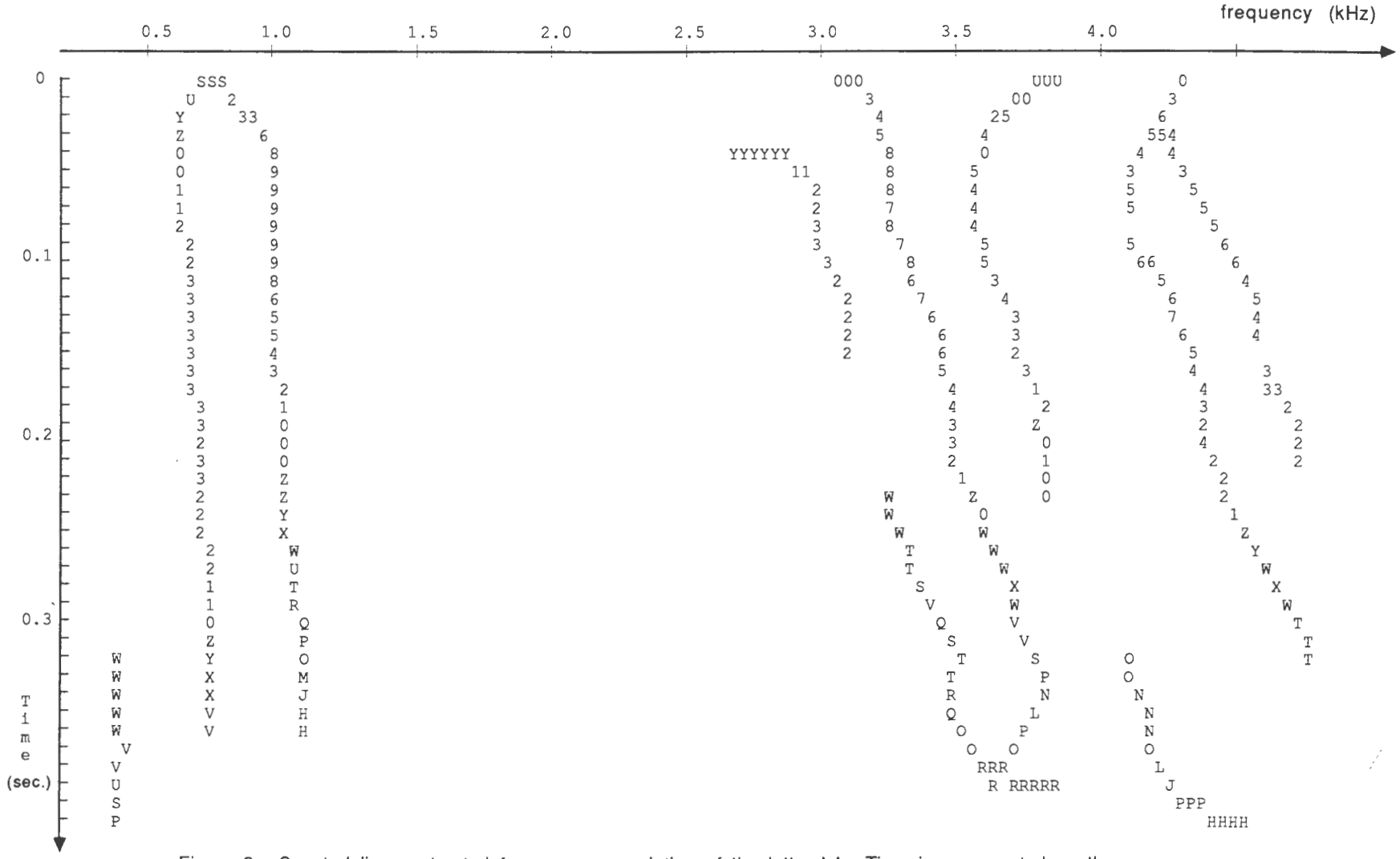



Figure 3 : Spectral lines extracted from a pronounciation of the letter 'a'. Time is represented on the vertical axis, frequency on the horizontal axis and energy by letters and digits (9>8>..>0>Z>Y>..>B>A).

computed in that segment. Details of segmentation can be found in [11].

Static representation of the speech data is based on spectral lines, already used by us for speech recognition tasks [14]. The original time signal is sampled at 20 kHz over 12 bits and its power spectrum (FFT every 10 ms) and zero crossings are computed. This information is used to identify the sonorant portions of the signal that exhibit resonances. This segmentation is based on rough spectral features that eliminates segments containing frication noise, silences and buzz-bars. The remaining spectrogram (time-frequency-energy pattern) is sent to the spectral lines extraction program.

## 4.1 Spectral Lines Extraction

To extract the spectral lines, the spectrogram is treated as an image. This image is processed by a thinning and a skeletonization algorithms, and then by a line tracing algorithm: these algorithms are described in (Palakal & De Mori, 1985)[12].

The output of the spectral lines extraction program is the description of a certain number (not fixed ahead of time) of spectral lines, each with its frequency and energy, for each time frame. We used only the average energy and frequency of the spectral lines over the sonorant segment. The first line is called the base (or anchor) line and is selected as the line of highest energy in the low frequencies. The others are usually described RELATIVE to the base line : the difference between their frequency and energy and those of the anchor line are provided to the neural nets. The base line frequency and energy are absolute.

Figure 3 shows the spectral lines extracted from the spectrogram of the letter 'a'. Time is represented along the vertical axis, each step corresponding to 10 msec. Frequency is represented along the horizontal axis. Letters and digits on the figure represent the amplitude of a given spectral component. For example, letter A represents only half the energy of letter B, letter Z represents half the energy of digit 0.

## 4.2 Coding

We describe here how the spectral lines and their energy/frequency description were coded. This coding has an impact on the efficiency (speed of learning and error) of the training process. The input nodes are first assigned to frequency intervals. One or several input nodes will represent a range of frequencies. The distribution of frequencies was inspired from an approximation of an ear model. Under 1 kHz (low frequencies), the characteristic frequencies grow linearly as follows :

```
frequency_index =
INT[(frequency - minimum_frequency) x
  low_freq_index_range/low_frequency_range]
                                        (10)
```

INT[x] represents the integer part of the real number x. Above 1 kHz, frequencies grow logarithmically (so that the higher the frequency represented by some input units, the larger the bandwidth they cover) as follows :

```
frequency_index =
   INT[c1 x log(frequency - 1000) - c2]      (11)
```

where c1 and c2 are chosen to make the minimum and maximum high frequency fall on the boundaries of the high frequency indices. Thus for high frequencies, the frequency index (on the grid) is a logarithmic function of the frequency.

To represent energy we decided to use several nodes within a frequency range, each representing a certain energy level. Thus the input nodes can be seen as points on a 2-dimensional grid of frequency and energy (c.f. figure 4). The distribution of energies was chosen empirically so as to make the number of samples falling in each range about equal. A typical number of levels chosen was 10. Better results are obtained with this multi-level coding of the energy than if we use only one node per frequency range, coding the energy continuously. In that case the system converges slower and makes a larger error, as shown in table I.

In addition to exciting one node for each energy/frequency input, we excite neighbouring nodes (with intensity decreasing with distance). Thus for each input spectral

line, there were 12 other nodes on the grid which would get excited. This strategy called coarse coding (or neighbourhood code, [15]) gives better results (c.f. table I) than the simple excitation of one energy/frequency node per spectral line.

Note that the energy level coded into the network is always relative to the energy of the base line, and we don't provide the network with the absolute energy of the base line. In fact a separate set of nodes (coding only for low frequencies, and no energy levels) is provided to represent the base line. Typically there are 15 nodes to represent (low frequencies) the base line, 15 (frequencies) x 10 (energy levels) = 150 nodes to represent low frequencies and 15 (frequencies) x 10 (energy levels) = 150 nodes to represent high frequencies. This makes a total of 315 input nodes as shown on figure 4.

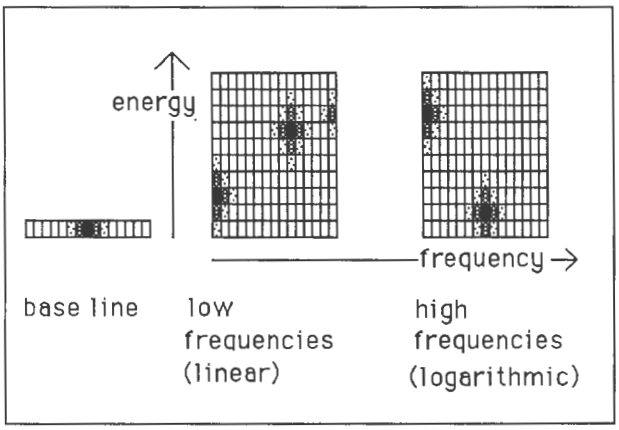



Figure 4. *Coarse coding (neighbours get excited), showing base line units, low frequency units and high frequency units.*

The values we want to assign to nodes on the input layer are continuous (between 0 and 1). However, the BMA units have binary outputs. To code a continuous value, we use the following stratagem. Since the output of a BMA unit **k** is chosen to be 1 (instead of 0) with the continuous probability $p_k$ (see definition **(2)**), we can assign the desired continuous input to $p_k$. This procedure will be effective since the BMA operates with a long relaxation cycle (often 10 to 100 cycles in our experiments). This method also has the advantage of automatically providing some noise to the input ( something that we had to implement with the error back propagation algorithm ).

## 4.3 Experimental Results

The programs were written in C on a VAX 8650. In some experiments with the error back propagation algorithm, the network would get stuck at local minima of the weight space. In this case, we restarted the learning process, with new random initial values for the weights.

The error back propagation algorithm was faster but less accurate than the Boltzmann machine algorithm. The results obtained with the two methods are shown in table II. Table II shows a comparison of speeds for the best performances of the two algorithms. These results were obtained with 2x200=400 hidden nodes (2 hidden layers, 103600 connections) for the error back propagation algorithm, and 100 hidden nodes (75490 connections) for the Boltzmann machine algorithm.

| | Boltzmann Machine | Error Back Propagation |
|---|---|---|
| Experiment 1 (coarse coding, multi-level energy coding, and relative frequencies) | 4.2% | 6.9% |
| Experiment 2 As before but No Coarse coding | 9.7% | 9.7% |
| Experiment 3 No multi-level energy coding | 8.3% | 15.3% |
| Experiment 4 Absolute instead of relative frequencies | 5.6% | 6.9% |

TABLE I : Error on Test Set for Various Coding Schemes

| | Boltzmann Machine | Error Back Propagation |
|---|---|---|
| Speed (CPU time for one sample) | 3 sec. | 0.21 sec. |
| Error on Test Set | 4.2 % | 6.9% |

TABLE II : Speed and Best Results for the 2 Algorithms.

The error shown in the tables is not the sum of the square of differences between actual and desired output values but rather the classification error. Even when the classification error reaches 0%, we can still do some learning. Every few learning cycles, the test set would be presented (and no learning would occur). After the error on the test set reached a certain low value and the error on the training set reached 0%, further learning cycles followed by presentations of the test set would result in oscillations of the error on the test set, with eventually an increase in this error.

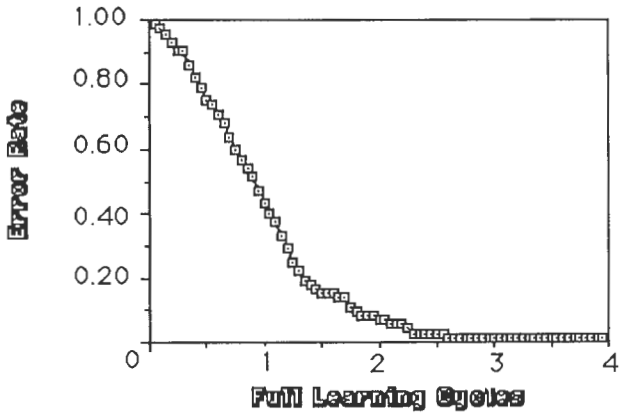



Figure 5 : *LEARNING ; Error Rate for the Training Set vs Number of Full Learning Cycles, for the BMA*. Note that the error stayed at 1.4% until the 20th cycle, when it reached 0% error.

Typically, the Boltzmann machine would converge to 0% error on the training set after about 20 presentations of the training set. The shape of the learning curve (error rate vs number of presentations of the training set) is shown in figure 5. Notice that initial learning is very fast, while the last few error percentage points need many more training cycles. After convergence we would continue learning for a certain number of presentations (about 10) to try to get slightly better results on the testing set. The error back propagation network would converge to 0% error on the training set after about 10 presentations of the training set. The running time shown in table II is for the presentation of one speech sample (from the test set).

We can compare the results obtained with those of experiments performed with the same data (and using the same spectral lines coding) but using Hidden Markov Models plus some rules (incorporating knowledge about the expected position of the spectral lines for each class). Note that the use of empirical rules makes the method less generalizable. These results were reported in (Merlo, De Mori & Palakal, 1986)[14], (c.f. table III).

| place of articulation | HMM alg. | HMM alg. + rules | |
|---|---|---|---|
| Back | 5% | 3% | error on |
| Central | 2% | 2% | |
| Front | 2% | 1% | training set |
| average | **3%** | 2% | |
| Back | 16% | 6% | error on |
| Central | 2% | 2% | |
| Front | 4% | 4% | test set |
| average | **7.3%** | 4% | |

TABLE III : Comparison with HMM Algorithm, Same Data

## 5. Conclusion

The results reported in this paper using connectionist models compare favourably with experiments performed on the same problem using HMM. They encourage us to try to apply neural networks to more difficult tasks. However, one weakness of these experiments is the small number of samples used for training the networks, especially with regard to the large number of connections in the networks, i.e. the memory capacity of these networks.

Note that with our neural nets it was always possible to reach 0% error on the training set. This suggests that we could get better results

on the testing set if we used a larger training set. The testing samples would be very close to samples which would have been already seen in the training set.

Although results with the Boltzmann machine algorithms were better than with the error back-propagation algorithm, the running time of the Boltzmann machine when simulated on a sequential machine is prohibitive for large networks *or problems that require many more training cycles* because of their complexity.

In future experiments concerning more complex problems, we consider training several small networks, thus each having a small number of connections. Each of these network could be specialized to a certain set of properties of the speech signal, for example the network described in this paper could be one of those, specialized on the analysis of spectral lines in vocalic segments of the speech signal. The output of these trained networks could then be combined as inputs in a hierarchy of higher level modules, that could themselves be successively trained. This approach would permit to bypass the problem of the increasing training period for increasingly complex problems.

---

## BIBLIOGRAPHY


[1] McCulloch,W.S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5,* pp.115-133.

[2] Hebb, D.O. (1949). *The organization of Behavior.* New York : Wiley.

[3] Rosenblatt, F. (1962). *Principles of neurodynamics.* New York : Spartan.

[4] Minsky, M. & Papert, S. (1969). *Perceptrons.* Cambridge, MA : MIT Press.

[5] Hinton, G.E., Sejnowski, T.J., & Ackley, D.H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn* (Tech. Rep. No. CMU-CS-84-119). Pittsburgh, PA : Carnegie-Mellon University, Department of Computer Science.

[6] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representation by error propagation. *Parallel Distributed Processing : Exploration in the Microstructure of Cognition. Vol. 1 : Foundations.* (pp. 318-362) Cambridge, MA : MIT Press.

[7] Ackley, D.H., Hinton, G.E., & Sejnowski, T.J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science, 9,* pp. 147-169.

[8] Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA, 79,* pp. 2554-2558.

[9] Kirkpatrick, S., Gelatt, C.D.Jr., & Vecchi, M.P. (1983). Optimization by simulated annealing. *Science, 220,* pp.671-680.

[10] Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 6,* pp.721-741.

[11] De Mori, R., Laface, P., & Mong, Y. (1985). Parallel algorithms for syllable recognition in continuous speech. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 7,* pp.56-69.

[12] De Mori, R., & Palakal, M. (1985). On the use of taxonomy of time-frequency morphologies for automatic speech recognition. *Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, CA,* pp.877-879.

[13] Naccache N.J., & Shinghal, R. (1984). SPTA : A proposed algorithm for thinning binary patterns. *IEEE Transactions on Systems, Man and Cybernetics, 14 (3),* pp. 409-419.

[14] Merlo, E., De Mori, R., Palakal, M., & Mercier, G. (1986). A continuous parameter and frequency domain based Markov model. *International Conference on Acoustics, Speech and Signal Processing,* pp. 1597-1600.

[15] Prager, R.W., Harrison, T.D., & Fallside, F. (1986). Boltzmann machines for speech recognition. *Computer Speech and Language, 1,* pp. 3-27.

[16] Bengio, Y., (1987). *Connectionist Models Applied to Automatic Speech Recognition.* M.Sc. Thesis, McGill University, Montreal, Canada.

# TOWARD THE AUTOMATED SYNTHESIS OF NONDETERMINISTIC PLANS USING GENERALIZED CONDITION/EVENT NETS

Dennis R. Bahler
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, 27695-8206
(919) 737-3369
Internet: drb@cscadm.ncsu.edu

## ABSTRACT

Existing planning techniques do not easily allow for nondeterminism in plan execution, and while there exist plan representations that accommodate nondeterminism, using these representations for plan synthesis itself is also problematic. In applications requiring the mutually exclusive use of scarce resources by multiple agents, automated planning is an even more difficult problem. One major problem is that actions by one agent may make it impossible for other agents to fulfill their goals; in a correct plan either such potential interference must be eliminated by sequentialization, or else the ability of one agent to perform its actions must "persist" beyond the actions of other agents. In this paper we introduce a technique of both representing and automatically synthesizing such high-level multi-robot plans through the use of a new subclass of Petri nets we call Generalized Condition/Event (GC/E) Nets. To avoid creating incorrect nets, we have identified an important new property, which we call **persistence**, which is essential to plans and therefore to any representation of them.

**Keywords:** planning, representation, algorithms

## 1. Introduction

One longstanding but stubbornly elusive goal of research in automated problem-solving has been the ability to "program" autonomous agents by providing only the specification of desired goals, rather than explicitly setting forth the tasks to achieve a goal and a sufficient ordering on those tasks. Automated planning for multiple agents in an application requiring the mutually exclusive use of scarce resources, such as robot collision avoidance, is an even more difficult problem. In this paper we introduce a technique of both representing and automatically synthesizing such high-level multi-robot plans involving nondeterminism, by modeling such plans using a new subclass of Petri nets we call Generalized Condition/Event (GC/E) Nets and then constructing such nets automatically. We also show through examples the functionality of our approach for representing and solving important problems in resource allocation in environments containing more than one agent.

Many modern planning systems, from Sacerdoti's NOAH system [8] to Chapman's TWEAK [2], employ partial orderings on their events, and thus may be modified relatively easily to produce plans intended for execution by multiple agents. However, these systems do not easily allow for nondeterminism. On another track, one or another variant of Petri nets [7] has been suggested as a representation for robot plans, including nondeterministic plans, in some recent work [3, 4, 5, 6]. This work, however, gives no indication of how nets may be employed *during the process of synthesis itself.*

Indeed, such nets commend themselves to plan representation for several reasons. First, nets are inherently parallel, in that enabled events that do not interact may occur independently. Second, although synchronization may be easily modeled, nets are asynchronous and thus independent of any particular notion of global time. Third, nets are nondeterministic and different orders of event execution may arise from the same net. Finally, pre- and postconditions of operators may be modeled naturally.

In simplest terms, a plan is a set of event occurrences that is partially ordered in (expected future) time. In a correct plan, if two sets of event occurrences, or sub-plans, are incomparable in the partial ordering, then it must be the case that they can occur in any order with respect to each other. Thus, if both sub-plans have all their preconditions established, the occurrence of one of the sub-plans cannot spoil the preconditions of the other. Any pair of sub-plans for which such destructive interference cannot be ruled out must be explicitly ordered by any correct plan synthesis algorithm. Because they fail to account for this fact, large classes of nets have properties which render them unsuitable as plan representations, and algorithms which construct nets having these properties will be incorrect. In formal terms, many nets have reachable states from which no state can be reached that satisfies the goal. To prevent this occurrence, we have identified an important new property of nets, which we call **persistence**, which is essential to plans and therefore must be preserved in synthesis. The basic idea is that the enablement of one sub-plan must "persist" beyond the occurrence of the other sub-plan. Specifically, persistence of enabled sub-orderings of events must be preserved by any net-growing algorithm that produces a plan involving nondeterministic mutual exclusion.

In addition, this work is the first to our knowledge that attempts actually to *plan with nets* by constructing net representations themselves automatically given only pre- and postcondition specification of initial and goal situations. This work and the much more detailed treatment in [1] attempt to fill this gap between modeling and synthesis.

## 2. An Example Problem

Consider a universe of two robots, each of which must move to a different location on a factory floor. Assume that a set of discrete locations can be identified on the floor, and that these locations in turn generate a set of paths between locations. If two or more paths intersect at a location, then there is the possibility of collision of robots using these paths. For example, if robot *r1* is to travel from location *a* to location *b*, and robot *r2* is to travel from *c* to *d*, and if the path from *a* to *b* intersects the path from *c* to *d* at a location *z*, then there is the possibility of collision if both robots try to occupy *z* at the same time. Any solution must prevent either robot from entering *z* (call it the "collision zone") whenever the other robot occupies that space. At the same time, a solution should recognize that it doesn't matter which robot enters the zone first; in that sense the solution should be **nondeterministic**. Let us see how to construct a nondeterministic plan for this problem, given only the specification of the floor layout and the initial and desired locations of the robots.

## 3. Situations

We begin by setting forth a framework in which to address this kind of planning issue. The situational model presented in this paper is based on a simplified form of first-order logic with certain restrictions on the structure of its formulas First-order logic was chosen because it provides a very general framework for expressing and solving planning problems.

We say that a **literal** is a (possibly negated) predicate symbol (e.g., CLEAR, ON, PATH, etc.) in a first-order logic, together with the arguments of that symbol. Arguments may be either variables (X,Y,Z, etc.) or constants (*a,b,c*, etc.). A **situation** is a conjunction of literals. For example, the formula

$$\text{ON}(c,a) \wedge \text{ON}(a,t) \wedge \text{BLUE}(c) \wedge \text{ORANGE}(a) \wedge \text{ROUND}(t) \wedge \textit{AFTERNOON}$$

is a description of a situation in which a blue object $c$ is on top of an orange object $a$, object $a$ is on a round object $t$, and the time of day is *afternoon*.

For our example problem, we can specify the floor layout by the formula

$$\text{path}(a,zone) \wedge \text{path}(zone,b) \wedge \text{path}(c,zone) \wedge \text{path}(zone,d)$$

The initial positions of the robots may be given by

$$\text{at}(r1,a) \wedge \text{at}(r2,c)$$

and the initial situation is the conjunction of the above two formulas with a literal safe(*zone*) denoting that *zone* is presently unoccupied.

The goal situation may be represented by the formula

$$\text{at}(r1,b) \wedge \text{at}(r2,d)$$

It should be clear how this representation can be extended to more complex topographies and more than two robots by adding literals to the specification of initial and goal situations.

## 4. Operators

We allow for two types of operators, propositional operators and predicate operators. A propositional operator may be written in production rule form as $R\colon \alpha_1, \ldots, \alpha_n \rightarrow \beta_1, \ldots, \beta_m$. $R$ is the label or name of the rule. The left-hand-side $\alpha$'s consist of literals indicating the preconditions for execution of this operator; the right-hand-side $\beta$'s consist of literals indicating the postconditions which will be true after execution.

To generate the set of predicate operators, we need the concept of a template which is subject to variable substitution. We call such a template a **schema**. An **operator schema** $R$ is a template of the form

$$R(v_1, \ldots, v_k)\colon \alpha_1(u_{1,1}, \ldots, u_{1,a_1}), \ldots, \alpha_n(u_{n,1}, \ldots, u_{n,a_n}) \rightarrow \beta_1(w_{1,1}, \ldots, w_{1,b_1}), \ldots, \beta_m(w_{m,1}, \ldots, w_{m,b_m})$$

Here, $R$ is the label of the rule; $k$ is the number of arguments in the rule label; $n$ is the number of left-hand-side literals in the rule; $m$ is the number of right-hand-side literals. Each predicate $\alpha_i$ has an argument vector $\{u_{i,1}, \ldots, u_{i,a_i}\}$ associated with it and similarly for the $\beta$'s. $a_k$ is the number of arguments of the $k^{th}$ left-hand atom of the rule; $b_k$ is the number of arguments of the $k^{th}$ right-hand atom.

An operator results when a **ground substitution** is applied to an operator schema. A **substitution** is a finite set of form $\{t_1/v_1, \cdots, t_n/v_n\}$ where the $v_i$ are variables, $t_i$ are terms different from the $v_i$'s and no two elements have the same variable after the /. A **ground substitution** is a substitution none of whose $t_i$'s are variable symbols. If $R$ is an operator schema, and $\Theta = \{t_1/v_1, \cdots, t_n/v_n\}$ is a substitution, then the operator $R\Theta$ is obtained from $R$ by replacing each occurrence of each variable $v_i$ in $R$ by the corresponding $t_i$. $R\Theta$ is called an **instance** of $R$. If $\Theta$ is a ground substitution $R\Theta$ is called a **ground instance** of $R$. An **operator** is a ground instance of an operator schema. Where no ambiguity would result, a predicate schema may be abbreviated without the argument lists as $R\colon \alpha_1, \ldots, \alpha_n \rightarrow \beta_1, \ldots, \beta_m$ similar to a propositional operator.

Let us illustrate these notions with a trivial example. Imagine a world of identical blocks on a table and a robot arm able to move these blocks around, stack them, and so on. A schema $R$ for placing a block X from on top of a block Z to on top of a block Y might be

PUT(X,Y,Z) : CLEAR(X), CLEAR(Y), ON(X,Z)
→ CLEAR(X), CLEAR(Z), ON(X,Y)

Applying the substitution $\Theta = \{a/X, b/Y, c/Z\}$ to the schema, an operator $R\Theta$ that places a block $a$ from on top of a block $c$ to on top of a block $b$ is

PUT(*a*,*b*,*c*) : CLEAR(*a*), CLEAR(*b*), ON(*a*,*c*)
→ CLEAR(*a*), CLEAR(*c*), ON(*a*,*b*)

Note that there is no restriction against the same literal occurring on the left-hand-side and the right-hand-side of the same rule. Consider, for example, a schema of the form

*R*(...,*r*,...): ...,free(*r*),... → ...,free(*r*),...

to indicate that resource $r$ must be free in order for $R$ to execute and will become free again after $R$ has terminated. The status of $r$ during execution of $R$ is indeterminate, but it is incorrect to assume that $r$ is free.

Simply by considering a subset of 3-space as a resource, the exclusive acquisition of which is a necessary precondition for an operator, we can accomplish the sort of exclusion we want in our two-robot example. For that problem we can employ the following schemata:

enter_zone(ZONE,LOC-1,ROB):
at(ROB,LOC-1), path(LOC-1,ZONE), safe(ZONE) → in_zone(ROB,ZONE)

leave_zone(ZONE,LOC-2,ROB):
in_zone(ROB,ZONE), path(ZONE,LOC-2) → at(ROB,LOC-2), safe(ZONE)

Note that these schemata are valid no matter how many locations are of interest, no matter what the topography of paths and intersections among paths may be, and no matter how many robots we wish to consider.

## 5. Plan Grammars and Planning Problems

A **plan grammar** $\Sigma$ consists of a set of operators, i.e., a set of ground instances of operator schemata, where each schema $R$ can be represented as a production rule of the form given in the preceding section.

What, then, is a planning problem and how may it be specified? A problem may be thought of as a plan grammar and two situations: the initial situation (denoted $S_1$) and the desired final or goal situation (denoted $S_2$). To unify the situational specification into the plan grammar, we append to the plan grammar two special kinds of rules, called **situational rules**. If $S_1 = \beta_1 \wedge \cdots \wedge \beta_m$ is a situation, a **situational rule for $S_1$** is a rule of the form:

$$R_{S_1}\colon S_1 \rightarrow \beta_1, \ldots, \beta_m$$

Similarly, if $S_2 = \alpha_1 \wedge \cdots \wedge \alpha_n$ is a situation, a **situational rule for $S_2$** is a rule of the form:

$$R_{S_2}\colon \alpha_1, \ldots, \alpha_n \rightarrow S_2$$

As we have seen in our running example, the plan grammar model is flexible enough to provide a convenient means of addressing some important kinds of open problems in automated planning research.

## 6. Nets and Net Representations

### 6.1. GC/E Nets

Situations, operators, and plans all have graphical representations, and we now show how to combine these representations to form nets. The planning process we use grows these nets automatically from situational specifications, using a plan grammar. We introduce a type of net called a **generalized condition-event (GC/E) net.**

**Definition:** A **generalized condition-event net** or **GC/E net** is $N = (P,T,E,\mu)$ where $P$ is a set of places, $T$ is a set of events, $P \cap T = \emptyset$, $E \subseteq (P \times T) \cup (T \times P)$ is the edge relation, and $\mu \subseteq P$ is called a **marking** of N. □

A net is typically represented graphically with a set of circles representing places, a set of rectangles representing events, a set of directed arcs representing the edge relation, and a token ( $\bullet$ ) residing in all places $p \in \mu$.

Let $N = (P,T,E,\mu)$ be a GC/E net. For $x \in (P \cup T)$, $\bullet x = \{y \mid (y,x) \in E\}$ is called the **pre-set** of $x$, and $x \bullet = \{y \mid (x,y) \in E\}$ is called the **post-set** of $x$. For $X \subseteq (P \cup T)$, $\bullet X = \bigcup_{x \in X} \bullet x$ and $X \bullet = \bigcup_{x \in X} x \bullet$.

An event $t \in T$ in a net N is **enabled** in a marking $\mu \subseteq P$ if $\bullet t \subseteq \mu$. The notation $\mu - t ->$ denotes that event $t$ is enabled in marking $\mu$. An event that is not enabled is **disabled.** The **firing** of an enabled event $t$ in a marking $\mu$ leads to a marking $\mu' = (\mu - \bullet t) \cup t \bullet$. $\mu'$ is called the **successor marking** of $\mu$ under $t$. $\mu - t -> \mu'$ denotes that the firing of enabled event $t$ in $\mu$ leads to successor marking $\mu'$.

The empty event sequence $\epsilon$ is enabled in every marking $\mu$ ( $\bullet \epsilon = \emptyset$ ), and $\mu - \epsilon -> \mu$. For a sequence of events $\omega \in T^*$ and an event $t$, $\mu - t\omega ->$ if and only if $\mu - t -> \mu'$ and $\mu' - \omega ->$.

A marking $\mu'$ is **reachable** from a marking $\mu$ if there exists an enabled sequence $\omega$ such that $\mu - \omega -> \mu'$. The **reachability set** of a net N with marking $\mu$, denoted $\mathbf{R}(N,\mu)$, is defined by $\mathbf{R}(N,\mu) = \{\mu' \mid \mu - \omega -> \mu'\}$. where $\omega$ is any sequence of events.

We call such nets *generalized* condition-event nets because these definitions constitute a significant generalization of the C/E net [7] in at least four ways. First, standard C/E nets allow no event to fire if any of its output places is already marked. Including this eventuality makes no difference to our results; storing a value into a memory location, for example, has the same effect whether or not that value was already stored there. Second, C/E nets exclude isolated events and places; again, we see no reason for this restriction. Third, C/E nets are in fact strongly connected. We must relax this restriction because we need two net components, or chains, representing the situational rules of a planning problem. Later, we shall see how the search for a plan is in a sense a process of connecting these two components. Fourth, and by far most important, C/E nets do not allow so-called **self loops** (a pair $(u,v) \in (P \cup T)$ is a self loop if $(u,v) \in E$ and $(v,u) \in E$); we have seen in the preceding section that allowing grammar rules with identical literals on the left- and right-hand-side has important representational power in many problem instances.

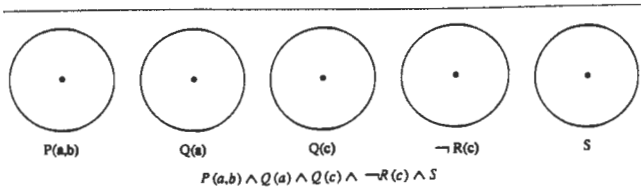



Figure 1. The Representation of a Situation

### 6.2. Net Representation of a Situation

Recall that a situation is a conjunction of literals in a logic. A net representation of a situation will consist of a set of places, each marked and each labeled with one of the conjuncts of the situation. An n-ary predicate, together with its arguments, is represented by an element $p \in P$, Each place is labeled by a literal (i.e., a predicate symbol together with its arguments). A place is marked by a token ( $\bullet$ ). For example, the truth of a formula $P(a) \wedge P(b)$ is represented by the presence of tokens in the place labeled by $P(a)$ and in the place labeled by $P(b)$. Figure 1 shows the most concise representation of the situation $P(a,b) \wedge Q(a) \wedge Q(c) \wedge \neg R(c) \wedge S$. (These places need not be the *only* members of a marking.)

### 6.3. Net Representation of an Schema

Associated with each operator are one input place for each occurrence of an $\alpha_i$ on the left-hand side of the rule and one output place for each $\beta_i$ on the right-hand side. For each $\alpha_i$ with arguments $\{u_1, \ldots, u_{a_i}\}$, the schema has an **input port** labeled by the n-tuple $<u_1, \ldots, u_{a_i}>$ and similarly has an **output port** for each $\beta_j$. The relationship of net event firing and operator execution is the following: the operator asynchronously receives a token from one of its input places at an input port to which that place is connected in the net. When all input ports for that operator have received tokens, the operator can execute. When the operator has concluded execution, it sends a token through each of its output ports to the place connected to that port.

Figure 2 shows a net representation of the operator schema

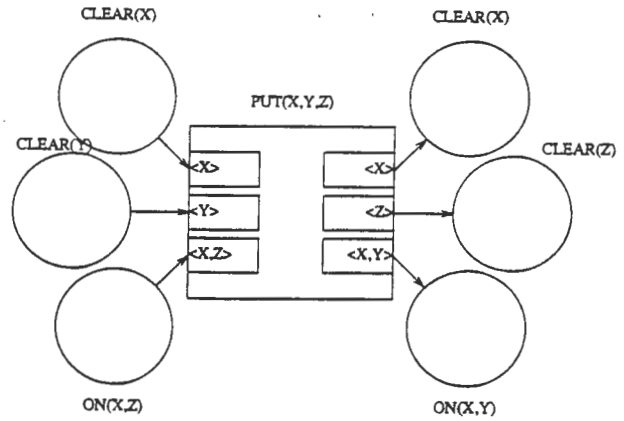



Figure 2. Net Representation of an Operator Schema

PUT(X,Y,Z): ON(X,Z), CLEAR(X), CLEAR(Y)
→ ON(X,Y), CLEAR(X), CLEAR(Z)

Just as a plan grammar $\Sigma$ consists of a set of operators, we can use the term **grammar net** of a grammar $\Sigma$ to denote the collection of graphical representations of the operators.

### 6.4. Net Representation of Plans

Although nets have been suggested as plan representations, this is the first work that attempts the actual automatic construction of such nets. Matters are complicated, as we shall soon see, because many nets have characteristics that render them unfit for plan representation. The nets we create, in which plans are embedded, themselves are generalized condition-event nets with a particular labeling on their places and events, and the algorithms used to create them must guard against introducing undesirable properties.

### 6.4.1. Operator Nets

Operators are used in plans to transform one situation into another. If a situation has certain characteristics, an operator will be enabled and can be employed at that point. If an operator is executed, the resulting situation will also have certain characteristics, perhaps different from the situation which preceded. The relation of situations to operators in plans motivates the definition of a particular labeling on a Condition-Event net.

**Definition:** Given a plan grammar $\Sigma$ and a logic, an **operator net** is a tuple (N,$\eta$,$\lambda$), where

- $N = (P,T,E,\mu)$ is a generalized C/E net,
- $\eta: P \rightarrow L$, the (ground) literals in a set of operators,
- $\lambda: T \rightarrow \Sigma$, the operators of a plan grammar,
- $(p,t) \in E$ iff $\eta(p)$ occurs on the left-hand side of $\lambda(t)$ in $\Sigma$,
- $(t,p) \in E$ iff $\eta(p)$ occurs on the right-hand side of $\lambda(t)$ in $\Sigma$, and
- $\mu \subseteq P$. □

In essence an operator net is a GC/E net in which the places and events have been appropriately labeled. The task of a planning algorithm is to construct an operator net (i.e., append elements to $P$ and $T$) in such a way that *any* execution of the net will constitute a plan for its input problem.

### 6.5. Persistence

Some GC/E nets represent plans and some do not. To help illustrate the difference, we look at an example of a net that does not represent a correct plan. Consider a world of identical blocks a, b, and c lying on a table t, and a robot manipulator arm which can lift one block at a time and place it on another. Assuming we have an operator schema

PUT(X,Y,Z) : CLEAR(X), CLEAR(Y), ON(X,Z)
→ CLEAR(X), CLEAR(Z), ON(X,Y)

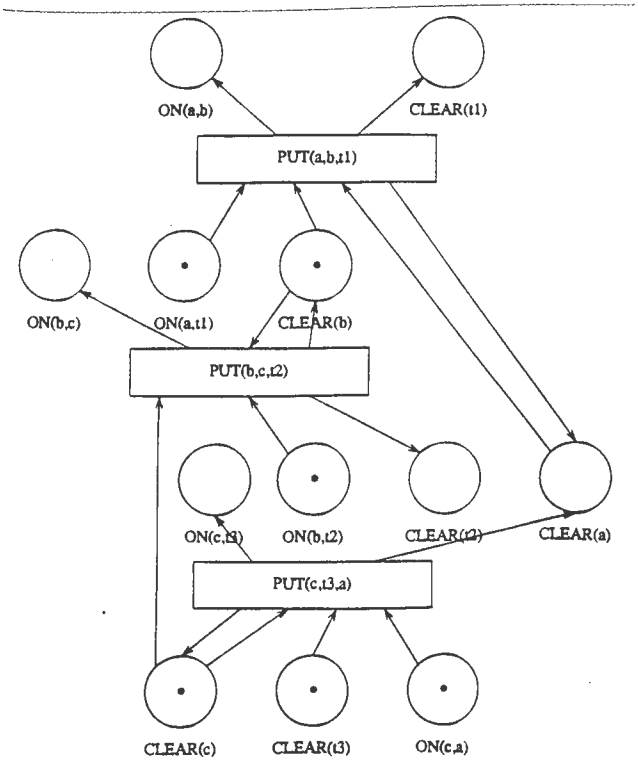



Figure 3. A Net That Is Not a Plan

the initial situation

ON(c,a), CLEAR(b), CLEAR(c) ON(b,t2), ON(a,t1), CLEAR(t3)

and a goal

ON(a,b), ON(b,c)

then Figure 3 shows a net representation which is not a plan, because when executed from the initial marking it will not necessarily halt in a marking satisfying the goal. CLEAR(c) is in the pre-sets of both PUT(c,t3,a) and PUT(b,c,t2) and in the post-set of PUT(c,t3,a) but not of PUT(b,c,t2). Furthermore, both events are enabled in the marking shown. If PUT(b,c,t2) fires before PUT(c,t3,a) the latter cannot fire; its enablement does not "persist" beyond the firing of the other event.

The idea of persistence has to do with the structure of the pre-sets and post-sets of events in a net. For an event $u$ to be persistent with respect to an event $t$, any places in the intersection of their pre-sets must occur in the post-set of $t$. The notion can best be defined with respect to sequences of events.

**Definition:** Let $N = (P,T,E,\mu)$ be a net. Let $\omega_1,\omega_2 \in T^*$ be sequences of events such that $\mu-\omega_1->$ and $\mu-\omega_2->$. $\omega_2$ is **persistent with respect to** $\omega_1$ if and only if $\bullet\omega_1 \cap \bullet\omega_2 \subseteq \omega_1\bullet$ Note that two enabled sequences whose pre-sets have empty intersection are persistent with respect to each other. A net $N = (P,T,E,\mu)$ is **persistent** if and only if for every $\omega_1 \neq \omega_2 \in T^*$, and every $\mu' \in \mathbb{R}(N,\mu)$ such that $\mu'-\omega_1->$ and $\mu'-\omega_2->$, $\omega_1$ is persistent with respect to $\omega_2$. □

We can illustrate persistence by returning to the above example. PUT(c,t3,a) is not persistent with respect to PUT(b,c,t2). Additionally, CLEAR(b) is in the pre-sets of both PUT(b,c,t2) and PUT(a,b,t1) and in the post-set of PUT(b,c,t2) but not of PUT(a,b,t1), so PUT(a,b,t1) is persistent with respect to PUT(b,c,t2) but not vice-versa.

## 7. Plan Synthesis

The plan synthesis process we employ grows an operator net, in which a strict partial ordering of operators is embedded, by appending components of the grammar net associated with $\Sigma$. This process continues until a net is constructed whose initially marked place is that labeled by $S_1$ and whose execution must result in marking a place $p_2$ labeled by the goal situation $S_2$.

We have seen what it means for a place in a net to be marked and what it means for an event to fire. Since in our view planning is concerned with simulation of events which are expected to take place in the future, we also need notions of "potential" marking and firing. We thus need to be clear about what it means for a place to be markable, i.e., potentially marked. For a place $p$ to be markable, certain properties must hold in the subnet of which $p$ is a root, that is, the net consisting of its pre-set, the pre-set of its pre-set, etc.

We say the **forward chain** of an operator net is that subset of nodes (places and events) to which the $S_1$ place $p_1$ is connected. The **backward chain** is that subset of nodes which are connected to the $S_2$ place $p_2$. The **forward-frontier (FF)** of a net is that subset of places of the net excluding $p_2$ having out-degree 0. The **backfrontier (BF)** is that subset of places excluding $p_1$ having in-degree 0. In the most general problem class, FF places can be found in both the forward and backward chains, but BF places occur only in the backward chain. See [1] for discussion of a classification scheme for planning problems and algorithms.

Returning to potential marking and firing, we can say an event $t$ of an operator net is **firable** if all input places of $t$ are markable, and a place $p$ of an operator net is **markable** if $p$ is

(i). in the forward chain and marked (in general true only of $p_1$), or

(ii). unmarked and any event in the pre-set of $p$ is firable, or

(iii). in the backward chain and unified with an eligible forward-frontier place (in either chain), provided no other place is unified with the same FF place.

Two places are unified if their labelings are identical under some consistent substitution. Unification is shown in our nets by dotted lines. Note that unification can be undone. "Eligibility" is

used to avoid introducing cycles, i.e., circular reasoning, into the net. See [1] for a detailed discussion of eligibility.

The planner seeks the creation of a solution for its problem, which can be represented by a **solution net**. A solution net is an operator net all of whose back-frontier places are markable. Equivalently, a solution net is one all of whose unmarked places have predecessor events in the net, or one in which the goal place $p_2$ is markable.

## 8. Final Place Merger and Mutual Persistence

Recall the two-robot collision zone problem. Let us examine a solution net (Figure 4) for this problem. (In the interest of brevity we will refer to labelings when strictly speaking we mean places and events in the net.) Note that the net in Figure 4 contains three places identically labeled 'safe(zone)', shown within the dashed rectangle. This particular net, which is deterministic, resulted from expanding 'at(r1,b)' before 'at(r2,d)'; other solution nets are possible. For example, if the algorithm had expanded 'at(r2,d)' first, the resulting net would have the same places and events but the edge relation would differ. Denote by $\omega_1$ the event sequence (enter_zone(zone,a,r1), leave_zone(zone,r1,b)). Denote by $\omega_2$ the event sequence (enter_zone(zone,c,r2), leave_zone(zone,r2,d)).

$\bullet\omega_1 =$ {path(zone,b),path(a,zone),at(r1,a),safe(zone)}, and $\omega_1\bullet =$ {at(r1,b),safe(zone)}. $\bullet\omega_2 =$ {path(zone,d),path(c,zone),at(r2,c),safe(zone)}, and $\omega_2\bullet =$ {at(r2,d), safe(zone)}. Thus $\bullet\omega_1 \cap \bullet\omega_2 =$ {safe(zone)}. Notice that $\omega_1$ and $\omega_2$ are mutually persistent, since $(\bullet\omega_1 \cap \bullet\omega_2) \subseteq \omega_1\bullet$ and $(\bullet\omega_1 \cap \bullet\omega_2) \subseteq \omega_2\bullet$. Thus it is possible to merge the three identically labeled places in the solution net as a final step in plan synthesis, because to do so will not destroy the mutual persistence of the the two enabled sequences. The merger can be accomplished by a straightforward isomorphic transformation from the places within the dashed rectangle of Figure 4 to the place within the dashed rectangle of Figure 5. The net of Figure 5, and hence the embedded plan consisting of its events, is nondeterministic in that either sequence $\omega_1$ or $\omega_2$ may occur before the other. It accommodates mutual exclusion on the collision zone because the place labeled "safe(zone)" is unmarked during execution of either sequence and thus one of the required tokens is unavailable to the other sequence. Note that the net contains two (intersecting) cycles. However, both cycles are markable and the events of each cycle are persistent with respect to the events of the other.

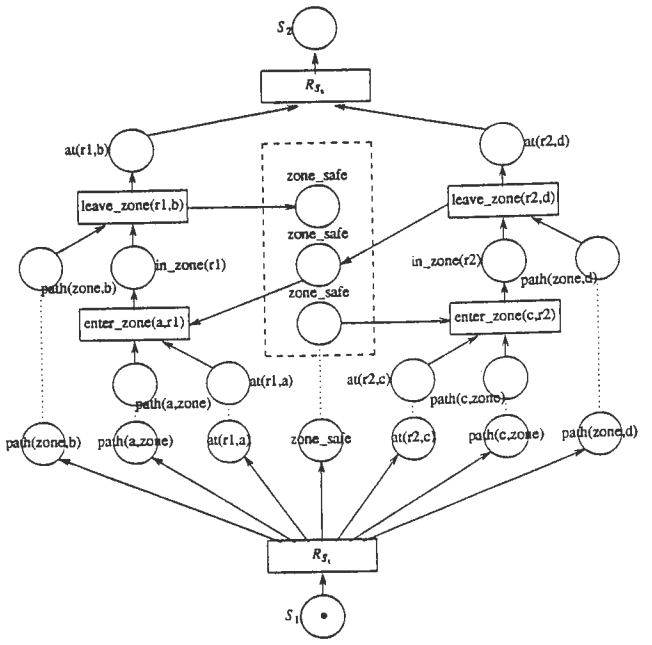



Figure 4. A Solution Net for a Spatial Planning Problem

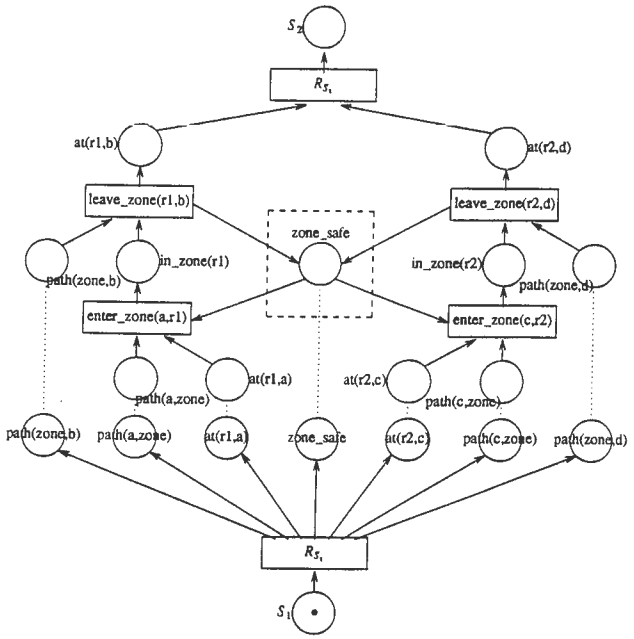



Figure 5. The Spatial Solution Net after Final Merger

The general merger rule thus is: we can merge places $p_i$ and $p_j$ to form place $p$ if and only if (i) their labelings ($\eta(p_i)$ and $\eta(p_j)$) are unifiable, and (ii) merger would not create any pair of sequences in the post-set of $p$ which are not mutually persistent.

## 9. Conclusions

In this paper we have been concerned with two aspects of planning: modeling plans and synthesizing plans. We have shown how to synthesize, not merely represent, plans using a restricted class of labeled nets called Generalized Condition/Event Nets. We also introduced the notion of plan grammar, and showed how such grammars allowed us to formulate problems requiring nondeterminism for their solution. We then discussed an important and necessary property of GC/E nets, persistence, which if present permits such a net to model a nondeterministic plan. Two event orderings are mutually persistent if the execution of one of the orderings does nothing to spoil the preconditions for the execution of the other. Finally, we discussed an algorithm to construct nets exhibiting the necessary persistence property, given only initial and goal situational specifications and a plan grammar. We concluded by showing how our algorithm would construct a net to solve our running example problem in a domain of simple multi-robot collision avoidance.

## 10. References


[1] D. Bahler, *Net-Based Plan Synthesis*, Ph.D. diss., Dept. of Comp. Sci., Univ. Virginia, Charlottesville, VA, 1987.

[2] D. Chapman, Planning for Conjunctive Goals, AI-Tech. Rep.-802, MIT AI Laboratory, Cambridge, MA, Nov. 1985.

[3] M. E. Drummond, Refining and Extending the Procedural Net, *Proc. IJCAI-85, 9*, (1985), 1010-1012.

[4] M. E. Drummond, A Representation of Action and Belief for Automatic Planning Systems, in *Reasoning About Actions and Plans*, M. P. Georgeff and A. L. Lansky (ed.), Morgan Kaufmann, Los Altos, CA, 1987.

[5] H. Gomaa, *Programming of Multiple Robot Systems*, Wang Inst. of Grad. Studies, Nov. 1986.

[6] C. A. Malcolm, Petri Nets for Representing Assembly Plans, DAI Working Paper No. 187, Dept. of AI, Univ. Edinburgh, Edinburgh, Scotland, March, 1986.

[7] W. Reisig, *Petri Nets: An Introduction*, Springer Verlag, New York, 1985.

[8] E. D. Sacerdoti, *A Structure for Plans and Behavior*, American Elsevier, New York, 1977.

# ITERATIVE CONSTRUCTS IN NON-LINEAR PRECEDENCE PLANNERS

Sam Steel
Dept Computer Science, University of Essex
Colchester CO4 3SQ, UK

ABSTRACT

Non-linear precedence planners (in the tradition of NOAH and NONLIN) need an iterative construct for achieving universally quantified goals and for doing certain sorts of recursion. A construct is suggested that can do special cases of both of these. It trades the generality of some other possible constructs for ease in deciding how to apply the proposed construct when it is applicable.

KEYWORDS: Planning, iteration.

## 1. Some Problems

Here are some problems that a non-linear precedence planner (in the tradition of NOAH (Sacerdoti 1975) or NONLIN (Tate 1976)) ought to be able to solve with the given action repertoire. The difficulty is that that the problems need repeated application of the operator, and there is no standard way of specifying repetition.

== chair painting
given: a set of chairs, some painted red, some not.
goal: all those chairs are painted red.
action: paint something red

== table clearing
given: a table with some some blocks on it.
goal: there are no blocks on it.
action: take a block off the table and put it on the floor.

== ladder climbing
given: a robot is at the bottom of a ladder.
goal: it is at the top of the ladder.
action: ascend one rung of the ladder

== name finding
given: a list of names in alphabetic order; the robot points at some name.
goal: the robot points at another, given, name, known to be in the list, and alphabetically later.
action: move finger along list from one name to another.

== tower demolishing
This problem is considered by (Manna & Waldinger 1986).
given: a block with a tower of other blocks above it of unknown but finite height.
goal: the bottom block is clear.
action: take a clear block off what it stands on.

## 2. A proposed solution.

One wants an iterative construct that is not just partially correct (if it terminates, its goal will be true) but totally correct (it will terminate and its goal will be true). That ruled out anything like a simple while loop of the form "WHILE goal not achieved DO action END" because as it stands there is no need for the loop to terminate.

In program verification, the standard way to prove that such a loop terminates is to divide the goal to be true after the loop into two parts, the variant and the invariant. The variant is a function whose value strictly decreases on each cycle, but which can't decrease below a known zero; so the loop terminates. The invariant is a property which if true before any cycle of the loop is also true after it. The goal must be equivalent to the invariant being true and the variant being at its zero. Each cycle must maintain the invariant and decrease the variant. Achieving that is the problem of planning what happens inside the construct. Then if one can achieve the invariant being true before the construct, and the variant being not less than its zero, the goal must be true after the construct.

The big problem about using this idea in program synthesis is deciding how to split the goal into the variant and the invariant. I shall propose specialized ways of doing this that loses generality but which (I hope) makes it much easier to produce totally correct loops when it applies.

Actions will be drawn as boxes, with their preconds on the left and their effects on the right. Protections (holding periods) will be drawn as double lines.

The main idea is this. Consider the whole iterative construct as a box that from the outside looks like an action with preconds and effects, perhaps involving quantifiers. During execution, it will go through N cycles, from 1 to N. Each cycle J will start in state J-1 and end in state J. So the whole loop will start in state 0 and end in state N. Furthermore, at each iteration, the cycle will be acting on an operand (which may be a tuple). That operand will will be called ^J. The set of objects ^1, ^2, ... ^N must be identified before execution of the iteration starts. The variant of such a loop is of course passage through the sequence 1, 2, ..., N.

Inside, however, the construct looks like a plan in its own right. That internal plan will be executed on each iteration, and it will be made just like any other plan. The question is, how do the initial state and the goals of that internal plan relate to the overall preconds and effects of

the whole iterative construct?

One can unroll an iteration into a sequence of repeated executions of the internal plan. These executions lie between the states 0,1,2,...,j-1,j,...,N. The validity of the iteration will be proved by an inductive argument, that some fact is true at all those states. That needs the situation calculus notation. I write "F @ S" for "F is true in state S" where F is a sentence not containing "@". It is a redundant notation; as usual, one can suppose all predications to have an extra argument, the state in which it holds. The induction is

```
                    ------- 1
                    I@(j-1)
                     :
      I@0           I@j
    1 ----------------------
      ∀j( 0=<j=<N, I@j )
```

The interest of that conclusion may be what it establishes about each of the states on the way to N; or just about its specialization to the final state N.

To complete that induction however, other assumptions may be needed:

* either about what is true in state 0: U@0
* or about what is true in every state: V@j
* or about conditionals of the form: W@(j-1) -> X@j

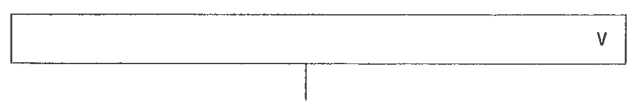
Such assumptions will assist an induction by fitting into the proof like this

```
                ------- 1
                I@(j-1)     V@j   W@(j-1) -> X@j
  U@0           ---------------------------------
  -----          :
  I@0           I@j
1 ----------------------
  ∀j( 0=<j=<N, I@j )
  ------------------
                I@N
```

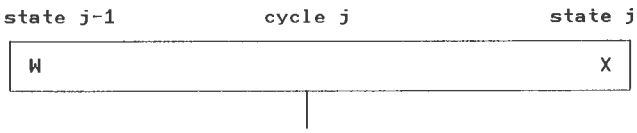
Each of the constructs below corresponds to some such induction. It has been proved in advance and canned as a construct. Nevertheless, the user still has to show that the assumptions that the proof makes are true. That is what filling in the construct does.

An external precond of the iteration corresponds to an assumption about what is true in state 0. It is drawn like this. (The situation names are implied by where the sentences are placed, and so can be omitted.)


A goal of the internal plan corresponds to the need to prove an assumption about what is true in every state.

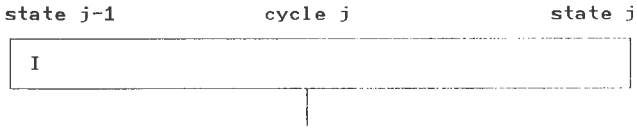



The making of a plan for an internal goal given facts about the internal initial state corresponds to filling in this constellation:

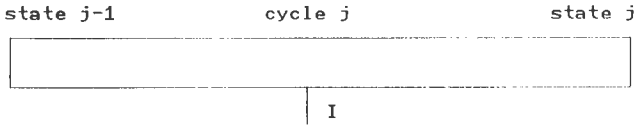



In the special case where W and X are the same, W is an invariant of the iteration.

And now for conclusions. Proving that I is true in all states corresponds to I being true in the initial state of the typical instance of the internal plan.




And the specialization of that corresponds to I being true in the final state of the iteration; that is, being an effect.

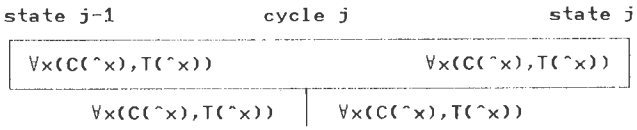



So different assumptions and conclusions in the iteration correspond to different constellations of facts in the iterative construct. Here are three useful constellations.

I: Leave things alone

If some set of objects C each start with a property T, and if all members of C that have property T at the start of a cycle retain it at the end of the cycle, then every member of C has property T at the end of the iteration.

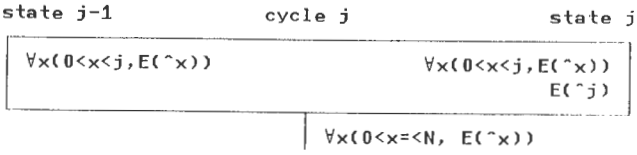
```
for all N, 0=<N,
If      ∀j(0<j=<N,∀x(C(^x),T(^x))@j-1
                        -> ∀x(C(^x),T(^x))@j )
and     ∀x(C(^x),T(^x))@0
then    ∀x(C(^x),T(^x))@N
```




II: Add an external effect

If at each cycle j all operands of past cycles retain property E, and if at the end of each cycle the operand of that cycle has property E, then after the iteration ends all operands have property E.

```
for all N, 0=<N,
If      ∀j(0<j=<N, ∀x(0<x<j,E(^x))@(j-1)
                        -> ∀x(0<x<j,E(^x))@j )
and     ∀j(0<j=<N,E(^j)@j)
then    ∀x(0<x=<N,E(^x))@N)
```




III: Add an internal fact (a fact that the body of the construct may rely on)

If at each cycle j all operands of future cycles retain property P, and before the iteration starts all operands have property P, then at the start of each cycle the operand of that cycle will have property P.

for all N, 0=<N,
If ∀j(0<j=<N,∀x(j<x=<N,P(^x))@(j-1)
-> ∀x(j<x=<N,P(^x))@j )
and ∀x(0<x=<N,P(^x))@0
then ∀j(0<j=<N, ∀x(j<x=<N,P(^x))@j)


Proofs: by induction on j.
From now on, "cycle j" will be left out of the diagrams.

## 3. Examples and refinements.

Here are some examples of how iteration can be used to achieve universally quantified goals. (It can also be used for "ladder climbing", described later.) It is convenient to divide quantification into two sorts; restricted and unrestricted.

### 3.1. Restricted quantification.

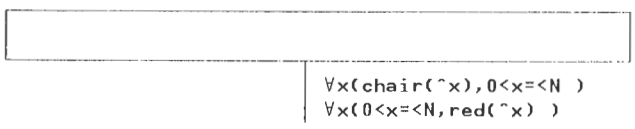
Consider painting the chairs red. The goal is

∀x( chair(^x), red(^x) ).

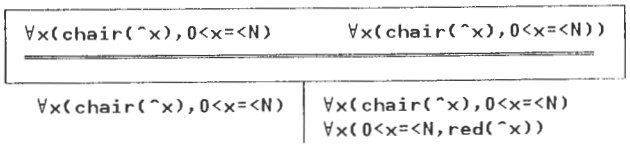
The heuristics that work on this are as important as the plan construction rules above. The goal should be split into

∀x( chair(^x), 0<x=<N )
∀x( 0<x=<N, red(^x) )

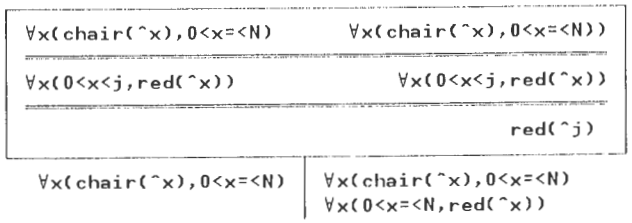
Then one should recognize that since there is no single action that achieves either of those goals, an iteration must be built.



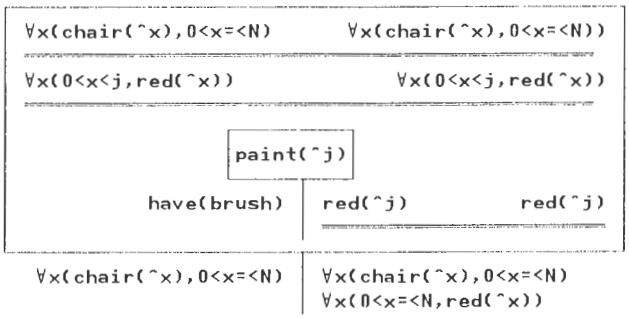

The first goal can be handled by the "leave things alone" constellation. The internal goal is reduced immediately. The protected fact amounts to a demand that the body of the iteration shouldn't create any more chairs than it started with. The external precond requires that all chairs are operands.



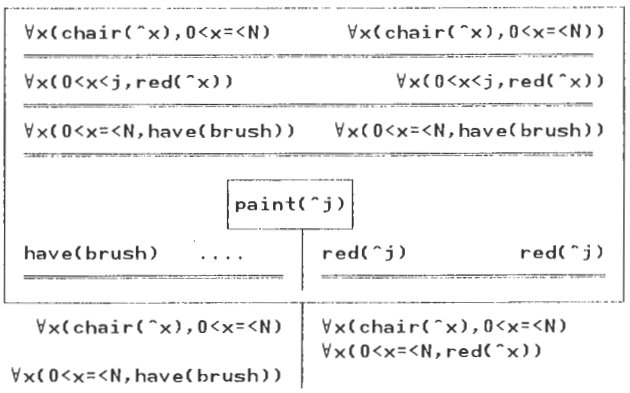

Actually getting the chairs painted needs the "add an external effect" constellation.




In order to paint the chair, one has to introduce an action into the internal plan.



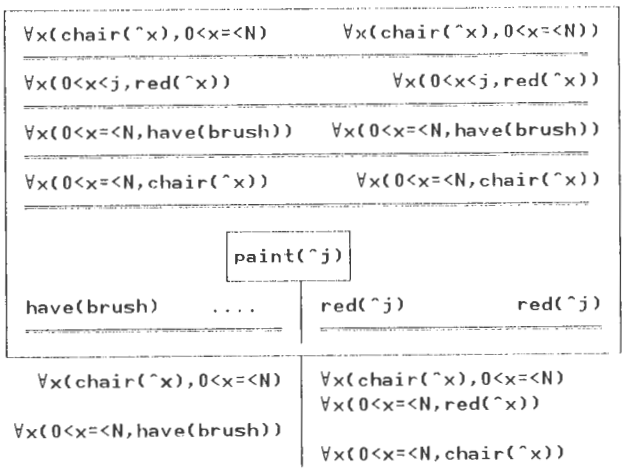

Now there is an internal subgoal. That can be achieved either by introducing another action into the internal plan; or, as here, by demanding it from the initial state of the cycle, using the "add an internal fact" constellation. The quantification is shown, even though it is vacuous. After this, the internal plan is complete.




So far there is no check that the things affected by the loop are chairs. Whatever they are, they get painted red. This is wasteful. A refinement avoids it. If one adds to the goal that requirement that only chairs are considered

∀x( chair(^x), 0<x=<N )
∀x( 0<x=<N, red(^x) )
∀x( 0<x=<N, chair(^x))

then it can be achieved via the "leave things alone" constellation, giving




There is also good independent motivation for the new invariant. Without it, there is no constraint that the the internal plan should not, so long as it paints the operand red, cause it to cease being a chair. If the iteration reduced all chairs to heaps of red splinters it would trivially have achieved

```
∀x( chair(^x), 0<x=<N )
∀x( 0<x=<N, red(^x) )
```

Now however any plan that destroyed a chair would violate the invariant just added and have to be dismissed.

### 3.2 Unrestricted quantification

Sometimes one wants everything to have some property. The goal of having a clear table is

```
∀x( -on(^x,table) )
```

If one sees that as

```
∀x( true(^x), -on(^x,table) )
```

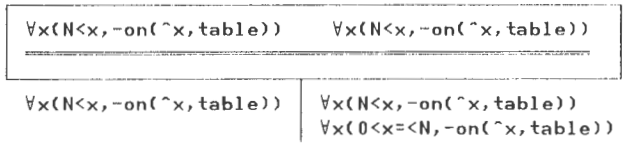
and treats it as before, one gets

```
∀x( true(^x), 0<x=<N )
∀x( 0<x=<N, -on(^x,table) )
```

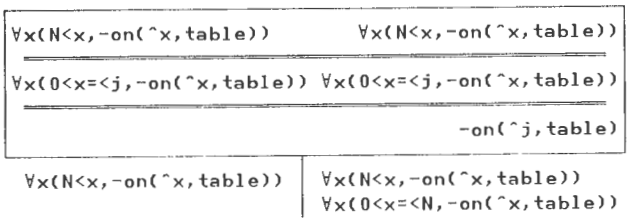
which ultimately involves one in treating every object in the universe as an operand; clearly a disaster. Instead one should see the goal as

```
∀x( N<x, -on(^x,table) )
∀x( 0<x=<N, -on(^x,table) )
```

Then one can build an iteration by first using the "leave things alone" constellation




then the "add an external effect" constellation.




It is then easy to make the required internal plan, to take a single thing off the table. The external precond is the requirement that everything that one doesn't consider during the iteration isn't on the table.

### 4. Using operands chained together by some relation.

The other examples need an extra piece of machinery. It is the assumption that there can be some relation, say <<, between the successive operands of the loop.

```
^0 << ^1 << ^2 << ... << ^N-1 << ^N
```

Note that a ^0 is assumed to exist.

There are two things that can be done with such a chain: "ladder climbing" and "tower building". This paper discusses only the first.

### 4.1 Ladder climbing

Here the idea is that the chain is found to exist already. The bottom item of the chain has some property. At each stage of the loop, that property is conferred on the next item in the chain, by an action that relies on the chain relation and on the property being true of the current item.

Formally, what happens is

```
-- one relies on        Q^0@0
-- one relies on
          ^0 << ^1 << ^2 << ... << ^f-1 << ^f
-- one achieves         Q^f@f
```

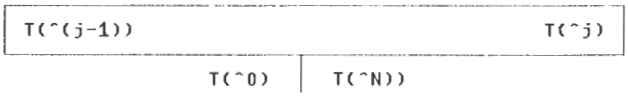
Each cycle of the loop must ensure that it
* achieves Q for the current operand
* preserves all the chain above the current operand
* preserves any extra preconds each cycle needs

Eg: the chain could be the rungs of a ladder;
the chain relation << would be
"(Rung1)is_below(Rung2)";
the property Q would be "I_am_at(Rung)".
The action would be "ascend(Rung1,Rung2)"
which would confer "I_am_at(Rung2)" on Rung2
because "(Rung1)is_below(Rung2)"
and "I_am_at(Rung1)".

To do this requires a new constellation.

IV: Transfer a property

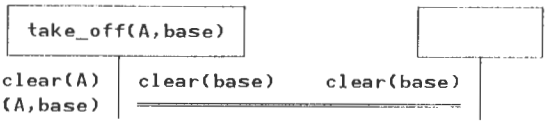
```
If      ∀j(0<j=<N, T(^(j-1))@j-1 -> T(^j)@j )
and     T(^0)@0
then    T(^N)@N
```



Ladder climbing is, I think, a very useful construct since it corresponds to what one wants to do when one reduces a goal to a very similar subgoal. It should be applied when near-loops occur. That struck me in the context of tower demolishing.

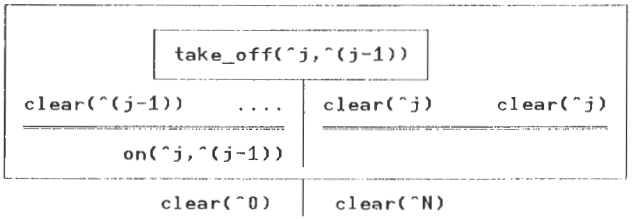
given: a block with a tower of other blocks above it of unknown but finite height.
goal: the bottom block is clear.
action: take a clear block off what it stands on.

The sequence of planning steps is: start with the goal "clear(base)". Once one sees that is not already true, one needs to find a suitable operator such as "take_off(A,B)" and use it.

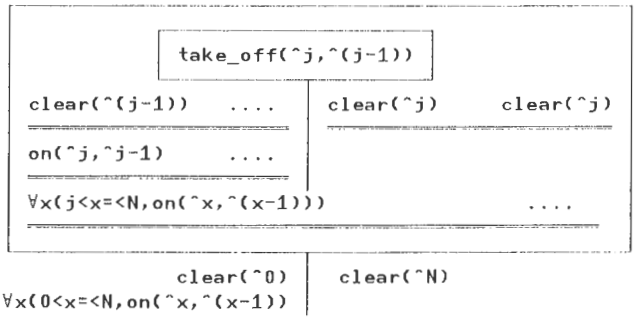



Now spot that the subgoal "clear(A)" is very like the original goal "clear(base)", and consider an iteration, using the "transfer a property" constellation.


The internal plan should involve the "take_off(A,base)" action already considered. Soon one reaches




Now use the "add an internal fact" constellation.




Rather oddly, the "base" of the chain of operands will be the top of the tower, and the "top" of the chain will be the base of the tower.

The preconds of the iteration amount to a demand that the tower being demolished have a top. One must rely on domain facts to establish that. That top will be a block ^0 which is itself clear and which is a certain number N of steps of "on" away from the base of the tower. If one uses the limited transitive closure, (u)<<*N(v) for any relation <<, defined as

$$(u)<<*0(v) \equiv u=v$$
$$(u)<<*(N+1)(v) \equiv \exists w((u)<<(w), (w)<<*N(v))$$

then that fact can be expressed as

$$\forall x(block(x), \exists N(0=<N, \exists y(block(y), clear(y) \ \& \ (y)on*N(x))))$$

Facts of that sort will be needed whenever ladder climbing is contemplated.

Name-finding is another instance of the same sort of task. The goal of being at the right name reduces to being at the name alphabetically just before the right name: and so on.

## 5. Execution

The execution of the construct involves doing the body of the loop on each successive operand. There is a problem about the knowledge requirement of the loop: when does one need to have vivid, operationally sufficient, descriptions of the operands? Some problems that seem to be of this sort are:

== When deciding to paint all chairs red, I must, before the loop, know of all the chairs that aren't red. But I don't have to decide until the beginning of each loop which particular chair to paint. So when do I grasp or decide the extension of the function ^ ?

== I also want to avoid looking again and again at chairs that I have painted to see if they are red or not, to tell whether one of them should be the next operand.

== Suppose one is dosing chicks. The dosed ones must not be put back with the others, since dosed and undosed chicks are identical. One would keep doing the same ones, and the loop need not terminate.

== What if there are several possible sets of objects that could be used as operands (eg different blocks that could be used to build a tower)? How do I choose which set to use as operands?

These are hard problems, but knowledge preconditions for action are so poorly understood in general that loops do not, I think, worsen the problem. Consider by contrast conditional constructs. For a conditional to be executable one must know at the moment it comes to be performed whether its condition is true or false. Now that is typically not made an explicit precond of the conditional. It is assumed that one will have complete information when it is needed. Here I am assuming complete information about the identity of the operands rather than about the truth of the condition; my assumption is as bad, but not worse. What one really needs is a full, general, account of how action relies on knowledge. But that is (alas) not offered here.

## 6. Other approaches

There seem to be four other main contenders for an iterative construct in this sort of planning.

### 6.1 Dijkstra's WHILE construct (Dijkstra 1975)

PRO: It is simple and well understood.
CON: If one also uses Dijkstra IFs, as one must, then why are there two constructs with conditionals in them?
CON: The main objection: termination is not provable unless one splits the goal into variant and invariant, which is hard and which was to be avoided.

### 6.2 Drummond's plan nets

Drummond (1985,1986) proposes a "Plan Net" representation of plans, which considers a Petri-net-like pebbling of a procedural net as a model of its execution. Iteration is seen as the recurrence of some state of pebbling.
PRO: This seems a natural representation for "whenever A do B" plans.
CON: It is not clear how to prove termination.
CON: It is a large step away from standard precedence planning.

### 6.3 Recursive actions

The idea here was to draw some complex plans in named boxes, which could contain recursive invocations of themselves; that is, they could contain boxes with the same name as themselves, and with the same preconds and effects, thought perhaps different arguments. This would allow arbitrary recursion.
PRO: Something like this is ineliminable if one wants to handle eg multiple recursion (eg quicksort).
PRO: The idea of the invariant arises naturally. It is those preconditions of the recursively invoked actions that are supplied, not by actions in the iterative construct, but from facts assumed true at the start of each cycle of the construct. They are "invariant" because no action in the construct may contradict them.
CON: Unless one wants to plan eg quicksort, the need for general recursion seems not to arise.
CON: The placing of the recursive call relative to the rest of the plan introduces an irrelevant choice between head/tail/middle recursion.
CON: The main problem is this: the natural thing to use as a variant is some argument to the recursive call. This should strictly decrease on each call. To achieve ∀x( Px∂n -> Qx∂n ), the natural argument is the set {x| -(Px -> Qx)} of things without the property. But this is a higher-order object. To say how sets changes across actions requires extra machinery. (Which, however, may be needed for eg actions that involve the destruction or creation of objects. But that is a big change.)

### 6.4 Manna & Waldinger's deductive synthesis

Manna and Waldinger (1986) apply their deductive synthesis method to the tower demolishing problem.
PRO: Their method is sound, elegant and general. It can generate arbitrary types of recursion. In the long run, I guess this is the way to go.
CON: As with all general methods, it is hard to control. The tower demolishing example in the paper is hand-generated. That is not a criticism of it; but they propose no heuristics to replace such hand guidance. In the short term, the construct proposed will be more manageable.

## 7. Limitations of this approach

There are some things this approach cannot do.

It is iterative rather than generally recursive, and I can't see that it can be extended to cases where one has to do a multiple recursion (where there is more than one invocation of the defined function within itself) such as quicksort.

The division of the task into operands sometimes looks rather forced. Consider cleaning a window. Each wipe cleans part of the window, and after all the wipes are done, the window is clean. The operands are presumably the parts of the window cleaned by each wipe. But no way of dividing the window is more natural than any other, so how how is any division either invented or selected? Similar problems can arise with any continuously divisible object.

Consider the table clearing strategy whereby at each cycle one puts on one block and takes off two. The table will be cleared, but that can't be shown by this approach. This approach numbers the objects to be operated on, which amounts to (doing recursion on)(having an induction hypothesis about) a sequence of strictly included sets. The sequence

```
^1, ^2, ..., ^N
```

corresponds to the sequence of sets of objects waiting to be operated on

```
{ ^1, ^2, ..., ^N }
{     ^2, ..., ^N }
{         ..., ^N }
...
{                 }
```

To handle the one-on-two-off problem, one needs to be able to recurse on a sequence of strictly smaller sets, whether or not one of them includes the next. That is more general.

## 8. Further work

The main difficulty is this: making the internal plan leads to new subgoals. Reducing those leads to a choice: shall I introduce further actions into the internal plan to reduce the new subgoals, or shall I add facts to the initial state of the internal plan? Of course, adding such facts is not free. Those facts will have to be achieved for each operand before the construct starts, and must be maintained across the internal plan. So the question is, How much of its own work should the construct do, and how much should it rely on having done to start with, only worrying about not undoing it?

Acknowledgements: I thank Jim Doran and Richard Young for their helpful comments on this paper.

References

Dijkstra EW, 1975: Guarded commands, nondeterminacy and formal derivation of programs: CACM 18:8 453-457

Drummond M, 1985: Refining and extending the procedural net: IJCAI 1985 1010:1012

Drummond M, 1986: A representation of action and belief for automatic planning systems: in: (Georgeff & Lansky 1986)

Georgeff M, Lansky A (eds) 1986: Reasoning about actions and plans: Morgan Kauffman

Manna Z, Waldinger R, 1986: A theory of plans: in: (Georgeff & Lansky 1986)

Sacerdoti ED, 1977: A structure for plans and behaviour: American Elsevier

Tate A, 1976: Project planning using a hierarchical non-linear planner: Tech report 25, Dept Artificial Intelligence, Edinburgh Univ

# Context Resolution: A Computational Mechanism for Intelligent Backtracking

*Jia-Huai You and Yigong Wang*

Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada, T6G 2H1

## Abstract

The nondeterministic nature of Prolog-based computations makes *intelligent backtracking* an important issue in logic programming. The problem is particularly crucial in Prolog programming for artificial intelligence where some of the well-known AI problems rely heavily on automatic backtracking, such as constraint-satisfaction problems. In this paper a resolution-based procedure, called *context resolution*, is presented, which, by associating terms with *contexts*, incorporates into resolution in a very natural way the information needed for intelligent backtracking. The method based on context resolution is more intelligent than those based on data dependency analysis, and easier to implement than those based on unification analysis. In addition, the context resolution method provides a flexible framework for intelligent backtracking in which the tradeoff between the overhead and the intelligence quotient can be easily adjusted. The resulting algorithm is proven to be correct; that is, backtrack points being pruned are only those that definitely lead to failure.

## 1. Introduction

Most Prolog interpreters use the depth-first search strategy with a naive backtracking scheme. That is, from the viewpoint of an SLD-tree, the leftmost branch is explored first, and when a failure node is encountered the prover backs up to the parent node to search alternative branches. Backtracking continues in this fashion until a success node is discovered or the search is trapped into an infinite derivation. Very often, some of the ancestor nodes are known to lead to failure. A smarter interpreter therefore should be capable of avoiding backtracking to these nodes; this is called intelligent backtracking. Intelligent backtracking can be realized by gathering all the information generated so far; based upon which the decision as to which ancestor node should be backed up to can be made. Ideally, we would like to prune all the nodes known to lead to failure and continue the search from the nearest ancestor node with an *uncertain* state, that is, whether it leads to failure or success or an infinite derivation is presently unknown.

The problem of backtracking in Prolog systems has caught much attention in the last few years, as it is not only a problem associated with sequential implementation of Prolog, but also an important issue to be addressed in most parallel execution schemes (See, for example, [CoKi85]). The problem is particularly crucial in Prolog programming for artificial intelligence where some of the well-known AI problems rely heavily on automatic backtracking, such as constraint-satisfaction problems. Previous work on intelligent backtracking can be divided into two categories based on the tools used in solving the problem.

Cox [Cox84] and Bruynooghe and Pereira [BrPe84] used deduction trees to analyze the possibility of unification of a subtree. Cox's algorithm was based on finding the maximal subtree such that unification is possible. Bruynooghe-Pereira's algorithm was based on finding the minimal subtree such that unification is impossible. Once the subtree has been found, the nodes that belong to an ununifiable subtree should be avoided. The method based on *modifying goals* by Pereira and Porto [PePo82] is another example of unification analysis. These analyses and the corresponding algorithms essentially laid out foundations for intelligent backtracking, by which the problem of backtracking can be adequately addressed conceptually. As indicated in [Cox84], however, the resulting algorithms can only be used in some restricted applications because the computation involved can be intensive. In addition, it is not very clear how an intelligent backtracking scheme can be effectively incorporated into resolution procedures.

In the second category, the main contributors are Conery-Kibler [CoKi85], Chang-Despain [ChDe85], and Lin-Kumar-Leung [LiKL86]. The tool used in their schemes is *data dependency graph*. Using data dependency analysis, a better backtrack point can be easily discovered from variable sharing information without performing a possibly intensive failure analysis. Another advantage of data dependency analysis is its direct support of synchronizing parallel evaluation of logic programs. However, data dependency analysis, which is based on computations of graph, can be expensive. In particular, when bindings introduced by unification include *nonground* terms, reconstruction of data dependency graph is often needed [LiKL86]. Another problem of the data dependency method is that it sometimes fails to locate the cause of failure *directly*, and is thus sometimes less intelligent than desired.

The scheme independently developed by Kumar and Lin [KuLi87] is also based on the data dependency relation. The restriction of ground bindings in the previous scheme [LiKL86] is eliminated by a technique called *tagging*. Their performance results indicate that this scheme entails the least

amount of overhead among several known schemes for a number of "typical" programs. However, some of the drawbacks of using the data dependency relation are still present (see Section 2).

The scheme to be proposed in this paper attempts to alleviate the problems encountered in the unification analysis and data dependency analysis approaches. The scheme treats ground and nonground terms uniformly. It is more intelligent than data dependency analysis and is easier to implement than unification analysis.

The main idea in our scheme is to associate each term with a context, a number indicating where the term was "introduced." Briefly, when unification takes place, a substituting term is assigned a context. When a failure is found to be associated with a symbol, the context of the symbol directly reveals where this failure can possibly be cured. In this way, a simple mechanism of maintaining contexts can be incorporated into resolution to keep track of the history of a computation for the purpose of intelligent backtracking.

Context resolution is a method aiming at formalizing and validating the above idea. Consider an SLD-derivation from some initial goal $G_0$ using the *leftmost* computation rule. Suppose the goal $G_i$: $\leftarrow p_1, ..., p_m$ is being processed. Next we will try to resolve $p_1$. When $p_1$ unifies with the head of a candidate clause, say $c \leftarrow c_1, ..., c_n$, we obtain a unifier $\{x_1/t_1, ..., x_l/t_l\}$. In order to record the terms that the current resolving step has affected, the variables $x_1, ..., x_l$ are replaced by $[t_1, \#i], ..., [t_l, \#i]$, respectively. The terms $[t_k, \#i]$ are called *context terms* where $\#i$ is the *context* pointing to the current goal $G_i$. The next goal is derived by replacing the selected literal with the body of the candidate clause and then by substituting these terms with their contexts into the resulting goal statement (all the symbols in the body are associated with $\#i$, too; see Subsection 3.3). In doing so, the context information is embedded into the newly derived goal and will be propagated to further derived goals.

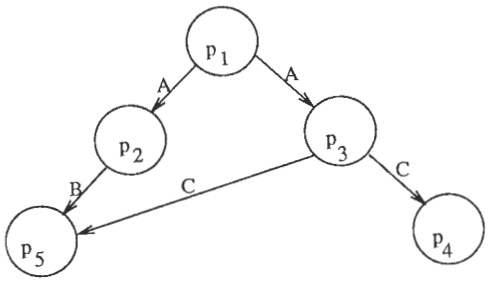
In the next section, we illustrate that data dependency analysis sometimes fails to locate the cause of a failure *directly*. Section 3 presents context resolution. We first extend terms to *context terms*, and ordinary unification to *context unification*, upon which context resolution is based. We discuss how different versions of context unification can yield different algorithms with different overhead and degrees of intelligence. A prover algorithm is then presented in Section 4, which makes use of context information to backtrack intelligently. Section 5 contains final remarks. An example of using the prover algorithm is included in Appendix I and the correctness proof of the algorithm is given in Appendix II. At the time of this writing, an implementation of the scheme on Waterloo Prolog [McCl87] has been attempted and partially completed. Preliminary performance results on a number of problems and their comparisons with two other schemes are provided in Appendix III.

## 2. Previous Work Based on Data Dependency Analysis

Data dependency analysis is based on the assumption that two literals are said to be dependent if they share at least one common variable. Chang-Despain's semi-intelligent backtracking scheme [ChDe85] employs *static* data dependency analysis, i.e., before the program is executed. Lin-Kumar-Leung's algorithm [LiKL86] implements a *dynamic* data dependency analysis. To carry out data dependency analysis, a data dependency graph is needed. As an example, suppose we have a clause

$$p(A, B, C) \leftarrow p_1(A), p_2(A, B), p_3(A, C), p_4(C), p_5(B, C).$$

The data dependency graph of this clause is shown as follows:




where $p_1$, in which the variable $A$ appears first, is called the generator of the variable $A$ and $p_2$ and $p_3$ the consumers of $A$; the variable $A$ is called a linking variable. When a literal fails, the execution backtracks to the closest literal on which it depends; for example, the failure of $p_5$ will cause the execution to back up to $p_3$.

One problem with this approach is that a data dependency relation changes frequently during execution; this includes disappearance of generators, generation of new generators and change of previously established dependency relation (A more detailed analysis can be found in [YoWa87]). For example, suppose we have

$$\leftarrow p_1(A, B),\ p_2(A, B),\ p_3(A, B).$$
$$p_1(f(X), Y) \leftarrow p_4(X, Y).$$

Resolving the goal yields

$$\leftarrow p_4(X, Y), p_2(f(X), Y), p_3(f(X), Y).$$

Now $p_4$ becomes the generator of $X$, and $p_2$ and $p_3$ depend on $p_4$ instead of $p_1$ initially.

Since the data dependency method keeps track of dependency at the literal level, not at the variable level, thus it may not directly locate the literal leading to unification failure. As an example, consider the following goal and program:

$$\leftarrow p_1(A),\ p_2(A, B),\ p_3(A, B).$$
$$p_1(a_1).$$
$$p_1(a_2).$$
$$p_2(a_1, b_1).$$
$$p_2(a_1, b_2).$$
$$p_2(a_2, b_1).$$
$$p_3(a_2, b_1).$$

Clearly, the first time backtracking occurs is at the goal $\leftarrow p_3(a_1, b_1)$ after resolving $p_1$ and $p_2$. According to the data

dependency relation, we should first backtrack to $p_2$ because of the linking variable $B$. We thus obtain $\leftarrow p_3(a_1, b_2)$ by selecting $p_2(a_1, b_2)$, which will fail again. It is observed that any change made on the second argument will not solve the conflict occurring at the first argument; the real cause of the failure of $\leftarrow p_3(a_1, b_1)$ is only due to the first argument of $p_3(a_1, b_1)$ in the goal, because it fails to match the first argument of $p_3(a_2, b_1)$ in the program. Thus, a more intelligent backtracking scheme should directly back to $p_1$ in this situation, based on the information that $a_1$ causes a conflict which was introduced at the time of resolving $p_1$. However, we must be careful with the situation. If we add the clause $p_3(a_1, b_2)$ to the program, we will have to consider $p_2$ as a backtrack point. This is because the subgoal $\leftarrow p_3(a_1, b_1)$ disagrees not only at the first argument with the clause $p_3(a_2, b_1)$, but also at the second argument with the clause $p_3(a_1, b_2)$. In this case, as the reader can see, if we directly backtrack to $p_1$ when $\leftarrow p_3(a_1, b_1)$ fails, the backtracking point ignored will lead to success. Therefore, if there is more than one backtrack point we must choose the latest one in order to guarantee that no solution will be missed.

The previous example is not really peculiar if we notice that the essential problem here is that the data dependency relation sometimes fails to provide accurate information about the cause of the failure directly. The situation can be serious if we are solving a goal such as

$$\leftarrow p_1(A, B), p_2(B, C), p_3(C, D), p_4(D, E), p_5(A, E, D).$$

Suppose the variable $A$ is bound to $a$ and every variable in the goal is bound in a sequence of derivations leading to $\leftarrow p_5(a, b, c)$. Further assume that there is only one candidate clause whose head is $p_5(b, b, c)$. Obviously, unless the binding for the variable $A$ is changed, the goal cannot be solved. Based on the data dependency relation, we should backtrack to $p_4$, on which $p_5$ depends more closely than others, and then to $p_3$, and so on; even though they have nothing to do with the cause of the failure. Even when they do contribute to the failure, for example, in the case that the head of the solely candidate clause is, say $p_5(b, c, a)$, it is easy to see that backtracking to $p_1$ is inevitable.

## 3. Context Resolution

In this section we are going to answer the question: without an explicit dependency relation how can we discover the correct backtrack point when a literal fails?

For convenience of technical representation, we will fix the computation rule to be used; we adopt the leftmost computation rule. Our method, however, does not preclude the use of other computation rules. In addition, like most Prolog systems, our interpreter is assumed not to perform "occur checks" in unification. Finally, we will not consider backtracking for the purpose of generating multiple answers. The solution to this problem is simple: backtrack to the nearest ancestor goal for which there are untried candidate clauses.

### 3.1 Context Terms

**Definition.** A *context* is a number $\#i$ which points to the goal $G_i$.

Let $\#i$ be a context. A *context term* is defined inductively as follows:

(i) An ordinary term is a context term;

(ii) If $f$ is an n-ary function and $t_1, ..., t_n$ are context terms, then $f(t_1, \ldots, t_n)$ is a context term;

(iii) If $t$ is a context term then $[t, \#i]$ is a context term.

Given a function symbol or a variable $s$ in a possibly nested context term of the form $[...[s, \#i]..., \#j]$, we say that $s$ is associated with the context $\#k$ if $\#k = \max(\#i, ..., \#j)$. We will denote by $context(s)$ the context associated with a given term $s$. In the case that $s$ is not associated with any context, i.e., $s$ is an ordinary term, we let $context(s) = \perp$.

We will denote by $functor(t)$ the leftmost function symbol in a nonvariable term $t$.

A context term is called a *context variable* if it is either a variable, or a variable associated with a context, or inductively, a context variable associated with a context. □

By definition, a context term is either an ordinary term, or constructed from terms, the brackets "[ ]" and contexts. Terms in the initial goal $G_0$ will be associated with a special context, denoted by $\#-1$, to reflect the fact that these terms are not introduced during the SLD-derivation. For convenience, this special context will often be omitted.

Note that a context term can be nested, for example, $[f([a, \#1]), \#2]$; we then say, by definition, $a$ is associated with $\#2$. This can happen, for example, in unifying the literal $p(X, f([a, \#1]))$ in $G_2$ (the current context is then $\#2$) with $p(Y, Y)$, by context unification to be described next. The implication of a nested context term is that a symbol therein has been "introduced" more than once, possibly along with some other symbols. The largest context happens to indicate the goal in which it was *last* introduced.

### 3.2 Context Unification

**Definition.** A substitution is now defined as a mapping from context variables to context terms, extended to an endomorphism of the set of context terms. □

**Definition.** Let *undo* be a function that maps context terms to ordinary terms such that, if $t$ is a context term then $undo(t)$ is the ordinary term obtained from $t$ with all brackets and contexts removed.

Two context terms $t$ and $s$ are said to be unifiable if there exists a substitution $\sigma$, such that $undo(t\sigma) = undo(s\sigma)$.

Two ordinary terms $t$ and $s$ (which may well be variables) are said to *disagree* if their leftmost symbols disagree. Two context terms $t$ and $s$ are said to *disagree* if $undo(t)$ and $undo(s)$ disagree; in the case that $t$ and $s$ are both nonvariable, $functor(t)$ and $functor(s)$ are called a pair of conflicting function symbols.

Let $S$ be a finite set of context terms. The *disagreement* set of $S$ is defined as follows. Locate the leftmost (function or variable) symbol position at which not all context terms in $S$ have the same

symbol and extract from each context term in $S$ the context subterms beginning at that symbol position. The set of all such context subterms with their contexts (if any) is the disagreement set. □

For example, $undo([f([a,\#2]),\#1]) = f(a)$; and the disagreement set of $f([a,\#2],[g(X),\#1])$ and $f([Z,\#4],[h(a),\#3])$ is $\{[a,\#2],[Z,\#4]\}$, where $g$ and $h$ are a pair of conflicting function symbols. Note that the number of terms in a disagreement set is at most the number of participating terms in $S$. Because of ignoring "occur checks," failure of unification can only be caused by the existence of a set of pairs of conflicting function symbols (constants are considered as 0-ary functions). Unification without "occur checks" have been addressed in Prolog II and related work (see, for example, [Colm84]).

The unification algorithm presented below is modified from the one given in [Lloy84]. In the algorithm, $S$ contains the two given terms to be unified; one is a context term which is the selected literal to be resolved, and the other is an ordinary term which is the head of a candidate clause. It is assumed the variables in these two terms are *standardized apart* so that the two given terms do not share common variables. In this special case, the disagreement set of any two context terms consists of exactly two terms. The algorithm will be invoked with the actual parameter for *context* being $i$ if the current goal is $G_i$. The composition of substitutions is denoted by $\bullet$.

**Context Unification Algorithm I:**

**unify-I**($S$, *context*);

1. Start with $k = 0$, $\sigma_0 = \varepsilon$.
2. If $\forall p, q \in (S)\sigma_k$, $undo(p) = undo(q)$, then stop, and return $\sigma'_k$ which is obtained by adding *#context* to each substitute in $\sigma_k$. That is, if $\sigma_k = \{x_1/t_1, ..., x_n/t_n\}$, then $\sigma'_k = \{x_1/[t_1, \#context], ..., x_n/[t_n, \#context]\}$. Otherwise, find the disagreement set $D_k$ of $(S)\sigma_k$.
3. If there exist $t$ and $s$ in $D_k$ and one of them, say $t$, is a context variable, then put
   $\sigma_{k+1} = \sigma_k \bullet \{t/s\}$;
   increment $k$ and go to 2.

   Otherwise, unification of $S$ fails. Let $t$ and $s$ be in $D_k$ and $f = functor(t)$ and $g = functor(s)$. If only one of $f$ and $g$ is associated with a context or both $f$ and $g$ are associated with the same context, then return that context. If $f$ and $g$ are associated with different contexts, then return the larger of the two. Otherwise, none of them is associated with a context; in this case, return the context associated with the context variable that caused the disagreement (see the third example below). □

**Examples.** Suppose the current goal is $G_5$.

$S = \{p([f([X,\#2]),\#4]),\ p(f(a))\}$; return $\sigma = \{[X,\#2]/[a,\#5]\}$

$S = \{p([f([b,\#2]),\#4]),\ p(f(a))\}$; return #4

$S = \{p([X,\#1],[X,\#1]),\ p(a,b)\}$; return #1 †

$S = \{p([[a,\#1],\#3],[[b,\#1],\#4]),\ p(X,X)\}$; return #4

In the last example, the context #4 is returned. If we had returned the context #3 which is associated with the first argument, we then might have lost the chance of possibly "correcting" the term $b$ to $a$ by using alternative clauses. This is why the more recent context of the two should be returned.

† The context associated with $X$, i.e., #1, is returned since $X$ caused the disagreement between $a$ and $b$.

Notice that for two non-unifiable terms **unify-I** finds the leftmost conflicting function symbols. Failure of unification, however, can result from more than one pair of conflicting function symbols. For example, with

$S = \{p([a,\#2],[b,\#3],[e,\#1],[r,\#4]),\ p(c,d,X,X)\}$,

the contexts #2, #3 and #4 indicate the goals where each of these conflicts may be resolved, respectively. Since unless all of the conflicts have been resolved, unification with the same atom (i.e., $p(c,d,X,X)$ in this example) cannot possibly succeed, the most effective way is to collect all contexts associated with conflicting function symbols and return the smallest (in this case, #2) to backup as far as possible. Based on this observation, we give a fuller version of context unification algorithm below. The algorithm works as follows. Whenever a pair of conflicting function symbols are discovered, instead of aborting the process and reporting a context, it saves the context and then proceeds to find the next pair of conflicting function symbols until all conflicts are collected. For each conflicting pair, the more recent context associated with the pair is obtained, and finally the smallest context among all so obtained contexts is returned.

**Context Unification Algorithm II:**

**unify-II**($S$, *context*);

1. Start with $S_0 = S$, $k = 0$, $\sigma_0 = \varepsilon$, $C = \emptyset$.
2. Find the disagreement set $D_k$ of $(S_k)\sigma_k$.
3. If there exist $t$ and $s$ in $D_k$, then do the following:
   (i) if one of $t$ and $s$, say $t$, is a context variable, then put
   $\sigma_{k+1} = \sigma_k \bullet \{t/s\}$;
   $C_{k+1} = C_k$;
   $S_{k+1} = S_k$;
   increment $k$ and go to 2;
   (ii) if neither $t$ nor $s$ is a context variable, then put
   $\sigma_{k+1} = \sigma_k$;
   $C_{k+1} = C_k \cup \{<t,s>\}$;
   $S_{k+1} = S_k$ with $t$ and $s$ being replaced by a new variable;†
   increment $k$ and go to 2.
4. Otherwise, $D_k$ is empty.
   (i) If $C_k$ is also empty, unification of $S$ succeeds. Let $\sigma_k = \{x_1/t_1, ..., x_n/t_n\}$. Return unifier $\{x_1/[t_1, \#context], ..., x_n/[t_n, \#context]\}$.
   (ii) If $C_k$ is nonempty, unification fails. For each pair $<t,s>$ in $C_k$, let
   $\#n = context(functor(t))$
   $\#m = context(functor(s))$,
   and obtain a context according to
   (a) if $\#n \neq \perp$ & $\#m \neq \perp$, get $max(\#n, \#m)$;
   (b) if $\#n \neq \perp$ & $\#m = \perp$, get $\#n$;
   (c) if $\#n = \perp$ & $\#m \neq \perp$, get $\#m$;
   (d) if $\#n = \perp$ & $\#m = \perp$, get the context associated with the context variable that caused the disagreement.

   Let the set of all these contexts so obtained be $C$. Then return $minimum(C)$. □

† The purpose here is simply to hide this conflict in order to find the next conflict.

The algorithm **unify-II** is more costly since it requires an extra amount of work as compared to **unify-I**, where failure is reported immediately after first conflict is discovered. If the returned context is not the smallest, as it may happen in **unify-I**, the conflicts resulted from some earlier context will be rediscovered as the cause of failure. Here we see a tradeoff between the overhead and the degree of intelligence.

Further extension of context unification will be to collect context information whenever a variable is bound, even in the case that unification succeeds. Since this discussion involves maintenance of $B$-list, we will leave it to the next section.

### 3.3 Context Resolution

Context Resolution is like standard resolution, except that it uses context unification and the context information is carried over along the derivation. The procedure **resolve** described below gives the actions to be performed in a single resolving step. Note that we have chosen the leftmost computation rule.

**resolve**($G_i$, $\sigma$);

where $G_i$ is of the form $\leftarrow l_1, \ldots, l_k$ and $\sigma$ is a unifier obtained by Context Unification for $l_1$ and $p$, $p$ being the head of a clause $p \leftarrow q_1, \ldots, q_m$. The next goal derived is of the form

$$\leftarrow \{q_1, \ldots, q_m, l_2, \ldots, l_k\}\sigma.$$

Since the symbols in the clause $p \leftarrow q_1, \ldots, q_m$ are also *introduced* in the current resolving step, they should be associated with the current context $\#i$. Let $q'_j$, $1 \le j \le m$, denote the literal obtained from $(q_j)\sigma$ by associating symbols in $q_j$ with the current context. Then, the next goal derived is:

$G_{i+1}$: $\leftarrow q'_1, \ldots, q'_m, l_2\sigma, \ldots, l_k\sigma$ □

**Example.** The goal $G_3$ below is derived from $G_2$ by resolving with $p_1(f(Y), b) \leftarrow p_3(Y, c)$:

$G_2$: $\leftarrow p_1(f([a, \#1]), X),\ p_2(X).$

↓

$G_3$: $\leftarrow p_3([[a, \#1], \#2], [c, \#2]),\ p_2([b, \#2]).$

## 4. A Prover Algorithm

We now present a prover algorithm that makes use of context information for intelligent backtracking. We have chosen to present the algorithm as simple as possible, similar to the prover in [Nils84], in order to clearly convey the main ideas and the mechanism. The algorithm is presented in Figure 1 and works as follows. A failed literal in a goal statement either failed *directly*, meaning it failed to unify with the head of *any* candidate clause, or failed *indirectly*, meaning it succeeded *at least* once but finally failed. Given a goal $G_i$ to be solved, we choose the leftmost literal to resolve. If there still exists an untried candidate clause, we then try to unify the head of the clause with the chosen literal. If unification succeeds, we go resolve the derived goal. Otherwise, unification fails. In this case we save the returned context in $B$-list, and try next candidate clause. At this moment, we do not know yet whether this will be a direct failure. By the time that all candidate clauses have been exhausted, if it turns out to be a direct failure, then all the contexts that are the sources responsible for each failure of unification have already been saved in $B$-list. We then backtrack to the most recent context in $B$-list. Otherwise, it is an indirect failure; in this case, all the contexts contained in the current literal are put into $B$-list (by `get_all_contexts(current_literal))` in Figure 1) and the most recent context in $B$-list is selected for backtracking. The selected backtrack point is then deleted from $B$-list (by `delete_from(backtrack_point, B-list)` in Figure 1). The execution continues at the restored goal pointed to by the selected backtrack point. A simple example in actions using this algorithm is included in Appendix I. The correctness proof of the algorithm can be found in Appendix II.

**A Prover Algorithm:**

```
B-list := {#-1};
          /*the special context #-1 is in initially*/
prover(G_i, program)
 {
 if (G_i = NIL) return SUCCESS;
          /*an answer has been found*/
 current_literal := leftmost(G_i);
          /*use the leftmost computation rule*/
try:
 clause := candidate_clause(current_literal,program);
          /*get an untried candidate clause*/
 if (clause ≠ NIL)
     {    /*there is still some untried candidate clause*/
     substitution := unify({current_literal,
                            head(clause)}, i);
     if (substitution is a context)
          {    /*unification failed*/
          B-list := union({substitution}, B-list);
               /*save a potential backtrack point*/
          goto try
          };
     if (substitution is not a context)
          {    /*unification succeeded*/
          G_{i+1} := resolve(G_i, substitution);
          return prover(G_{i+1}, program)
          }
     };
 if (clause = NIL)
 {         /*no more candidate clause*/
 if (failed directly)
     backtrack_point := max(B-list);
 if (failed indirectly)
     {
     B-list := union(B-list,
                     get_all_contexts(current_literal));
          /*add all contexts to B-list*/
     backtrack_point := max(B-list);
     };

 if (backtrack_point = #-1) return FAILURE;
          /*failure due to the initial goal*/
 B-list := delete_from(backtrack_point, B-list);
 B-subgoal := reset_subgoal(backtrack_point);
     /*restore the goal pointed to by backtrack_point*/
 return prover(B-subgoal, program)
 }
}
```

**Figure 1:** A prover algorithm.

The $B$-list used in the algorithm is global. In [LiKL86, KuLi87], one $B$-list is attached to each clause. By using local $B$-lists, some of the irrelevant information can be further

eliminated. This approach can be adopted in our algorithm without any difficulty. See [KuLi87] for details of manipulating local B-lists.

The reason to include all contexts when a goal fails *indirectly* is due to the fact that context unification did not return context information when unification succeeded. This may introduce redundant backtrack points since not all contexts are relevant to the failure. Actually, when unification succeeds relevant contexts are those that are contained in the substitutes of the unifier. Thus a remedy to this problem would be to return, in addition to a unifier, the contexts contained in the substitutes of the unifier. This is another tradeoff between the overhead and the degree of intelligence. Adding this process into unification would be beneficial only in the case that solving a goal involves a large mount of indirect failures. Incidentally, if we add this process into the context unification algorithm **unify-II**, the resulting algorithm becomes very close to the schemes based on unification analysis, such as Pereira and Porto's selective backtracking method [PePo82], which is more intelligent than ours. Our scheme using either **unify-I** or **unify-II** is therefore essentially a simplified version of theirs.

## 5. Final Remarks

By comparing the existing approaches and analyzing the problems therein, we have proposed an intelligent backtracking scheme, based on the concept of context resolution. The advantage of our scheme is that it naturally embeds the context information into resolution; as a result, it can locate the cause of failure directly. The overhead in the resulting algorithm includes maintenance of $B$-list and an association process built into unification. Each activation record on the stack will consume a little more space to keep several extra pointers. Backtracking to the goal pointed to by a context is easy as long as we have a good scheme for keeping the traces of execution. In addition, we have seen how tradeoffs between the overhead and the degree of intelligence can be made and analyzed around the notion of context unification.

An implementation of the scheme proposed here has been attempted for Waterloo Prolog [McCl87]. Intelligent backtracking is considered an add-on feature on top of the system, providing the user with several options that differ in the entailed overhead and the degree of intelligence. In the case where the computation is determinate or involves little backtracking, the user can simply ignore this add-on feature as a whole or on a per-clause basis. At the time of this writing, the implementation is not yet fully completed. We can only provide in Appendix III some very preliminary performance statistics and a comparison with two other schemes.

The mechanism of intelligent backtracking will later be combined with a mechanism of computing with constraints (in particular, with equations [YoSu86]), another feature we plan to implement on top of Waterloo Prolog. It is our hope that constraint generation and propagation can significantly reduce the need of backtracking, resulting in an execution mechanism similar to that of means-ends analysis [Rich83].

**Acknowledgments.**

This work is supported in part under NSERC operating grant number A9225. We would like to thank Brian Wong for implementing the scheme and providing the performance results. Referees' comments are useful and have helped improve the presentation of this paper.

## REFERENCES

**[BrPe84]** Bruynooghe, M. and L.M. Pereira, "Deduction revision by intelligent backtracking," in *Implementations of Prolog*, pp. 194-215, Ellis Horwood Limited, 1984.

**[ChDe85]** Chang, J.-H. and A.M. Despain, "Semi-intelligent backtracking of Prolog based on a static data dependency analysis," *Proc. of IEEE Symposium on Logic Programming*, pp. 10-21, Boston, Mass., July, 1985.

**[Colm84]** Colmerauer, A., "Equations and inequations on finite and infinite tree," in *Proc. of International Conference on Fifth Generation Computer Systems*, pp. 85-102, Tokyo, Japan, November, 1984.

**[CoKi85]** Conery, J.S. and D.F. Kibler, "AND parallelism and nondeterminism in logic programs," in *New Generation Computing*, Vol. 3(1985), pp. 43-70.

**[Cox84]** Cox, P.T., "Finding backtrack points for intelligent backtracking," in *Implementations of Prolog*, pp. 216-233, Ellis Horwood Limited, 1984.

**[KuLi87]** Kumar, V and Y. Lin, "An intelligent backtracking scheme for Prolog," *Proc. of IEEE Symposium on Logic Programming*, San Francisco, September, 1987.

**[LiKL86]** Lin, Y., V. Kumar and C. Leung, "An intelligent backtracking algorithm for parallel execution of logic programs," *Proc. of IEEE Symposium on Logic Programming*, pp. 55-68, SLC, Utah, 1986.

**[Lloy84]** Lloyd, J.W., *Foundations of Logic Programming*, Springer-Verlag, New York, 1984.

**[McCl87]** McClurkin, D.J., "WUP Version 3.0 User's Manual," Logic Programming and Artificial Intelligence Group, Department of Computer Science, University of Waterloo. August, 1987.

**[Nils84]** Nilsson, M., "The world's shortest Prolog interpreter?," in *Implementations of Prolog*, pp. 87-92, Ellis Horwood Limited, 1984.

**[PePo82]** Pereira, L.M. and A. Porto, "Selective backtracking," in *Logic programming*, eds. K.L. Clark and S-A. Tarnlund, pp. 107-114, Academic Press, 1982.

**[Rich83]** Rich, Elaine, *Artificial Intelligence*, McGRAW-HILL Book Company, New York, 1983.

**[YoSu86]** You, J.-H. and P.A. Subrahmanyam, ''Equational logic programming: an extension to equational programming,'' in *Proc. of 13th POPL*, pp. 209-218, St. Petersburg, Florida, January, 1986.

**[YoWa87]** You, J.-H. and Y. Wang, ''Intelligent backtracking by finding the cause of failure,'' Dept. of Computing Science, University of Alberta, June, 1987.

## Appendix I: An Example

*Program:*

$p_0(A, B, C) \leftarrow p_1(A), p_2(A, B), p_3(A, B).$
$p_1(f(D)) \leftarrow p_4(D).$
$p_4(a).$
$p_4(b).$
$p_2(f(a), b).$
$p_2(f(b), b).$
$p_3(f(b), b).$

*Goal:*

$\leftarrow p_0(X, Y, Z).$

*SLD-derivation:*

#0: $\leftarrow p_0(X, Y, Z).$
$\sigma_1 = \{X/[A, \#0],\ Y/[B, \#0],\ Z/[C, \#0]\}$
$B$-list = $\{\#-1\}$†

#1: $\leftarrow p_1([A, \#0]),\ p_2([A, \#0], [B, \#0]),\ p_3([A, \#0], [B, \#0]).$
$\sigma_2 = \{[A, \#0]/[f(D), \#1]\}$
$B$-list = $\{\#-1\}$

#2: $\leftarrow p_4([D, \#1]),\ p_2([f([D, \#1]), \#1], [B, \#0]),$
$p_3([f([D, \#1]), \#1], [B, \#0]).$
$\sigma_3 = \{[D, \#1]/[a, \#2]\}$
$B$-list = $\{\#-1\}$

#3: $\leftarrow p_2([f([a, \#2]), \#1], [B, \#0]),$
$p_3([f([a, \#2]), \#1], [B, \#0]).$
$\sigma_4 = \{[B, \#0]/[b, \#3]\}$
$B$-list = $\{\#-1\}$

#4: $\leftarrow p_3([f([a, \#2]), \#1], [b, \#3]).$

Fails because of the disagreement between $a$ and $b$. This is a direct failure. The conflict between $a$ and $b$ caused context #2 to be added to $B$-list. So, $B$-list = $\{\#2, \#-1\}$ and max($\{\#2, \#-1\}$) = #2; the execution backtracks to #2.

#2: $\leftarrow p_4([D, \#1]),\ p_2([f([D, \#1]), \#1], [B, \#0]),$
$p_3([f([D, \#1]), \#1], [B, \#0]).$
$\sigma_2 = \{[D, \#1]/[b, \#2]\}$
$B$-list = $\{\#-1\}$
/*#2 is deleted after backtracking*/

#3: $\leftarrow p_2([f([b, \#2]), \#1], [B, \#0]),$
$p_3([f([b, \#2]), \#1], [B, \#0]).$
$\sigma_3 = \{[B, \#0]/[b, \#3]\}$
$B$-list = $\{\#-1\}$

#4: $\leftarrow p_3([f([b, \#2]), \#1], [b, \#3]).$

Succeeds after one more step.

## Appendix II: Correctness Proof

We will first determine the relationship between a failed literal and the goal it backtracks to. First a definition.

**Definition.** Two literals $p$ and $q$ are said to be at the same *level* if they appeared simultaneously in a derived goal during SLD-derivation.

A literal $p$ is the *ancestor* of a literal $q$ if $q$ is an instance of a literal occurring in the body of the clause whose head unifies with $p$. □

In the following derivation, $p_1$, $p_2$ and $p_3$ are at the same level, and so are $p_3$ and $q_1$. But $p_2$ and $q_1$ are at different levels, and so are $p_1$ and $q_1$. In addition, $p_2$ is the ancestor of $q_1$.

$\leftarrow p_1, p_2, p_3.$
$\downarrow$
$\leftarrow p_2, p_3.$
$\downarrow$
$\leftarrow q_1, p_3.$

Notice that a literal and its ancestor must be at different levels. In what follows, context unification refers to **unify-I** described in Subsection 3.2; however, the results can be easily extended to **unify-II**.

**Lemma 1.** Let $p$ be the leftmost literal in the current goal that fails. If $q$ is the leftmost literal at the goal pointed to by $\#k$, $\#k$ = max($B$-list), then $q$ is either at the same level as $p$ or the ancestor of $p$. □

**Proof:**

Let the symbol $f$ be associated with $\#k$. Assume that $q$ and $p$ are at different levels (otherwise $p$ and $q$ are at the same level). Show $q$ is the ancestor of $p$. Let $q'$ be the ancestor of $p$. We will then encounter the following two goals:

$\#i$: $\leftarrow q', s, ...$
$\#i+1$: $\leftarrow ..., p, ..., s, ...$

where $q'$ unifies with the head of $q'' \leftarrow ..., p', ...$ and $p$ is an instance of $p'$. From the condition that $\#k$ is the largest in $B$-list and $f$ is introduced by resolving $q$ (which is at a different level from $p$), $f$ must be introduced at the point $\#i$ or before. Furthermore, since $p$ contains $f$, either $p'$ itself contains $f$ or a variable in $p'$ is bound to a context term containing $f$ by unifying $q'$ with $q''$; in the first case, according to context resolution, $f$ is associated with $\#i$. In the second case, $f$ is also associated with $\#i$ by context unification, because $\#i$ is the largest at the point $\#i$. Therefore, $\#i$ must be equal to $\#k$ and $q$ must be $q'$. □

**Theorem 2.** Let $p$ be the leftmost literal in the current goal that failed. If $\#k$ = max($B$-list) is tried next, no solution can be missed; i.e., all the backtrack points between $\#k$ and currently failed goal will definitely lead to failure. □

**Proof:** By Lemma 1, the leftmost literal of the goal pointed to by $\#k$ is either at the same level as $p$ or the ancestor of $p$. Therefore, the leftmost literal in any derived goal between $\#k$ and the current goal must be at the same level as $p$:

$\#k+1$: $\leftarrow q, ..., p_{k+1}, ....$
.....
$\#j$: $\leftarrow ..., (p_{j-1})\theta_{j-1}, ....$
.....
$\#l$: $\leftarrow (p_{l-1})\theta_{l-1}, ....$

where for any $j$, $k+1 \le j \le l-1$, $\theta_j$ is the unifier associated with the resolving step at the $j$-th goal, $(p_j)\theta_j = p_{j+1}$, and $(p_{l-1})\theta_{l-1} = p$. We show that no refutation can be found by backtracking to any of these goals.

If new bindings produced by re-resolving $q$ and all subsequent literals before $p$ do not change $p$ at all, obviously $p$ will fail again. If they do, there are two cases depending on whether $p$ failed directly or indirectly.

**case a:** $p$ has not previously succeeded (i.e., failed directly).

In this case, the head $h_l$ of each candidate clause has failed to unify with $p$. Note that since $p = (p_{k+1})\theta_{k+1} \bullet \cdots \bullet \theta_{l-1}$, $h_l$ is in fact unifying with $(p_{k+1})\theta_{k+1} \bullet \cdots \bullet \theta_{l-1}$. Let $f$ and $g$ be a pair of conflicting function symbols in unifying $p$ and $h_l$, and $\#n_l$ be the context returned accordingly. Note that $\#n_l \le \#k$. Note also that since "occur checks" have been

† Recall (Section 4) that the context $\#-1$ is initially in $B$-list and new contexts are added to $B$-list when a chosen literal fails either directly or indirectly.

ignored, failure of unification can only be caused by conflicting function symbols. We first show that neither $f$ nor $g$ was introduced by the substitution $\theta_{k+1} \bullet \cdots \bullet \theta_{l-1}$; hence they must have already existed in $p_{k+1}$ or been drawn from $h_l$.

By **unify-I**, if $f$ is associated with a context but $g$ is not, then $g$ is a function symbol originated from $h_l$ and $f$ is already in $p_{k+1}$ because $\#n_l \leq \#k$. If both $f$ and $g$ are associated with some context respectively, then $f$ and $g$ are already in $p_{k+1}$ because $\#n_l$ is the larger of the two contexts but no larger than $\#k$. If the context returned is associated with a context variable, then this context variable must occur (at least twice) in $p_{k+1}$ and $f$ and $g$ are symbols in $h_l$. (Note that the context variable must be changed from multiple occurrences of the same variable to distinct variables or other terms in order to unify with $h_l$.) It follows that for any substitution $\eta$, $(p_{k+1})\eta$ cannot be unifiable with $h_l$.

Since the above is true for each candidate clause, it follows that new bindings generated by backtracking to any goal after $\#k$ will not be able to resolve $p$ and thus definitely lead to failure.

**case b:** $p$ has previously succeeded at least once (i.e., failed indirectly).

In this case, all the contexts in $p$ are included in $B$-list. If new bindings generated by backtracking to any point after $\#k$, say $\#i$, affect $p$, then $\#i$ must be in $p$ already and therefore in $B$-list at the point when $p$ failed; this contradicts the assumption that $\#k$ is the largest in $B$-list.

We therefore conclude that backtracking to any goal between $\#k$ and the current one having $p$ as the leftmost literal will fail to solve $p$. □

## Appendix III: Performance Results

Our intelligent backtracking scheme is tested on a variety of problems. The same set of problems is used in the evaluation of the backtracking schemes of [BrPe84] and [KuLi87].

The run time of a program is obtained by measuring the amount of time spent inside the interpreter alone. This is done by calling the UNIX† *times* command which returns the CPU time used while executing instructions in the user space of the program. *times* is called just before entering the interpreter and it is called again just after leaving it. Thus the amount of time spent in the interpreter can be determined.

The results are obtained by running the programs on an idle Sun-3/50 work station. All the programs have very short running times, except for the simple map-coloring program. Thus to obtain a more reliable timing measurement, they are queried several times and the total time spent is recorded. Then the same programs are run on an unmodified interpreter and the two sets of times are compared to give the results. In each table entry, the percentage change of CPU time of the intelligent scheme with respect to the naive scheme is entered. Thus a negative percentage means a speed up and a positive percentage means a slow down. The number of machine cycles executed in the scheme of [KuLi87] is also included in column three, since the authors of [KuLi87] think that the CPU time taken by their simulator may not be accurate.

Among the three schemes, [BrPe84] is the most intelligent in the sense that their scheme eliminates the largest amount of search space in theory. In terms of run time, however, their scheme is not as practical as ours because of the substantial overhead involved in the unification analysis. The scheme of [KuLi87] sometimes fails to pin point the cause of failure directly. During the generation of a B-list, whenever a predicate P fails, they will consider the generators of *all* the variables that occur in the argument of P and backtrack to the most recent one. This backtrack point may not be the one that caused the unification failure. Thus P may fail again.

Our scheme gives similar or better speedup when a program backtracks heavily. In addition, it gives a small overhead (around 10%) for programs which do not benefit much from intelligent backtracking. (That is, programs which are deterministic or where backtrackings are done to the nearest non-deterministic ancestor node. For example, the tree and the clever 6-queen programs.) Since the backtracking scheme is controlled by a high level Prolog predicate, the user can turn it on and off at any time. For those programs where intelligent backtracking only creates overhead, the user could turn off the scheme. When the scheme is turned off, no overhead will entail.

At the present time our modified interpreter cannot handle Prolog programs with *assert, retract,* or *prove* unless we turn the scheme off. Although a full scale implementation is not finished, the present results seem promising.

† UNIX is a trademark of Bell Laboratories.

Comparison of execution times

| Query | Bruynooghe & Pereira (CPU) | Kumar & Lin (# cycles) | Kumar & Lin (CPU) | You & Wang (CPU) |
|---|---|---|---|---|
| database query | -20.0% | -8.8% | -28.9% | -68.9% |
| 5-queen simple | -36.0% | +165.2% | +16.6% | -73.8% |
| 5-queen clever | +99.0% | +92.6% | +7.5% | +9.1% |
| binary tree | +44.0% | +11.6% | +5.9% | +11.1% |
| mapcolor clever | +63.0% | +8.9% | -3.6% | +11.3% |
| mapcolor simple | -99.7% | -99.2% | -99.9% | -92.6% |

# LEW-P: Learning by Watching in the Planning Domain.

**Patrick Constant*, Stan Matwin§, Franz Oppacher+**

*Ecole Nationale Superieure des Telecommunications, 46 r. Barrault, 75016, and Cognitech, Inc., Paris, France.

§Dept. of Computer Science, University of Ottawa, Ottawa, Ont. K1N 9B4, Canada.[1]

+School of Computer Science, Carleton University, Ottawa, Ont. K1S 5B6,Canada.[1]

**Abstract**

This paper describes LEW (Learning by Watching), a novel learning technique implemented in Prolog, and discusses its application to the learning of plans. LEW is a domain-independent learning system with user-limited autonomy that is designed to provide robust performance in realistic knowledge acquisition tasks in a variety of domains. It partly automates the knowledge acquisition process for different knowledge types, such as concepts, rules and plans. The inputs to the system, which we call *cues*, consist of an environmental component and of pairs containing a problem and its solution. Unlike traditional forms of 'Learning by Example' (e.g.[Winston 1984]), in which the system uses the teacher's answer to improve the result of a prior generalization of an example, LEW treats the problem-solution or question-answer instances, i.e. the cues themselves, as the basic units for generalization.

**Keywords:** machine learning, knowledge acquisition, planning, learning by watching.

## 1. Introduction

This paper describes LEW (Learning by Watching), a novel learning technique implemented in Prolog, and discusses its application in the planning domain. Our approach is presented more rigorously in [Constant et.al. 1988], and its use in the construction of knowledge-based software advisor systems is described in [Constant et.al. 1987]. LEW, a domain-independent learning system with user-limited autonomy, is designed to provide robust and reliable performance in realistic knowledge acquisition tasks. It partly automates the knowledge acquisition process for different knowledge types and in a variety of different domains, and may thereby help overcome the so-called 'knowledge acquisition bottleneck' phenomenon [Boose, Gaines 1986].

The inputs to our system - which we call *cues* - consist of an optional environmental component (see below) and of pairs containing a specific problem and its specific solution. Unlike traditional forms of 'Learning by Example' (e.g.[Winston 1984]), in which the system uses the teacher's answer to improve or modify the result of a prior generalization of an example, LEW treats the problem-solution or question-answer instances, i.e. the cues themselves, as the basic units for generalization.

Most of the learning systems described in the current literature are vehicles for exploring theoretical issues in learning and only few are robust enough to support knowledge acquisition for practical knowledge bases [Michalski et al. 1986]. Even the systems used to facilitate realistic knowledge acquisition tasks suffer from various restrictions. Most of them work only in domains characterized by a small number of dimensions (for an exception, see [Lebowitz 1986]). Some have very limited representation languages such as feature vectors or monadic predicate logic [Quinlan 1986]. Others, i.e. the ones that function incrementally, require noise-free data [Winston 1984]. Alternatively, noise-resistant systems [Quinlan 1986] require extensive statistics and initial availability of all positive and negative instances. Still others presuppose complete and formalized theories of their domains [Mitchell et al. 1986; DeJong, Mooney 1986].

Only few, if any, systems seem capable, without major modifications, of learning different types of knowledge, such as concepts and rules as well as plans.

Section 2 briefly explains the design goals that have influenced the construction of LEW, and section 3 presents the basic learning algorithm.

[1] The work described here has been partially supported by NSERC grants No.A2480 and A0313, and by a grant from Cognos Inc., Ottawa, Canada.

Section 4 illustrates LEW's functioning as a plan learner with several examples and compares LEW briefly with some other knowledge acquisition tools. Section 5 concludes by mentioning some possible extensions and enhancements.

## 2. Design Goals

It is our main goal to create a practically useful, robust, and domain-independent knowledge acquisition and refinement tool. This goal implies the following criteria on which the design of LEW is based.

- The system should be capable of learning different types of knowledge.
- Learning should be incremental.
- Learning should be order-independent.
- The system should be noise-resistant.
- The representation language should be unrestricted.
- The system should be able to learn 'from scratch', i.e. starting with an empty knowledge base.
- System performance should degrade gracefully.
- The growing knowledge base should be kept efficiently indexed.

LEW combines aspects of Learning by Example and of Learning by Analogy in order to realize its objective of *knowledge refinement*, i.e. to pack a maximum amount of knowledge into a minimal number of cues.

Knowledge refinement is achieved by keeping the knowledge base organized into *similarity clusters*, i.e. groups of closely related cues. These clusters serve as indices into the knowledge base and make learning by analogy feasible: given a new question, the clusters facilitate access to and matching against old cues with relevantly similar questions. Once such a cue is found, its answer is transformed in accordance with the similarities established by the match and given, as well as remembered, as the answer to the new question.

The structural similarity that the cues in each cluster bear to one another is not only the basis for our indexing scheme but also supports generalization. Since successful generalizations have the effect of collapsing two or more clusters into one, the efficiency of the indexing scheme is enhanced by the quality of the obtained generalizations. A more efficient indexing scheme, in turn, increases the chances for more powerful subsequent generalizations.

LEW's learning behavior is, thus, reminiscent of the functioning of dynamic memory based approaches [Schank 1982; Kolodner 1984; Lebowitz 1986]: the continuous reorganization of the knowledge base in terms of previous generalizations facilitates the detection and exploitation of fresh structural analogies; the latter may lead to new generalizations which trigger further re-indexing.

LEW could be used as an autonomous learner but we have decided, for practical reasons, to impose on it a protocol of interaction with an expert. This protocol allows LEW to make only relatively reliable generalizations by itself and to prompt an expert to validate the rest (see [Constant et.al. 1988] for more detail).

## 3. The Learning Algorithm of LEW

As remarked above, a *cue* consists of a question-answer pair with an optional environmental component, and is represented as follows:

- cue(Cluster_Index, Cue_Index, Environment, Question, Answer).

The Question consists of a marker indicating the question type - such as 'How do I' or 'Why' - and a phrase representing its propositional content. Both the Answer and the Environment are lists of phrases. In certain cases, the Answer may comprise a list of lower-level questions, the answers to which are taken to jointly constitute the given Answer. The Environment, if nonempty, can specify domain knowledge to constrain generalization or, in planning applications, describe the initial environment that is consulted to see if an action's preconditions are met.

When LEW is supposed to learn plans, it is also provided with a basic repertoire of *actions* which have the following form:

- action(Action_Index, Input_Environment, Action, Output_Environment).

Each action predicate is to be interpreted as asserting that performing Action in the Input_Environment yields a state described by Output_Environment. Input_Environment and Output_Environment are lists of phrases, while Action is a single phrase. (The Action_Index will be omitted in several illustrations below).

A *similarity cluster* is a set of cues each of which is linked to at least one other cue in the cluster by virtue of having a question or an answer in common with it or of being synonymous with it or of belonging to the same type (see below). The generalization procedure is only applied within one such cluster because, intuitively, generalizing on sets of less similar cues would lead to a proliferation of far misses.

Domain-specific *synonyms* are supplied by the expert or inferred on the basis of single differences between cues. Thus, LEW creates a synonym in the following two situations: either two cues have the same question (answer) and the two answers (questions) differ by only one word and the two environments are identical or differ by that same word, or two identical cues have environments that differ by only one word. In either case, LEW assumes such cues themselves to be synonymous and collapses them into one.

*Types* are either specified by the expert before learning begins or created by LEW as a result of learning. A type is created whenever two questions differ by only one word, the two corresponding answers differ by only one other word, and the two corresponding environments differ by at most these two words. Naturally, learning proceeds more smoothly and generalizations are more powerful in the presence of a well developed, hierarchized type structure supplied by an expert. But it should be noted that useful learning takes place even without any prespecified type information (see [Constant et.al. 1988]).

Learning and knowledge refinement occur whenever the assertion or retraction of a cue by the expert, i.e. its presentation as a positive or negative example, triggers the process of knowledge base reorganization (KBR for short). Since KBR is a costly process, it is only performed at times when there are no other demands on the system; in practice, this amounts to storing expert-supplied cues (e-cues, for short) until the expert puts LEW explicitly into KBR mode.

In KBR mode, each e-cue can bring about a complete reorganization of memory, according to the following algorithm:

- Compute minimal environments.

(This process of computing minimal environments is described in detail in section 4.)

- Use negative examples to specialize.

  Negative examples serve to prune overgeneralization. If e-cue is presented as a negative example that does not yet occur in the knowledge base, it is put on a list of rejected cues that is consulted by LEW to prevent wrong generalizations. If e-cue is presented as a negative example, but is already asserted in the knowledge base, it is removed from its similarity cluster. This cluster is then restructured from scratch (see below).

  - Determine the extent of the memory reorganization required by a positive e-cue.
    - Introduce e-cue in as many different similarity clusters as possible.

      Thus, e-cue is put in each cluster containing at least one cue that has a question or an answer in common with e-cue or is synonymous with it or belongs to the same type.
    - Collapse the modified similarity clusters into one cluster.

      The clusters modified by the recent introduction of e-cue have, of course, e-cue in common, and constitute by definition a unique larger cluster. This new cluster is the only one affected by the next step.
  - Restructure the new cluster from scratch.
    - First, forget all acquired synonyms and types.

      Convert all cues in S to non-compact form by replacing all synonyms and types with their values.
    - Second, construct new synonyms.

      Each cue in the unpacked cluster S is compared to all others, and all possible synonyms are created. The propagation of the new synonyms throughout S

constitutes one form of learning in LEW.

- Third, construct new types.

  Type construction constitutes another form of learning in LEW. The overriding concern in type construction is to obtain a maximally compact cluster, i.e. one that both has a minimal number of cues and preserves all prior knowledge. Such a cluster is determined as follows: different knowledge-preserving clusters will result, depending on what types are built on what cues. From these temporarily created clusters LEW first selects those that have a minimal number of cues. From this set, it then selects a cluster whose effective creation leads to the construction of a minimal number of new types. This choice criterion has the double effect of condensing the knowledge base and of reducing the number of cues that are submitted to the expert for validation.

The above steps are repeated for all intermediate results of generalization until no further changes happen.

The first step of the process of KBR and its role in the acquisition of plans is described in the next section.

## 4. Plan Learning

In order to learn generalized sequences of actions, i.e. general plans as quickly as possible, LEW determines for each basic action A which parts of the description of its Input_Environment and Output_Environment ($E^i_A$ and $E^o_A$ for short) are *relevant* to performing the action. Since (a part of) an environment description is irrelevant to the execution of an action if it is the same in $E^i_A$ and $E^o_A$, the following formula is used to compute the relevant, i.e. minimal Input_Environment, $min\_E^i_A$, for each action A:

$min\_E^i_A = E^i_A - [E^i_A \cap E^o_A]$. Similarly, $min\_E^o_A = E^o_A - [E^i_A \cap E^o_A]$.

For example[2], in the case of the action

action(1, [cube(A) ontop cube(B), cube(B) ontop table], put cube(A) on table, [cube(A) ontop table, nothing ontop cube(B), cube(B) ontop table]),

'cube(B) ontop table' is superfluous, and LEW will actually enter into its knowledge base the following action and its associated cue (we do not elaborate here how parts of the cue other than the environment are obtained):

action(1, [cube(A) ontop cube(B)], put cube(A) on table, [cube(A) ontop table, nothing ontop cube(B)]),

cue(1,1, [nothing ontop cube(A), cube(A) ontop cube(B)], "how do i drop cube(A) on table?", "put cube(A) on table").

Once the minimal environment, $min\_E_A$, of each basic action A is determined as above, the minimal environment, $min\_E_C$, for any cue C can be computed with the following algorithm.

First, compute $E^o_C$ of cue C:

Let C be cue(_,_, $E^i$, Question, Answer), where Answer is "$action_1$, $action_2$,..., $action_n$". Since LEW has to consider $min\_E^i_A$ and $min\_E^o_A$ for each $action_i$ in Answer, we assume that the knowledge base at this point contains:

action($E^i_1$, $action_1$, $E^o_1$), action($E^i_2$, $action_2$, $E^o_2$), ... , action($E^i_n$, $action_n$, $E^o_n$), where $E^i_j$, $E^o_j$, $1 \le j \le n$, are minimal. Then

$E^o_C = [...[[[[E^i - E^i_1] \cup E^o_1] - E^i_2] \cup E^o_2] \cup ... - E^i_n] \cup E^o_n]$.

Second, compute $min\_E_C$, using $E^o_C$:

Let C be as above, and let $E_1 = [E^i - E^i_1]$, $E_2 = [[[E^i - E^i_1] \cup E^o_1] - E^i_2]$, ..., $E_n = E^o_C = [...[[[E^i - E^i_1] \cup E^o_1] - E^i_2] \cup ... - E^i_{n-1}] \cup E^o_n]$. Then

$min\_E_C = E^i - \{\cap_{j=1,n} E_j\}$.

The examples below illustrate the concept of a minimal environment and some of the uses to which it is put by KBR in the learning of plans.

**Example 1**: Suppose that we have

action(1, [cube(A) ontop cube(B)], put cube(A) on table, [cube(A) ontop table, nothing ontop cube(B)]),

action(2, [nothing ontop cube(D)], put cube(C) ontop cube(D), [cube(C) ontop cube(D)]),

and that the expert enters the cue $C_1$

cue(_,_,[cube(D) ontop table, cube(C) ontop cube(B),

[2] In these examples, *cube(A)* denotes a logic term with a variable *A*, and *cube(a)* denotes a fully instantiated logic term.

cube(B) ontop cube(A), cube(A) ontop table], "how do i put cube(D) on cube(B)?", "put cube(C) on table, put cube(D) ontop cube(B)").

Then $\min\_E_{C_1} = E^i - [E_1 \cap E_2]$, where $E^i$ is the Environment of $C_1$, $E_1 = E^i - E^i{}_1$ = [cube(D) ontop table, cube(C) ontop cube(B), cube(B) ontop cube(A), cube(A) ontop table], $E_2 = [[E_1 \cup E^o{}_1] - E^i{}_2]$ = [cube(D) ontop table, cube(B) ontop cube(A), cube(A) ontop table, cube(C) ontop table, nothing ontop cube(B)]. Thus, the environment of $C_1$ is updated to $\min\_E_{C_1}$ = [cube(C) ontop cube(B)].

**Example 2**: Assume the same two actions as in Example 1 and their associated cues

cue(1,1,[nothing ontop cube(A), cube(A) ontop cube(B)], "how do i drop cube(A) on table?", "put cube(A) on table"),

cue(2,1,[nothing ontop cube(C), nothing ontop cube(D)], "how do i place cube(C) ontop cube(D)?", "put cube(C) ontop cube(D)").

At this point, LEW does not have enough information to answer, for instance, a user's question "how do i put cube(c) on table?" in the particular environment [cube(a) ontop cube(b), cube(b) ontop cube(c), cube(c) ontop cube(d)]. But with just one further e-cue, say,

cue(3,1, [cube(A) ontop cube(B), cube(B) ontop cube(C)], "how do i put cube(B) on table?", "put cube(A) on table, drop cube(B) on table"),

LEW can now put any element of any tower of cubes on the table. Since the question of cue(3,1,...) matches the above user's question and its environment is included in the environment specified by the user, LEW can answer the user's question with "put cube(b) on table, drop cube(c) on table", and, if more detail is wanted, with "put cube(a) on table, drop cube(b) on table, drop cube(c) on table".

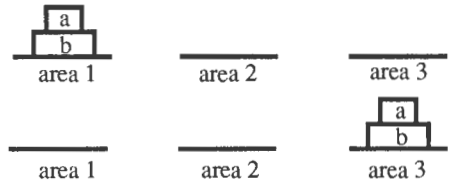
**Example 3**: This example illustrates how LEW comes to be able to solve the Towers of Hanoi problem. Omitting notational details, the following description of the figure below is entered by the expert in the knowledge base:

$E^i$ = [nothing ontop $area_2$, nothing ontop $area_3$, b ontop $area_1$, nothing ontop a, a ontop b, a smaller_than b],

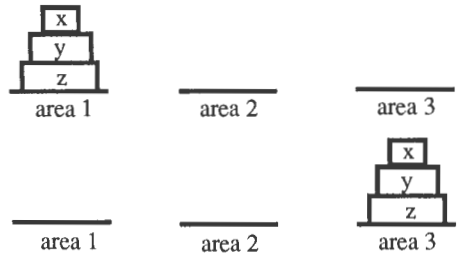
Question = "how do i move $tower_b$ to $area_3$?" (where $tower_x$ represents a tower consisting of the block x and all the blocks above x),

Answer = "move a to $area_2$, move b to $area_3$, move a to $area_3$", and

$E^o$ = [nothing ontop $area_1$, nothing ontop $area_2$, b ontop $area_3$, nothing ontop a, a ontop b, a smaller_than b].




Next, the user asks question $Q_1$ = "how do i move $tower_z$ to $area_3$?" in the context of environment $E_1$ (again represented graphically for brevity).




Finally, the user problem is matched against the knowledge base as follows:

a) An environment $E_2$ is found such that $E_1$ includes $E_2$ and $E_2$ is the largest such environment. In our example, $E_2 = E^i$. The necessary name conversions are done during matching, e.g. x → a, y → b.

b) A question $Q_2$ accompanying $E_2$ in a cue in the knowledge base is identified, and again the necessary name conversions are done as in a).

c) $Q_1$ is divided into ($Q_2$, $Q_3$, $Q_4$), where $Q_3$ moves what remains of $E_2$ in $area_1$ to $area_3$, and $Q_4$ moves the result (output environment) of $Q_2$ to $area_3$:

$Q_1$ = "how do i move $tower_c$ to $area_3$?", $Q_2$ = "how do i move $tower_b$ to $area_2$?", $Q_3$ = "how do i move c to $area_3$?",

$Q_4$ = "how do i move $tower_b$ to $area_3$?".

d) $Q_2$, $Q_3$ and $Q_4$ are answered by fetching answers $A_2$, $A_3$, and $A_4$ from the knowledge base and, for $A_2$ and $A_4$, converting the directions of moves accordingly, e.g. $area_2$ replaces $area_3$ and conversely in $A_2$, and $area_2$ replaces $area_1$ and conversely in $A_4$.

As this example shows, LEW learns to solve the Towers of Hanoi problem and similar tasks efficiently and with a minimum of background information. It achieves this by breaking a problem or question $P_1$ into two problems: an existing problem $P_2$ and the "remainder" $P_3$. $P_2$ is a subproblem of $P_1$ in the following sense: the environment of $P_2$ is contained in the environment of $P_1$, and a solution to $P_2$ will contribute to obtaining a solution to $P_1$. Candidates for $P_2$ are selected and compared on the basis of their environments, and the one with the maximal environment is chosen as $P_2$. Answers to $P_2$ and $P_3$ are combined into the answer to $P_1$. The problem $P_1$, including its environment, problem statement and answer is stored in the knowledge base for future use when solving problems larger than $P_1$.

From the preceding description of the basic algorithm and the examples it should be clear that LEW has several unique features.

The underlying learning technique does not, unlike traditional forms of 'learning by example' [e.g. Winston 1984, ch. 11], use a teacher's answer to improve the result of generalizing an example but treats the question-answer or problem-solution pair as the basic unit for generalization.

As new knowledge is added to the knowledge base, an increasingly efficient indexing scheme is maintained and updated through the ongoing processes of generalization on cues and the formation of similarity clusters. The efficiency of this indexing scheme depends on the quality of the generalizations obtained: a poor choice of question-answer pairs yields a less compact knowledge base, rather than leading to incorrect generalizations. Moreover, the generalizations themselves lead normally to a condensation of knowledge. This condensation effect is often quite dramatic (see [Constant et al. 1987]).

Incrementally learning systems presuppose that the inputs are presented to them in a carefully chosen order. Different sequences of inputs may lead to the learning of different, and even wrong, concepts. By contrast, LEW, although an incremental learner, enjoys the property that its results are not influenced by the order in which the cues are entered (see [Constant et al. 1988]). Moreover, unlike other incremental systems, LEW retains all cues provided by the expert, thereby facilitating the construction of reliable knowledge bases.

Some incrementally learning systems, i.e. those relying on explanation-based generalization (see, e.g., [Mitchell et al. 1986; DeJong, Mooney 1986]), presuppose that they are supplied with a goal concept and a comprehensive, fully formalized theory of the domain before any learning on the basis of examples begins. Explanation-based generalization assumes that the domain theory is sufficiently strong to prove that the example instantiates the goal concept. Since the theory, with the example as an additional premise, deductively implies the inferred generalization, the results of explanation-based generalization are guaranteed to be correct relative to the theory. This strong property of explanation-based generalization can be achieved only in those relatively rare domains for which complete and consistent theories are available. However, explanation-based generalization does not address the question how such theories might be acquired in the first place.

Our decision to confine learning to near miss situations counsels learning in small steps. The latter policy is referred to in [Winston 1984, p.401] as Martin's Law: 'You can't learn anything unless you almost know it already'. It seems to us that this common-sensical law is often misinterpreted to mean that you can't learn anything unless you already know a lot. While we agree that learning should proceed in small steps, i.e. avoid far misses, we don't subscribe to the view that to learn anything one already has to know a lot. On the contrary, we wish to emphasize that LEW can learn useful things even in the absence of prior domain knowledge or background information in the form of a type hierarchy. In fact, it can even learn an initial type hierarchy from scratch (see [Constant et al. 1988]).

## 5. Concluding Remarks

We have described a Machine Learning method that has been designed as a realistic knowledge acquisition tool, rather than an autonomous learning system. Our method is incremental, relatively noise-resistant, and does not impose any serious restrictions on the representation language. It uses both generalization and specialization to generate a knowledge base that can subsequently be used by an expert system. It is applicable in domains for which no axiomatization of the domain knowledge, such as is often required by explanation-driven methods, is available. It relies on the participation of a cooperating expert who provides the problem-solution pairs, but the knowledge can be entered from scratch, i.e. starting with an empty knowledge base. Moreover, in this paper we show that LEW is applicable to the learning of plans, and we have demonstrated with an example that it outperforms some other recently reported methods proposed for this task (e.g. [Anzai 1987]).

## References


Anzai, Y., Doing, Understanding, and Learning in Problem Solving. In [Klahr et al. 1987], pp.55-97.

Boose, J.H., Gaines, B.R., eds, Proceedings of the Knowledge Acquisition for Knowledge Based Systems Workshop. Banff, 1986.

Constant, P., Matwin, S., Szpakowicz, S., Question-Driven Approach to the Construction of Knowledge-Based Software Advisor Systems. Proceedings of the Third Conference on AI Applications, pp. 29-37, Orlando, 1987.

Constant, P., Matwin, S., Oppacher, F., LEW: Learning by Watching. A Machine Learning System for Knowledge Acquisition. Forthcoming 1988.

DeJong, G., Mooney, R., Explanation-Based Learning: An Alternative View. Machine Learning 1, pp.145-176, 1986.

Klahr, D., Langley, P., Neches, R., Production System Models of Learning and Development. MIT Press, Cambridge, Mass., 1987.

Kolodner, J., Retrieval and Organizational Strategies in Conceptual Memory. Lawrence Erlbaum, Hillsdale, NJ., 1984.

Lebowitz, M., Concept Learning in a Rich Input Domain. In [Michalski, Carbonell, Mitchell 1986], pp.193-214, 1986.

Michalski, R., A Theory and Methodology of Deductive Learning. In [Michalski, Carbonell, Mitchell 1983], ch. 4, 1983.

Michalski, R., Carbonell, J., Mitchell, T., Machine Learning, Volume 1. Morgan-Kaufmann, 1983.

Michalski, R., Carbonell, J., Mitchell, T., eds., Machine Learning. Volume 2. Morgan - Kaufmann, 1986.

Mitchell, T., Keller, R.M., Kedar-Cabelli, S.T., Explanation-Based Generalization: A Unifying View. Machine Learning 1, pp.47-80, 1986.

Quinlan, J.R., The Effect of Noise on Concept Learning. In Michalski et al., pp.149-166, 1986.

Schank, R., Dynamic Memory: A Theory of Learning in Computers and People. Cambridge University Press, 1982.

Winston, P.H., Artificial Intelligence, 2nd edition, Addison-Wesley 1984.

# Generic Strategies and Representations for Communications Networks Sales

Innes A. Ferguson
Daniel R. Zlatin

*Artificial Intelligence Exploratory*
*Bell-Northern Research Ltd.*
*Ottawa, Ontario, Canada K1Y 4H7*

## ABSTRACT

This paper focuses on our research into identifying several generic tasks and knowledge types in the networks sales domain, and describes the architectural approaches that have been used to model each. This is done by looking at **ENS** (**E**xpert **N**etwork **S**elector), a knowledge-based tool which gives advice on the design and sales of customer networking and communications facilities. A multi-paradigm knowledge representation scheme has been successfully implemented in Prolog to define the various types of knowledge and information typically handled by communications networks sales staff.

*Keywords*: knowledge-based systems, expert systems, telecommunications, networks design, design tools, networks sales.

---

Cet article rend compte de nos recherches pour identifier plusieurs méthodes génériques et types de connaissances dans le domaine de la vente des réseaux, et décrit les approches architecturelles qui ont été utilisées pour modéliser chacun d'eux. Ceci est fait en considérant **ENS** (**E**xpert **N**etwork **S**elector), un système a base de connaissances qui conseille la conception et la vente aux clients de moyens de communications et de mises en réseaux. Une classification multi-paradigmes de la représentation des connaissances a été implantée avec succès en Prolog pour définir différents types de connaissances couramment manipulées par le personnel de vente des réseaux de communications.

*Mots clés* : systèmes à base de connaissances, experts systèmes, télécommunications, conception de réseaux, outils de conception, ventes des réseaux.

## 1. Introduction

The aim of this paper is to present the architectural principles and generic concepts which we have identified in our chosen sales domain and subsequently modelled in the Expert Network Selector (**ENS**). ENS [1] is a knowledge-based tool which is currently being developed to assist Bell Canada networks sales people in their daily operations, i.e. in the design and sales of communications networks. The tool employs a multi-paradigm knowledge representation scheme to translate a customer's communications and networking requirements into fully-configured solutions. Implemented in Prolog, frames and rules have been successfully combined in ENS to create an intelligent, user-friendly tool for sales staff who may not possess some of the in-depth technical knowledge required to operate "conventional" design tools.

Communications networks design is a complex, iterative task which involves determining from a large search space the "best" available carrier service meeting a customer's needs. Typically, a sales person or engineer (either or both, depending on the complexity of the problem being solved) will gather a set of customer requirements, and from these produce a functional specification of the design to be used. The parameters which make up this description will normally be analyzed using one or more network design tools – in most cases, several iterations of customer requirements specification and analysis will take place before a fully-configured solution is produced. Certain aspects of network design make this already complex problem harder still. Most notably, services and facilities (whether owned by the computer manufacturer, the customer, or the telephone company itself) constantly change, making it increasingly harder for sales staff to produce optimal and up-to-date quotations. In addition, relevant knowledge and information about network design is usually distributed among various sales, engineering, and product experts, which at times can contribute to incomplete or conflicting interpretations of the overall design process. In human terms, this makes the design task time-consuming and thus expensive.

Artificial Intelligence (AI), and in particular, Knowledge Based Systems (KBS) technology, provides attractive means for managing the complexity of the network design process [1,2]. This is achieved by facilitating the use of heuristic (rather than algorithmic) problem solving techniques, as well as declarative languages for the explicit representation of the more "volatile" knowledge types generally associated with this domain. Interest has grown steadily in the application of KBS technology to the problems of network configuration and design. Notable examples of work in this area are XCON [3,5], which configures VAX-11[1] and PDP-11[1] computer systems; and DESIGNet[2][4], which produces topological designs (backbone networks and terminal clusterings) for wide-area data networks. Both tools are aimed at skilled designers or analysts, and consequently have limited usefulness for less technical sales representatives. Aware of this, the same team that created XCON later developed XSEL [5], a rule-based, higher-level, interactive front-end to XCON. Requiring less detailed input, XSEL allows sales staff to quote prices on orders at their point of sale, and can assist less experienced personnel in filling out order forms by displaying relevant, legal field values. ENS differs from XSEL in that, besides performing sales tasks similar to those found in XSEL (such as selecting the network components to be configured), it also performs various configuration tasks which are more akin to those performed by XCON.

The generic concepts which are discussed in this paper, namely sales tasks and knowledge types, can be split into three categories: those which are essentially knowledge representation

issues, those which form part of the tool's problem solving strategy, and those which relate to the user interface component of ENS. Details of these, together with a brief introduction to the chosen sales domain and some thoughts on the lessons learned and future directions, make up the remainder of this paper.

## 2. The Sales Domain

ENS [1] was originally designed to assist data network sales representatives in gathering a customer's data communications requirements, choosing a set of technically viable data services according to various selection criteria (see fig. 1), and then configuring and costing the services for presentation to the customer. Due to a positive response from Bell Canada's sales support organization, we were encouraged to further enhance and generalize this prototype system.

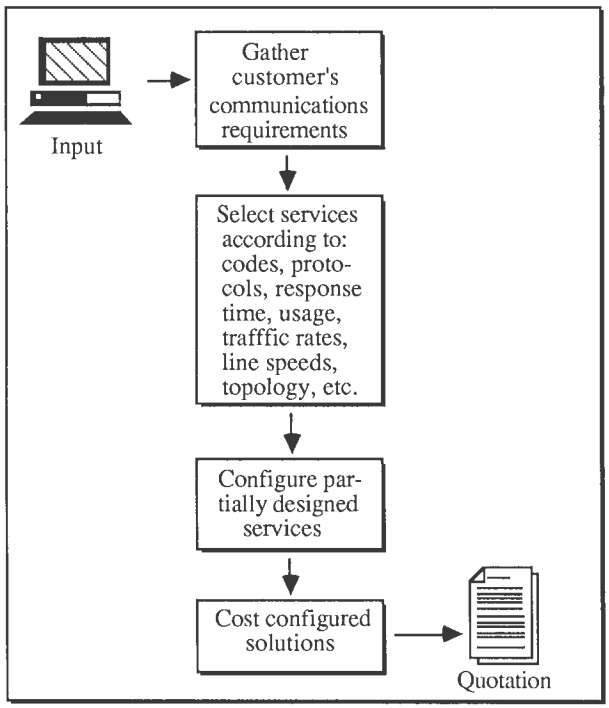



*fig. 1: Data communications networks design flow.*

Data network sales representatives are faced with several challenging tasks. One of these is having to cope with the vast body of information about the various services and facilities they sell. In addition, much of the information (e.g. rates, tariff structures, service descriptions, customer equipment specifications, etc.) is of a rapidly evolving nature. Existing design tools are focused on isolated aspects of the network design process, and are therefore better suited to experienced designers or analysts. These tools provide little guidance for more naive users, as designers are expected to possess a fairly strong technical background. Many sales representatives must deal with a large number of customers and thus do not have the time to use tools that demand substantial user input.

On selecting the data communications design domain, and in particular, in aiming to assist the sales representatives in this field, we felt that a successful solution would have to satisfy some important requirements. Namely, that it: (i) minimize the amount of user input required during a consultation; (ii) abstract the level of dialogue to a point at which the sales staff would feel comfortable; (iii) take into account more experienced sales persons' heuristics, which they use to speed up the design process; (iv) be able to handle qualitative and subjective customer requirements, since these cannot always be defined in purely technical terms; (v) provide adequate means for updating knowledge as service-specific parameters change in time; and (vi) allow the sales person to quickly explore alternative customer recommendations by means of a high-level, interactive simulation or *what-if* facility.

ENS constitutes an environment for solving various network sales and design tasks, as outlined above. By making use of various knowledge representation techniques, technical, heuristic, and volatile knowledge can be represented in a declarative and easily modifiable manner. Besides automating various repetitive and time-consuming sales tasks, ENS provides several useful features which can help simplify and shorten the overall design process. It will be shown below how specific knowledge representation, abstraction, explanation, and simulation techniques have been combined in the tool in an attempt to achieve this.

## 3. Knowledge Representation

Knowledge in ENS is represented as both frames and rules, with Prolog as the chosen implementation vehicle. After much experimentation with various data structures and paradigms for representing our domain knowledge, we were able to home-in on some suitable techniques. The top-level appearance of customer networks – independent of the actual technology or services used – was found to be very similar in each case we studied, and could therefore be represented using a fairly *static* structure. However, this structure would also have to provide mechanisms for easily replicating – through either copying or inheritance – certain sub-network components which invariably appear repeated in a single customer network (e.g. nodes or links). For this reason, frames have been chosen. Application-specific design information, on the other hand, is reasonably volatile, and may change each time a network service description changes. This type of knowledge, we have found, is more adequately represented as production rules.

For the purpose of describing how each representation paradigm is being used, the following main knowledge categories can be identified:

- a hierarchy of frames to represent the customer's network requirements,
- a rule base of design/selection criteria to generate partial designs from these requirements, and
- frame-like design plans which are used to generate and represent fully-configured solutions.

The remainder of this section will describe each category in more detail.

### • Customer Requirements Hierarchy

For the purpose of creating a design, ENS views a network as consisting of a hierarchy of sub-network design components. Networks are represented as a collection of customer applications (e.g. process control, remote job entry, file transfer) running on a set of network links (see fig. 2). Each link, in turn, can be described by its two node locations, where each node can be described by the type of termination equipment it represents. Each of these design components is represented as a frame whose slot values are obtained either from the user or from a knowledge base of stored defaults. The structure of the design component hierarchy is essentially fixed by ENS; however, since certain network components often appear repeated any number of times (e.g. a customer's links),

instances of the frames corresponding to these will be added to the hierarchy dynamically. Likewise, the existence of certain frame types will not be known until run-time, so these must also be inserted dynamically. For example, the Customer-Termination and Customer-Application frames inherit their default values and properties from libraries (implemented as IS-A hierarchies) of previously-defined termination and application classes (see fig. 3).

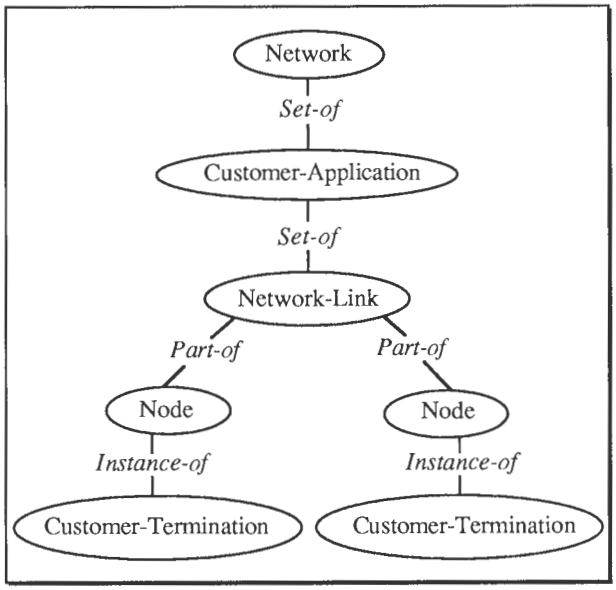



*fig. 2: Customer Network Requirements hierarchy.*

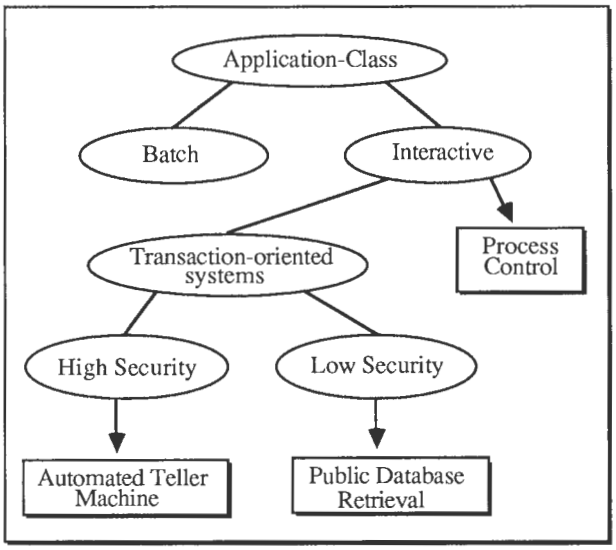



*fig.3: New network application types can be defined using inherited defaults, and then stored in a hierarchical library for future use.*

## • Design Criteria Rule Base

The second knowledge category is formed by the rule base of service-specific design and selection criteria. Each **Criterion** (be it technical, qualitative, or heuristic) is composed of two parts: a **Design-Task** and a **Constraint-Set** (see fig. 4). The first of these generates a minimal set of partial design data that will be used to determine which network carrier services are viable for the customer, and those which are not. The results (rejections or recommendations) will be recorded in special slots of the Customer-Application frames, which will later be used by the user interface module (see below) to explain the reasoning behind the decisions made. The second part of the rule, as its name indicates, is a set of design constraints which are ordered according to our domain experts' own heuristics, and which become activated each time a **Design-Task** has finished executing.

```
Criterion: Packet-switched-Host-Access
   Design-Task:
     Determine-Incoming-Links(N),
     Worst-case-Bit-Rate(B),
     Select-Access-Speed(B, S).
   Constraint-Set:
     {N * B <= S ;
       Redo(Select-Access-Speed(B, S))},
     {Valid-Host-Access-Speed(S) ;
       Redo(Determine-Incoming-Links(N))}.
```

*fig. 4: An example selection/design criterion.*

Each design constraint in a **Constraint-Set** can be viewed as consisting of two sub-parts: a boolean operation which is performed on a specific set of design parameters appearing in the associated **Design-Task**, and an instruction on what to do should this boolean operation fail. These instructions provide a certain degree of intelligent backtracking information, as they essentially point to design tasks which should be re-done in order to satisfy the failed constraint. The example **Criterion** in fig. 4 is used to design host access lines for a packet-switched carrier service. The **Design-Task** determines the number of incoming terminal lines, the worst-case bit-rate required by any one of these lines, as well as a "first guess" as to what access line speed should be chosen. The first constraint dictates that the number of incoming links times the worst-case bit-rate must not exceed the chosen access line speed. If it is exceeded, then a new access speed must be found and the constraint re-satisfied. Once this constraint can be satisfied, the next one will be attempted, and so on. By allowing the backtracking direction to be controlled in this manner, we can avoid making an exhaustive search at this time. It should be pointed out, however, that if a constraint cannot be satisfied after exploring all avenues within the current design criterion, ENS will backtrack (naively) to the previous **Criterion** and attempt to re-do its **Design-Task**. In the worst case, thus, an exhaustive search would be performed.

## • Design Plans

The third and last category of knowledge consists of a library of frame-like *design plans*. ENS regards a customer's links as the basic building block in the communications networks it helps design. In view of this, every different communications service link type known to ENS will have an associated template **Design-plan**. These frame-like structures (see fig. 5) have two basic parts to them: a **Props** slot that indicates which sub-link components originating from the partial design produced thus far will take part in the configured solution, and an **Event-Sequence** method that will dictate the sequence in which these components actually get configured. Once each **Design-plan** has been fully executed, ENS will attach them to their corresponding Network-Link frames so that they can later be used to generate a detailed, printable cost quotation.

```
Design Plan: Packet-Switched-Type-1

  Props:

     PAD(Node1, P1), PAD(Node2, P2),
     Access-Speed-Cost(Node1, S1, C1),
     Access-Speed-Cost(Node2, S2, C2),
     Usage-Charge(C3).

  Event-Sequence:

     Configure-Access(Node1, S1, C1),
     Configure-Access(Node2, S2, C2),
     Determine-Usage(Node1, P1, Node2, P2),
     Compute-Cost(C1, C2, C3).
```

*fig. 5: An example design plan.*

## 4. Problem Solving Strategy

As mentioned above, ENS's problem solving methodology is based on partial design plan generation with constraint propagation. That is to say, the total design space for a given application is divided into partial design stages, with either technical or heuristic constraints being applied to the appropriate design parameters at each of these stages. Constraints applied to parameters at a higher level (e.g. at the Network level) will propagate down to – and constrain – all parameters at levels below it (e.g. at the Link and Node levels). Constraints, which can be supplied either by the user (certain input fields implicitly become constraints) or the expert (these would already reside in the knowledge base), are used for two different, but complementary, purposes. The first of these is to ensure that technically viable design plans are being chosen at each stage in the design process. The second is to provide some intelligent backtracking information when a design failure occurs due to a constraint being violated. These constraints can be regarded as heuristics whose role is to direct the search for the associated design parameter which, upon re-calculating, will most likely ensure the constraint is not violated a second time. In this respect, our approach is similar to the *advice mechanism* described by Mittal and Araya in [6].

At the program level, ENS's design strategy can be seen to consist of three main stages. During the first of these, the data structure representing the customer's requirements (see fig. 2) is constructed. The "building blocks" for this will be obtained from either the user or from a knowledge base of stored, heuristic defaults. Once completed, this data structure is passed through to the second design stage where it is processed by the rules representing the service-specific selection criteria – this helps determine which communications services can be used to configure each of the applications running on the customer's network. This involves applying the **Design-Task** and **Constraint-Set** parts of each selection **Criterion** (see above) to the relevant slot values within the data structure. When completed, ENS is able to present the user with the partial design results achieved so far: a list of all technically viable communications services found, plus detailed justifications for those which were rejected on technical or qualitative grounds.

The third design stage amounts to a detailed configuration and costing of the partial designs produced thus far. Here, ENS "tags" each *logical* link in the customer's network according to its geographical node locations, termination node types, etc., in order to associate with each one a set of executable template design plans. Before executing these, however, the dollar costs of the link components appearing in each of the **Props** slots (see fig. 5) are first determined. Costing figures are currently held in an entity-relationship database to which ENS provides an interface. This interface is capable of generating optimized database queries based on the items found in each **Props** slot. After performing these queries, the returned cost figures are converted into Prolog facts, which are used by the **Event-Sequence** blocks of each of the design plans involved. On completion, each link's attached **Design-plan** shows the detailed component-level configuration produced – *logical* links can then be viewed as *physical* links.

Note that any of the design parameters defined during a session can be changed by simply returning to the user interface form in which it was originally defined, or by making use of the *what-if* facility. In the latter case, ENS effects a "snapshot" mechanism which effectively allows the freezing of any number of parallel customer scenarios, thus permitting the user to change design parameters at will and compare different network scenarios in a quick and cost-effective manner. In order to determine what gets affected by changing a specific design parameter, each of these editable parameters is associated with the **Design-Task** in which it first appeared. Similar (explicit) associations are made between this task, and any other tasks that should be re-visited whenever a certain type of change occurs. Once each relevant **Design-Task** and **Constraint-Set** pair has been re-satisfied, the new recommendations will be shown to the user.

## 5. User Interface

Besides being a very knowledge-intensive domain, communications networks design is also highly interactive. Because of this, the user interface portion of ENS occupies a large part of the system's total code (approximately two thirds). The latest appearance and functionality of the interface have been reached after numerous consultations with the tool's end-users – Bell Canada's sales representatives and support staff. In order to deliver a tool consistent with the computing environment with which they were familiar, a mainframe-based forms interface is being used. Implemented as frames, ENS treats forms and menus (the latter are just a special case of forms) as states in a Finite State Machine (FSM). This FSM consists of a set of states, together with appropriate inter-state transition arcs (these correspond to the function key settings which are made available to the user in each form), and a state-transition table consisting of a set of *NextState(CurrentState, PFKey, NextState)* Prolog facts. This generic FSM model has greatly simplified the prototyping effort of the user interface module by allowing efficient experimentation with different state definitions and state-transition table entries.

Besides simply providing a forms-based interface, our survey of the sales community has shown the need for providing several other important functions. These include a facility for storing and retrieving customer scenarios; libraries of commonly-used and generic network termination and application types from which user-defined types can inherit default values (see fig. 3); a facility for printing fully-detailed, costed quotations reflecting the design solutions that were finally arrived at; and a template-based, English-text explanation function (beyond the context-sensitive help mechanism also provided) to indicate why some network service types were not recommended, and why others were (see fig. 6). ENS also provides mechanisms for directly querying its own knowledge base on specific network services and rates facts. Our

experience with the Bell Canada data networks application has demonstrated that, in many cases, factual knowledge can be easily re-used when rules are broken down and coded as a *control-part* plus *facts*.

Perhaps the most useful facility that ENS can provide sales people with is a high-level *what-if* mechanism for quickly exploring real or hypothetical alternative customer configurations. Once a customer's network has been configured for the first time, ENS's users can quickly access and change most pieces of information provided by the customer. This facility immediately allows the sales person to see the effect on the customer's network after adding certain carrier service options, or after increasing the traffic, usage, and/or response time requirements on a particular network link.

- **Data**link[3] was excluded for the process-control application for the following reason(s) :

  - one of the nodes host-vax-4 or term-vt100-7 on link 9 (most likely the lower speed terminal) can't support the appropriate line speeds.

*(a) a typical rejection explanation.*

- Since the customer expressed a desire for controlling access to the network, **Data**pac[3] is well-suited since it provides several features and options for achieving this:

  - built-in features: reverse charge.
  - optional features: call blocking, SVC bar call. (See PSMM pp. 730:14, 18-20, 21-26 for details).

*(b) a typical recommendation explanation.*

*fig. 6: Two examples of generated explanations.*

## 6. Lessons and Current Status

A prototype version of ENS has been under development for just over two years. After numerous interviews and testing sessions with both sales representatives and network designers, we have learned to appreciate the wide range of sales and design tasks (and hence knowledge types) that have to be considered. Early versions of our system were largely rule-based, but our experience has shown that certain types of knowledge are more adequately represented using frames. Examples include knowledge that will be modified at run-time or whose structural component will be shared and refined to create other similar structures; and perhaps more interestingly – since this type of effort is generally underestimated – specifications of user interface forms or menus which, as mentioned above, constitute a significant part of the total program code. We have found Prolog a suitable vehicle for implementing both frames and rules.

As pointed out in section 2 above, any useful tool aimed at assisting communications networks sales staff would have to satisfy at least six basic requirements. It's worth summarizing at this stage which features of ENS have helped in satisfying these requirements:

(i) Minimizing user input has been achieved through the use of libraries of stored network termination and application types, as well as numerous stored default design parameters.

(ii) Abstracting the level of user dialogue to an appropriate level has been arrived at by careful consideration of both the sales persons' vocabulary, as well as their knowledge of the network design process. ENS's frame/FSM user interface model allowed fruitful rapid prototyping of this information.

(iii) Experienced sales persons' heuristics have been modelled in a very declarative style in the form of both design rules and constraint statements. The use of Prolog as the implementation language certainly contributed to this.

(iv) Qualitative and subjective customer requirements are currently interpreted in order to *qualify* any recommendations made by the tool. Eventually they will be used directly in the selection process, in much the same manner that technical requirements are today.

(v) The separation and subsequent representation of generic and application-specific tasks has greatly reduced the effort of updating the knowledge base whenever service- and application-specific information changes.

(vi) A quick and effective means for altering design data – while still maintaining a consistent network design – has been achieved through the provisioning of a *what-if* facility that is fully-integrated with the ENS concepts of design plans and constraint sets.

Encouraged by potentially large time-savings, a more consistent and unified approach to network design, and the prospect of being able to spend more time satisfying both new and existing customers, Bell Canada is showing a positive interest in ENS. Having embarked on a joint Bell Canada - Bell Northern Research plan to productize ENS, our major development activities are two-fold. On the one hand, we are continuing the process of extending the tool's functionality, adding more carrier services to our data communications application (currently Bell Canada's five major data services are handled), as well as considering voice and integrated voice and data (IVD) services. In addition, we are interested in further separating generic design tasks from those which are considered application-specific. Ideas are already emerging in this area [6,7,8,9], some of which are of particular interest to us. These include extending our present method of combining constraint propagation with heuristic (and user-provided) advice to aid backtracking on design failure, as well as further formalizing and enhancing our current use of design plans.

## 7. Conclusions

Requirements of sales-oriented design tools are very different from those of engineering tools. In particular, the latter tend to focus on isolated aspects of the design process, require substantial user input, and assume a strong technical background. Sales-oriented tools [1,5], among other things, must attempt to reduce the data-entry and knowledge overload of their users, handle qualitative customer requirements, and consider network design flexibility and evolution. Our research has identified both the specific sales tasks that must be addressed by a design tool, as well as the different knowledge types that are associated with these tasks.

We have implemented ENS, a prototype knowledge-based tool to assist sales staff in designing networks, and have discussed some of our research concerning the extraction of various tasks and knowledge types which we regard as being generic to the networks sales domain. The tool's architecture and problem-solving strategy make use of both frames and rules to represent these various knowledge types which, among other things, include design plan hierarchies and the heuristic and technical constraints used for pruning the design search space.

Exercising our tool on services other than data (in particular, voice and IVD) should further help us in refining our ideas on generic strategies for the networks sales domain. Our work so far has shown that ENS can successfully be used both to reduce design turnaround time for sales staff, and to generate more uniform and consistent design solutions.

## 8. Acknowledgments

The ENS project would not have been possible without the helpful assistance of various members of the network planning, sales, and design groups at Bell Canada: James Kennedy, Garth Mitchell, George Smith, and Louis LePage, in particular. From BNR I would like to thank Suhayya Abu-Hakima for providing valuable development support, and Dick Peacocke for suggesting useful changes to earlier drafts of the paper.

## 9. References

[1] Ferguson, I., Rabie, S., Kennedy, J. and Peacocke, D., "A Knowledge-based Sales Assistant for Data Communications Networks", *Proc. of the IEEE International Conf. on Communications*, June, 1987.

[2] Kai-li, K., "Expert Systems in Telecommunications Network Planning and Design", *Proc. of the 1st International Conf. on Applications of Artificial Intelligence in Engineering Problems*, pp. 1161-1164, vol. 2, April, 1986, Sriram, D. and Adey, R. (eds.), Springer Verlag (1986).

[3] McDermott, J., "R.1: A Rule-based Configurer of Computer Systems", *Artificial Intelligence* (19)1, pp. 39-88, January, 1982.

[4] Mantelson, L., "AI Carves Inroads: Network Design Testing and Management", *Data Communications* 15(8), pp. 106-123, July, 1986.

[5] McDermott, J., "Domain Knowledge and the Design Process", *Proc. of the Eighteenth IEEE Design Automation Conf.*, pp. 580-588, June, 1981.

[6] Mittal, S. and Araya, A., "A Knowledge-Based Framework for Design", *Proc. of the Fifth National Conf. on Artificial Intelligence*, pp. 856-865, vol. 2, August, 1986.

[7] Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design", *IEEE Expert* 1(3), pp. 23-30, Fall, 1986.

[8] Chandrasekaran, B., "Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks", *Proc. of the International Joint Conf. on Artificial Intelligence*, pp. 1183-1192, vol. 2, August, 1987.

[9] Brown, D. C. and Breau, R., "Types of Constraints in Routine Design Problem-Solving", *Proc. of the 1st International Conf. on Applications of Artificial Intelligence in Engineering Problems*, pp. 383-390, vol. 1, April, 1986, Sriram, D. and Adey, R. (eds.), Springer Verlag (1986).

[1] VAX-11 and PDP-11 are trademarks of Digital Equipment Corporation.
[2] DESIGNet is a trademark of Bolt Beranek and Newman Inc.
[3] **Data**pac and **Data**link are trade marks of Bell Canada.

# Qualitative Modeling: Application of a Mechanism for Interpreting Graphical Data

Sheila A. McIlraith
Advanced Technologies Department
Alberta Research Council
Calgary, Alberta

## Abstract

This paper describes the notion of Qualitative Modeling (QM) for interpreting graphical data. The need for qualitative modeling arises from the difficult task of modeling complex systems mathematically, and the resulting method used by experts of modeling observed data by visual interpretation. Given a set of observations, physical attributes of the system may be determined by plotting the data and comparing it qualitatively to representations of known physical models. As with many qualitative reasoning theories, the concept of Qualitative Modeling arises from formalizing and implementing the qualitative reasoning performed by experts in a particular field, in this case, well test interpretation. QM incorporates techniques from numerical analysis, qualitative reasoning, and syntactic pattern recognition to abstract observed data and to fit it qualitatively to a characteristic form. Following an introduction of the material, and presentation of the problem of well test interpretation, discussion deals with the application of syntactic pattern recognition to qualitative representation and the mechanism employed for qualitative modeling.

**Keywords:** qualitative reasoning, blackboard, graph interpretation

## Introduction

Application of artificial intelligence to engineering problem solving has resulted in a generalization and formalism of techniques for reasoning qualitatively about physical systems, broadly referred to as Qualitative Physics. Qualitative reasoning tasks include diagnosis of the cause of aberrant behaviour, prediction of future behaviour, and the capturing of certain commonsense reasoning processes [Bobrows, 1985]. Most qualitative reasoning applications commence with the discrete representation of continuous processes by a set of qualitative descriptors which capture the behaviour of the system. This abstraction of the often complex process is then used to reason qualitatively about the functioning of the system.

Much of the work to date in Qualitative Reasoning has been done in such areas as Qualitative Simulation [Kuipers, 1986], Qualitative Analysis [Williams, 1985] and Qualitative Process Theory [Forbus, 1984], all of which attempt to predict or explain the behaviour of a system based on its qualitative abstraction. This paper describes the notion of Qualitative Modeling (QM) for *visually* interpreting graphical data. The technique involves quantization of a domain to a discrete representation. However, instead of using this representation for predicting or explaining behaviour, it is used as a description for modeling other collected data. Given a set of observations, it is common to summarize the data by *modeling* it, or fitting it to a predefined form. Simple data fitting can often be performed using quantitative techniques such as least squares curve fitting, to identify parameters defining a model. However, in many cases, models are represented by formulae or, when dealing with physical systems, the interaction of several models defined by complex formulae. In cases such as these, interpreters often attempt to visually extract features pertaining to specific models or to almost intuitively identify a model based on the shape of the plotted data. QM attempts to capture this implicit reasoning strategy.

As with many qualitative reasoning theories, the concept of Qualitative Modeling is the result of formalizing and implementing the qualitative reasoning performed by experts in a particular domain [Yip, 1987], in this case, well test interpretation. This paper describes the incorporation of Qualitative Modeling into a prototype expert system for well test interpretation. Following a description of the problem of well test interpretation and the design of a suitable expert system architecture, the author describes techniques for qualitative domain abstraction, and model identification. Further discussion identifies related work, describes the implementation and speculates on extensions to the system.

## The Problem

The purpose of a *well test*, also known as a *pressure*

*transient test*, is to evaluate the state of the ground formation surrounding a well through the analysis and interpretation of the pressure-time data resulting from a test. When performed correctly, a well test and interpretation should provide a physical explanation for the collected data including such qualitative physical features as formation permeability, extent of wellbore damage or stimulation, reservoir pressure, and possibly reservoir boundaries and heterogeneities. Characterization of the well from pressure-time data is a complex and somewhat inexact science; it forms the basis of well test interpretation. Well test interpretation is achieved by fitting the pressure-time data to complex, enhanced models of fluid flow through porous media. For ease of understanding, and brevity, many technical details have been omitted, and only those portions of the interpretation procedure pertinent to this paper are discussed.

Briefly, the well test engineer interprets the well test data by plotting different functions of pressure versus functions of time, and matching them in two different ways against classic pressure-time plots. The goal is to find a suitable model representing the physical features of the formation, and then to find parameter values corresponding to the quantitative measure of some of those features. In the second matching process, performed to determine parameter values, each plot can be visually divided into 3 regions: early-time, middle-time, and late-time. Each of these intervals represents a transition in reservoir fluid activity at an increasing distance from the wellbore, as well as a visual transition.

The first step in the interpretation process is to select a model that accurately reflects the physical characteristics of the formation. In order to identify physical characteristics of the formation, the well test interpreter views a plot of pressure-vs-time, looking for characteristic curve shapes that identify physical features. Figure 1, reproduced from [ERCB, 1979], illustrates a small subset of characteristic shapes recognized by a well test engineer for a gas well. These are simple "ideal" plots. In reality, a formation can have a combination of those features, one for early-time, one for middle-time, and one for late-time. Once the physical features are identified, the mathematical models that reflect those features are selected.

Now the well test engineer has one or more general models to explain the observed well test data. Specific parameter values for those models must be selected. Each model has a family of solutions dependent on realistic values of the variable parameters. These models are very complex, eliminating mathematical curve fitting as a modeling technique. Consequently, to determine the values of the parameters further defining the model, the well test engineer attempts to match the test data with one of a family of *type curves*. Each member of a family of type curves is represented by a curve of plotted data: the solution to the enhanced fluid flow model for particular parameter values. The matching process between the test data and the model generated data is again performed visually.

A further complication to the curve matching process

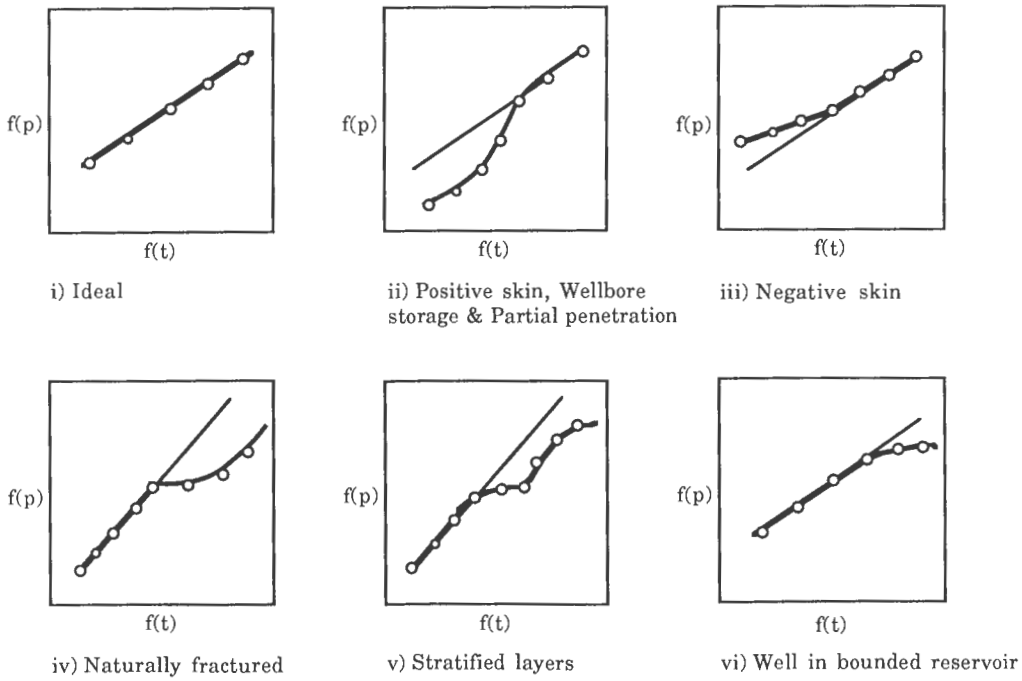



**Figure 1. Characteristic Shapes for Gas Well Buildup Curves**

is that early or late data may be missing from the test sample because of testing techniques or insufficient testing time. To compensate for this, well test engineers perform a type of heuristic curve matching where middle-time data is prioritized and matched followed by early- and late-time data. Heuristics are identifiable by the well test engineer, depending on the model in question.

The interpretation process continues with parameter specific model confirmation both by another similar curve fitting process and by checking the hypothesized formation characteristics against existing formation information. This may include the geology of the region in which the well was drilled, results of core samples, log results, and the treatment history of the well. Conflicting information will not cause the well test engineer to reject a hypothesis but rather to associate a lower certainty with the hypothesis.

The visual interpretation process is currently done with the assistance of plotting programs. Pressure-time plots may be rapidly displayed, and type curve matching performed by displaying the type curves, and shifting the observed data with the arrow keys until a good overlay is found.

**The Architecture**

To address the full problem of well test interpretation, an expert system was proposed. Figure 2 illustrates the design of the expert system *kernel*. The kernel is connected to a user interface and some existing well test software not illustrated in this diagram. The expert system design is a modification of a general blackboard architecture [Erman, Hayes-Roth et al, 1980], proven to be successful in past applications involving diverse sources of knowledge. There are 9 knowledge sources (KSs) in this blackboard system. As illustrated below, they have been partitioned into *Visual Interpretation KSs*, *Auxiliary KSs*, and the *Well Test Interpretation KS*. The Well Test Interpretation KS is an explicit controller. It invokes certain KSs and ultimately evaluates competing hypotheses much the way a well test engineer would. The Auxiliary KSs exist to provide information and expertise when requested. They do not react opportunistically, but rather are invoked by the Well Test Interpretation KS to substantiate or refute an interpretation provided by the Visual Interpretation KSs. Finally, the Visual Interpretation KSs perform the task of interpreting the plotted data. Each of these KSs deals with the data at a different level of abstraction, reacting opportunistically to facts and hypotheses on the blackboard. Qualitative Modeling

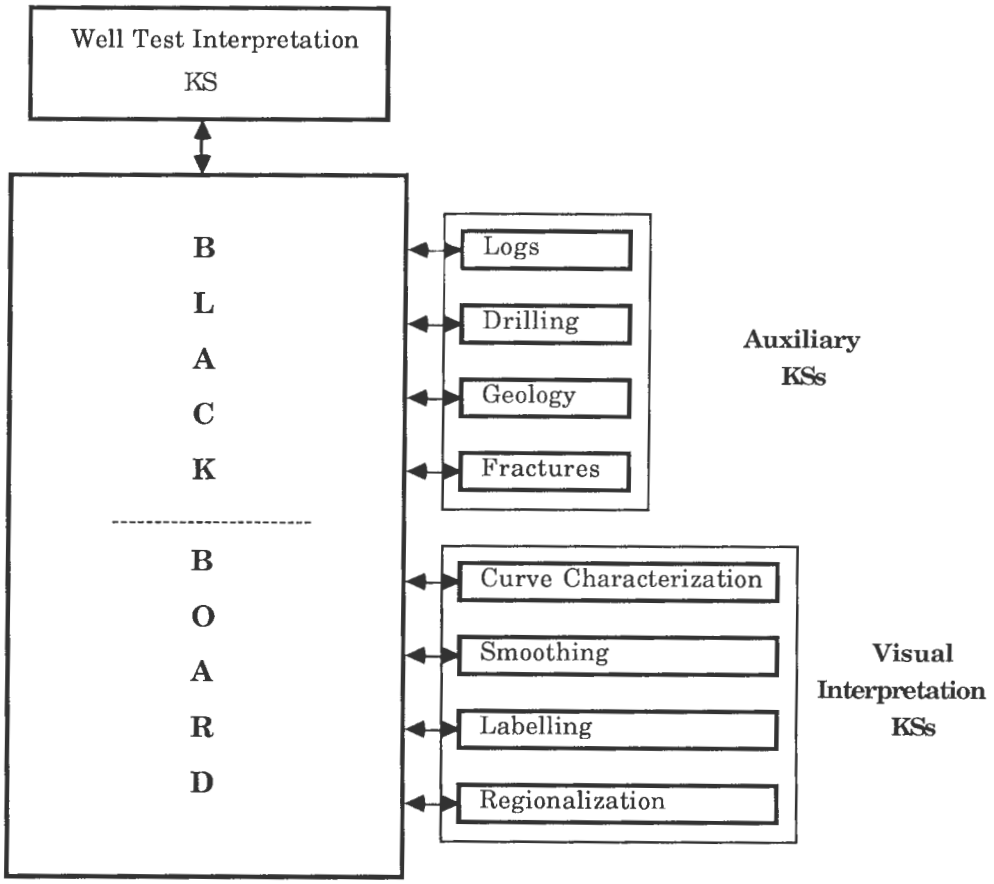



**Figure 2. Expert System Kernel**

is performed in the Visual Interpretation KSs.

## The Approach

In his paper on Qualitative Simulation [Kuipers, 1986], Kuipers restates earlier findings that symbolic manipulation of qualitative descriptions is a plausible model of human expertise. The Visual Interpretation KSs were developed using a combination of techniques from numerical analysis, syntactic pattern recognition, and qualitative reasoning in an attempt to reproduce the qualitative visual interpretation expertise of a well test engineer. Numerical analysis provides a mechanism for approximating and mathematically characterizing graphical data. An extension of concepts from qualitative reasoning and syntactic pattern recognition offers a means of representing and reasoning about graphs at a high level of abstraction.

QM performs several tasks through the Visual Interpretation KSs. 1) It qualitatively represents curve shapes and qualitatively matches observed data against these shapes, or combinations of these shapes. 2) It heuristically matches observed data with model solutions. An initial qualitative match is performed to reduce the number of curves to be matched, and then a quantitative match is performed.

## Model Representation

In order to reason qualitatively about the models and observed data, a qualitative abstraction of the curves into a set of discrete symbolic descriptors must be performed. The quantity space {+,0,-} [Bobrow, 1985] and associated landmarking concepts used by Forbus and others are not amenable to this application because they do not capture the character of the curves. Consequently, syntactic pattern recognition was examined for assistance in defining the discrete symbol set.

Syntactic pattern recognition techniques [Fu, 1977] were developed to represent and reason with visual objects abstracted to high level symbolic descriptors. These descriptors were used to represent the structural relationship between pattern primitives as well as the primitives themselves.

Just as a grammar may be defined for a language, one may be defined for a picture description language. The grammar is composed of a set of easily identifiable pattern primitives or terminals, a set of conceptually high level non-terminals, start symbols, and a set of production rules governing the composition of terminals and non-terminals into legal patterns. The syntactic approach enables the description of many different patterns using easily identifiable building blocks, or pattern primitives and a set of production rules.

The curves in the well test interpretation domain are generally similar in nature to those illustrated in Figure 1. To characterize changes in system behaviour, many qualitative reasoning applications use landmark or transition boundaries. Similarly, QM uses regions to identify transition boundaries or changes in the character of the curve; curve changes are often indicative of behaviour changes. The difference is that the regions, not the transition boundaries, yield the description of the behaviour.

Curves are defined as the concatenation of regions. Each region in turn is a portion of the curve with homogeneous shape; regions are defined in terms of symbols. Proceeding top-down in describing these symbols, the following is an example of some general production rules represented in BNF notation.

```
<curve> ::= <region> <curve>
<curve> ::= <region>
<region> ::= <region-type> <length>
<region-type> ::= <curve descriptor> <curve>
<region-type> ::= <slope grade> <line>
<curve descriptor> ::= <curve rate> <curve shape>
<curve rate> ::= <gradual> | <rapid>
<curve shape> ::= <concave> | <convex>
<slope grade> ::= <gentle> | <moderate> | <steep>
<length> ::= <short> | <normal>
etc.
```

At this point, simple BNF-like production rules are augmented by the use of variables. This is the way in which much of the symbolic smoothing of data is performed. In this pseudo code example, variables are represented by the suffix "-x" on the non-terminals. This notation indicates two symbolically identical instances of a line or a curve.

```
<curve-x> ::= <curve-x> <curve-x>
            | <curve-x> <line-y><short> <curve-x>
            | <line-y><short> <curve-x>
            | <curve-x> <line-y><short>
```

At the next level, symbolic descriptors are augmented by or transformed into quantitative numeric descriptors. Lines and concave/convex curves are determined by analysing the observed data points in sequence looking for points of inflection or changes in the character of the data curve. These line or curve segments are characterized symbolically and quantitatively. <gentle>, <moderate>, <steep>, <gradual>, and <rapid> are determined by a mapping from the quantitative descriptors. <short>, and <normal> are determined by the relative length of the region and the number of data points involved.

To further illustrate the use of this representation scheme, the following production rule describes the shape of Figure 1 (ii), "Positive skin, Wellbore storage & Partial penetration". In this particular curve, the physical features listed depend on the existence of a gradual concave curve during early-time data. The

region that follows is of little importance in determining the existence of these features. Consequently, the following description suffices.

<Figure 1 (ii)> ::= <identifying region> <region>
<identifying region> ::= <gradual> <concave>
<length> <curve>

The description language developed herein is sufficient for many applications including well test interpretation. It may easily be modified or extended for other graphical interpretation problems.

### Quantization of the Domain

Abstraction of the characteristic curve forms, families of model solutions, and observed data, from lists of data points, is performed by the 4 Visual Interpretation KSs. A syntactic pattern recognition approach is taken to both the generation of the representations and to some of the matching. Data that is input to the Visual Interpretation KSs has been mathematically reduced to remove noise caused by measuring instruments and to reduce the number of data points.

Each knowledge source deals with the data at a different level of abstraction. The *Regionalization* KS segments the curve at points where the character of the curve changes, thus partitioning it into preliminary regions. At this point the regions are simply identified in terms of their direction and rate of change; these are the pattern primitives or terminals of the grammar. The *Labeling* KS mathematically characterizes each region with curve fit parameters. The result is a mathematical characterization of each region in terms of shape and length; the concatenation of the regions describes the original curve. Included within the Labeling KS is a mapping from the mathematical characterization into the symbolic pattern language characterization described above. Thus, regions are described at both a mathematical level and a symbolic level.

The *Smoothing* KS performs a heuristic smoothing of the regions based on human-like methods of visual smoothing. A set of production rules is defined using the non-terminals for identifying anomalous regions and amalgamating them with one or both adjacent regions, depending on the shapes of those regions. It does not replace mathematical noise reduction, but rather compensates for over zealous mathematical regionalization. So, for example, a concave curve followed by a small spike (two short lines) and another concave curve would be smoothed into one continuous concave curve. Similarly, part of curve that was mislabeled as a line, thereby breaking the curve into regions of curve-line-curve, would be corrected to a single curve. As soon as the smoothing routine invokes a change in the regionalization of a curve, the Labeling KS reacts to relabel the regions. The Smoothing KS then reacts to the newly asserted labeling. This opportunistic interplay continues until the curve representation is in a steady state. Finally, the *Curve Characterization* KS takes the string of labeled, smoothed regions and, depending upon what has been requested from the Well Test Interpretation KS, either matches it to one of a family of model solutions, or simply identifies features of the well from the characterizing shapes of the regions.

Both the characteristic curve forms and the families of model solutions are static information. They are represented using the aforementioned descriptors and exist permanently in the knowledge base. The families of curves are represented as a hierarchy of frames. A model or family of solutions has particular parameter solutions, which in turn are composed of regions each of which is described both at a mathematical abstraction and a symbolic abstraction. The inheritance property of frames enables a concise representation of pertinent information. The characteristic curve forms make some use of frames but generally are represented as further production rules in the previously defined grammar. Justification for this implementation of the representation is included in the next section.

### Qualitative Data Modeling

Qualitative data modeling is performed by qualitative curve matching procedures. Two distinct types of curve matching are executed in the Curve Characterization KS: identification of characteristic curve forms in observed data, and matching of observed data to one of a family of model solutions.

The identification of characteristic curve forms is performed by extending the production rules of the grammar to encode combinations of high level non-terminal region shapes which imply physical attributes of the well or formation. This was illustrated by the <Figure 1 (ii)> production rule in the section on model representation. If a curve description parses, then a match occurs. Note that more than one production rule may fire, indicating the existence of several physical features. By using this technique the characteristic form of the curve is captured without imposing a strict physical template matching process. Figure 1 illustrates some characteristic curve forms and their associated physical features. In some cases observed data will incorporate a combination of several characteristic shapes, in other cases, part of the well test data may be missing resulting in only a partial shape. These constraints may all be incorporated into the production rules of the grammar which yield a representation more all encompassing than the previously illustrated forms. The result of the characteristic curve matches is some proposed physical attributes of the well or formation; "positive skin, wellbore storage and partial penetration" is one example of the result of a match. The Well Test Interpretation KS takes this asserted fact and

proposes a model or several models that incorporate these physical attributes into their model. The Curve Characterization KS in turn attempts to match the observed data with one of these models.

One family of solutions, representing a specific model with different parameter values often consists of more than ten individual curves. Figure 3 illustrates a subset of a typical family of buildup curves. Attempting to perform a mathematical match of each curve in turn is computationally inefficient. Consequently, a two stage heuristic curve match is performed to reduce computations and to incorporate the rules of thumb a well test engineer uses for matching curves of this nature. The first stage of the curve match is a symbolic matching of the qualitative shape of the regions of the observed data with the shape of the regions of the solutions to the model. Dealing again with the problem of missing data either at the beginning or the end of the test, weights are associated with the existence of different regions of the curve, just as probabilities are associated with stochastic grammars used in syntactic pattern recognition [Fu, 1973]. A set of weighted production rules are incorporated into the grammar for implementation of this match. Using this type of qualitative match, several curves are often found which have the same qualitative shape as the observed data.

These hypothesized matches then proceed to the second stage of curve matching in which a more stringent mathematical matching of data is performed. Heuristics from well test engineering are used for matching the regions of the curve based on the model being used and the quality of the observed data. This results in a prioritized and weighted least squares curve fit being performed. As an example, in many of the models, regions corresponding to middle-time data are matched first since they represent unobstructed formation flow. If the match is successful then remaining regions are matched and weighted in order of importance. Successful matches are spawned as a context or interpretation of the data. A measure of belief is associated with the interpretation based on the degree of match of the least squares technique. The Well Test Interpretation KS reacts to the hypothesized modeling of the data by requesting substantiation from the Auxiliary KSs, or from another type of visual model too similar to warrant individual description. Measures of belief are adjusted accordingly and a final best interpretation is selected.

To summarize, the problem of visual interpretation of graphical data currently being performed by well test engineers has been addressed using concepts from qualitative reasoning, syntactic pattern recognition and numerical analysis. The result is a method for qualitatively modeling observed data in a manner that yields interpretations at the level of an expert.

## Related Work

Throughout this paper, passing comparisons have been made to work in syntactic pattern recognition by K.S. Fu, and to work in qualitative reasoning by Kuipers, Forbus, and De Kleer and Brown. The discussion would not be complete without mentioning work by Elisha Sacks on piecewise linear reasoning [Sacks, 1987] and Kenneth Yip [Yip, 1987] on extracting qualitative dynamics from numerical experiments. Both of these authors employ a phase-space representation for their qualitative reasoning work. Sacks' work on a technique called piecewise linear reasoning attempts to analyse dynamic systems describable by finite sets of ordinary

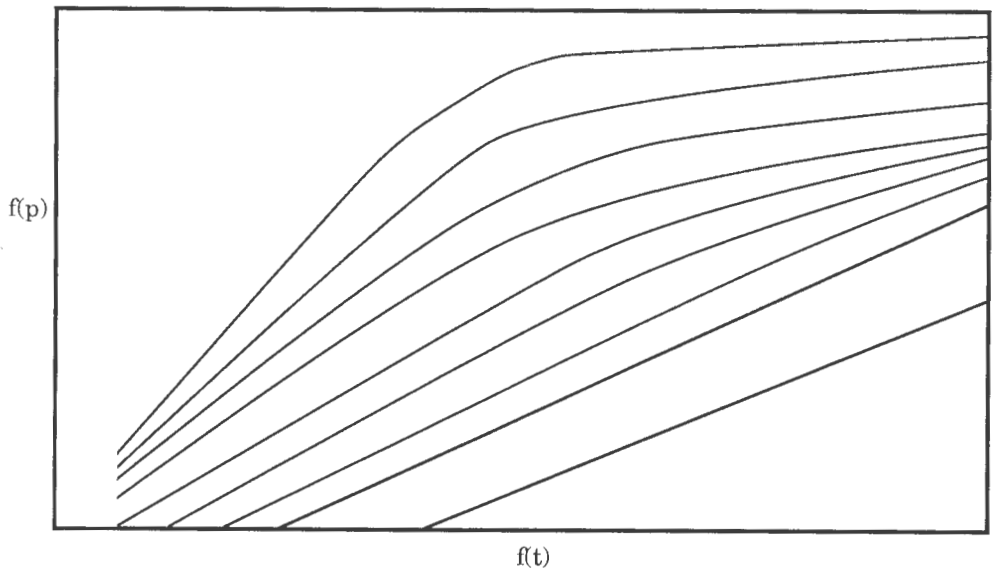



**Figure 3. Sample Family of Curves**

differential equations. Yip's work uses phase-space representation to reason about nonlinear dynamical systems. This work is more theoretically founded than the application of QM, but all three techniques share the goal of qualitatively representing solutions to models of physical systems.

## Implementation and Extensions

A prototype Qualitative Modeling environment was developed for the domain of well test interpretation, as part of the aforementioned expert system design. Performance of the QM with the well test data provided was at the level of the well test engineer. The modeler was implemented in ART and Common LISP on a Symbolics 3670. ART provided a rapid prototyping environment for implementation of much of the blackboard architecture. Furthermore, ART's forward chaining production rule system was ideal for representing the grammar. Representation of models and competing hypotheses was facilitated by the frame representation and viewpoint mechanism. The Symbolics was satisfactory, but is not the ideal environment for the extensive mathematical calculations which might be found in a production system.

QM provides the theory for qualitatively modeling any graphical data that has been noise reduced to a reasonable extent and that has a characteristic form when viewed in 2-D. To date, it has been applied to the domain of well test interpretation and will presently function for any other graphical data definable by its descriptors. The representation symbol set may easily be extended to provide for more complex graphical data without interfering with the existing reasoning mechanism. Potential applications of the technique include other domains within engineering problem solving, and in the domain of statistical analysis and economic forecasting. In the future, the author would like to extend the implementation and attempt to further formalize some of the concepts identified in this paper. Work in grammatical inference and learning might also be incorporated into this application.

## Concluding Remarks

In this paper I have described a mechanism for *visual* interpretation of graphical data: Qualitative Modeling (QM). It incorporates techniques from numerical analysis, qualitative reasoning and syntactic pattern recognition to abstract observed data and to fit it qualitatively to a characteristic form. The need for qualitative modeling results from the difficult task of modeling complex systems mathematically. This is of particular importance when dealing with real physical systems which often do not reflect a single mathematical model. The work presented herein describes the notion of syntactic pattern recognition grammars for quantization of the domain to a discrete and usable symbol set. This symbol set is then employed to qualitatively model the data. The techniques described in this paper are applicable to a wide range of graph interpretation problems. Incorporation of domain specific knowledge to substantiate or refute hypothesized interpretations strengthens the modeling environment.

## References


[Bobrow, 1985] Daniel G. Bobrow. Qualitative Reasoning about Physical Systems: An Introduction. In Daniel G. Bobrow, editor, *Qualitative Reasoning about Physical Systems*, pages 1-5, MIT Press, 1985

[De Kleer and Brown, 1985] Johan de Kleer and John Seely Brown. A Qualitative Physics Based on Confluences. In Daniel G. Bobrow, editor, *Qualitative Reasoning about Physical Systems*, pages 7-83, MIT Press, 1985

[ERCB, 1979] Energy Resources Conservation Board (ERCB). Guide 3: GAS WELL TESTING Theory and Practice, 4th edition, 1979 (metric)

[Erman, Hayes-Roth et al, 1980] D.L. Erman, F. Hayes-Roth, V.R. Lesser and D.Raj. Reddy. The HEARSAY-II speech understanding system: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Survey*, 12: 213-253, 1980

[Forbus, 1984] Kenneth D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24: 85-168, 1984

[Fu, 1973] K.S. Fu. Stochastic Languages for Picture Analysis. *Computer Graphics and Image Processing*, 2: 433-453, 1973

[Fu, 1977] K.S. Fu. Introduction to Syntactic Pattern Recognition. In K.S. Fu, editor, *Syntactic Pattern Recognition, Applications*, Springer-Verlag, 1977

[Kuipers, 1986] Benjamin Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29: 289-338, 1986

[Sacks, 1987] Elisha Sacks. Piecewise Linear Reasoning. In *Proceeding AAAI-87 Sixth National Conference of Artificial Intelligence,* pages 655-664, 1987

[Williams, 1985] Brian C. Williams. Qualitative Analysis of MOS Circuits. In Daniel G. Bobrow, editor, *Qualitative Reasoning about Physical Systems*, pages 281-346, MIT Press, 1985

[Yip, 1987] Kenneth Man-kam Yip. Extracting Qualitative Dynamics from Numerical Experiments. In *Proceeding AAAI-87 Sixth National Conference of Artificial Intelligence,* pages 665-670, 1987

# Exploiting fine-grained parallelism in Production Systems

**Bruce T. Smith**
**Department of Computer Science**
**University of North Carolina at Chapel Hill, NC 27514**

**David Middleton**
**Institute for Computer Applications**
**in Science and Engineering,**
**NASA Langley Research Center, VA 23665**

## Abstract

*Fine-grained parallelism is attractive since it offers large increases in execution speed if it can be effectively exploited. However, recent studies in implementing OPS5 have shown that straightforward parallel implementations may perform little better than clever sequential ones. This has led to conjectures that no useful fine-grained parallelism exists in the OPS5 language. The large majority of computation in an OPS5 program lies in the matching phase which searches for patterns in a data-base of facts. The RETE algorithm enables sequential implementations of OPS5 to reduce the costs of matching by storing partial matches from previous phases. We present a matching algorithm which exploits the fine-grained parallelism available within RETE itself, by using abstract node processors to emulate the RETE network directly. These node processors operate concurrently and internally exploit low-level parallelism, such as associative search. Each abstract node processor can be implemented by a group of fine-grained programmable processors such as is provided by the FFP machine.*

## 1 Introduction

The speedup to be gained by using parallelism in an application depends on the number of processors that can be simultaneously applied. The desire for very large increases in speed in expert systems in general and the OPS5 language in particular leads to the search for fine-grained parallelism. Apparently though, straightforward fine-grained parallel implementations of OPS5 perform little better than some sequential implementations. These sequential implementations use the RETE algorithm for the pattern matching phase, which consumes the vast bulk of processing time. The RETE algorithm reuses partial matches from prior matching phases instead of recomputing them, enabling the majority of OPS5 programs, those in which the working memory changes very slowly, to perform the matching process as rapidly as those implementations exploiting fine-grained parallelism [Gupta 84, Stolfo *et al.* 84, Quinlan 85, Forgy *et al.* 86]. This led to conjectures that parallelism was only of limited use in accelerating OPS5 programs.

The RETE algorithm itself, however, contains many opportunities for exploiting fine-grained parallelism, such as associative searches, and it is towards exploiting this parallelism that we propose special purpose node processors for implementing the RETE algorithm. These abstract node processors may in turn be implemented using the virtual machines supported by the FFP Machine, a fine-grained MIMD computer originally designed for executing Backus's Functional Programming languages.

Section 2 describes the OPS5 language and the matching process which is the bottleneck in its operation. The RETE algorithm, one implementation of this process, involves compiling the patterns of an OPS5 program into a discrimination network, a data-flow graph of simple comparisons. During execution, tokens percolate through the RETE network, being combined into tuples which finally match the patterns. Section 3 describes the virtual machines supported by the FFP machine, independent variably-sized groups of fine-grain processors created during execution. Section 4 describes abstract node processors which support the execution of the RETE network graph and which can be implemented using virtual machines. Parallelism is extracted from the RETE algorithm by having many node processors executing concurrently, each employing internal parallelism. Section 5 summarizes the possibilities of this approach and the factors which might affect the usefulness of the parallelism.

## 2 OPS5

OPS5 is a production system language for building expert systems. Our descriptions of OPS5 and the RETE algorithm relate to the network of node processors described in Section

This work was supported in part by NSF Grant No. MIP-8702277, ONR Grant N00014-86-K-0680 and NASA Contract No. NAS1-18107.

```
(p match-boxes
    (box ^width <x> ^height < 12 )
    (box ^width 10 ^height < <x> )
    --> (remove 1))
```

a) Example rule

```
1:  (box ^name Crate1 ^width 10 ^height 15)
2:  (box ^name Crate2 ^width 10 ^height 10)
3:  (box ^name Trunk ^width 20 ^height 10)
```

b) Example working memory elements

**Figure 1.** Fragment of an OPS5 program with data.

4, rather than traditional descriptions. Conventional, more comprehensive, descriptions of OPS5 and RETE exist elsewhere [Brownston 85, Forgy *et al.* 87].

An OPS5 program is a set of rules which manipulate elements of a working memory. The left hand side of each rule is a pattern which can be matched by a tuple of working memory elements; actions in the rule's right hand side create, delete or modify working memory elements according to matches that are found. Figure 1 shows a sample OPS5 rule and a working memory with three elements, each containing a unique identifier, the element's type and a list of attribute-value pairs (the attributes having been defined in the type declarations). Each working memory element represents a specific box. The rule `match-boxes` has two condition elements in its left hand side and one action in its right hand side. The first condition element matches any working memory element whose `height` is less than 12 and assigns the value of its `width` to the variable `x`. The second condition element matches any working memory element whose `width` is 10 and whose `height` is less than the value of `x` (for each working memory element matching the first condition element). This entails a comparison between pairs of working memory elements. Finally, the action of `match-boxes` would remove the element matching the first condition element in the chosen instantiation from the working memory.

The RETE algorithm is an incremental method for performing the matching process. The left hand sides of rules are compiled into a discrimination network, a directed acyclic data-flow graph whose nodes perform tests specified in various condition elements. The tokens that pass through this network represent tuples of working memory elements that simultaneously match part of the left hand side of some rule. When a working memory element is created, one or more new tokens with that working memory element as a singleton enter the "top" of the network. There is a separate output from the network for each rule; tokens leaving an output represent tuples of working memory elements completely matching the rule's left hand side. The conflict set comprises all the tokens that have come out the "bottom" of the RETE net, each labeled with its particular exit point. (These labeled tokens are complete rule/tuple instantiations).

Notice that the restrictions in a condition element can be divided into two types: those that involve a single working memory element and those that compare one working memory element with others. These restrictions are implemented in the RETE net by *one-input* and *two-input* nodes, respectively. The leaves of the RETE network, one for each condition element, are one-input nodes which perform those tests that apply to a single working memory element. (That is, a one-input node performs all tests not involving variables set in other condition elements). When a new working memory element is created, a singleton token is distributed to each of these leaf nodes. The nodes act as filters, passing to the interior of the network only those tokens whose working memory elements satisfy their condition elements' restrictions (including its type).

The two-input nodes in the interior of the tree perform tests that involve more than one working memory element. Input tokens arriving from the two subtrees represent sequences of working memory elements that could become part of an instantiation. Two-input nodes store these tokens in a *token memory* to participate in comparisons with future arriving tokens. (This memory is often divided into a *left-memory* and *right-memory* corresponding to the two paths along which the tokens arrive). The node only makes comparisons when a new token arrives: if the new token arrives from the left subtree, it is compared against each token stored in the right-memory, and vice versa. When the node finds matching left and right tokens, that is, ones that satisfy the constraints, it produces an output token that includes the working memory elements from both input tokens.

Keeping tokens in this way saves the matching work done in creating them, from being repeated in later execution cycles. For most OPS5 programs, fewer than three working memory elements change (on average) between rule firings, even in a working memory with many thousands of elements [Gupta *et al.* 83]. It is by storing partial matches in the node memories that the RETE matching algorithm derives speed from this stability.

Figure 2 shows a RETE network for the sample rule from Figure 1. The numbers along the left and right input arcs to the two-input node

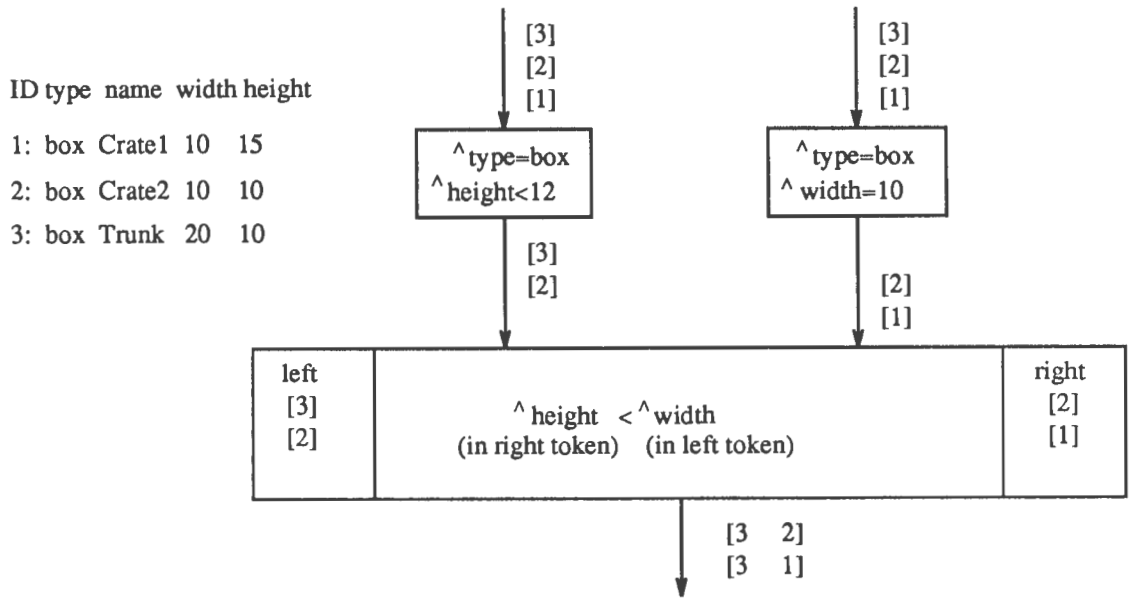



**Figure 2.** RETE network for example fragment.

are the identifiers of the (singleton) tokens arriving at the node (to be stored in the left-memory and right-memory, respectively). The pairs on the output arc from the two-input node are tokens containing full matches to the left hand side of the rule `match-boxes`.

When a working memory element is removed from the working memory (as in our sample rule's action), all tokens that represent partial matches including that working memory element must also be removed. This can be accomplished by dropping a *killer token* into the RETE network. As killer tokens propagate through the network, they delete the appropriate tuples from left and right memories. If we were to follow the progress of the RETE network in Figure 2 after choosing instantiation (`match-boxes 3 2`), the action (`remove 1`) would generate a killer token for working memory element 3. It would remove tuples containing working memory element 3 from the left and right memories of the two-input node and from the conflict set.

In OPS5, a rule's left hand side may contain *negated condition elements* which disable any instantiation for that rule if a matching working memory element exists. It is sufficient to note that removing a working memory element may cause new instantiations to be created when that working memory element matched a negated condition element. Thus, nodes in the RETE network may need to create new tokens on receiving a killer token, and to create killer tokens on receiving an ordinary token.

As with other implementations of RETE, the parallel one described here relies on an extensive compilation phase in which the rules are analyzed, optimisations are made and corresponding networks are constructed. Common compile-time operations include the following: extra space is allocated to each working memory element to hold a unique identifier; attribute names and variables are translated into numeric offsets specifying the position of the designated value in a token.

Some optimisations that are effective for sequential implementations, such as only sending new tokens to particular one-input nodes and distinguishing between left and right memories, may or may not be of any use in this implementation of RETE. In particular, we do not discuss the sharing of RETE subtrees which greatly improves sequential implementations. RETE allows the order of evaluation of condition elements and the tests within them, to be swapped.

This system does perform further operations during compile-time analysis, often creating new attributes at the same time. The `left/right` flag described below is one such, and in the case of tests involving expressions with several attribute values, a new attribute would be added whose value represents the suitable combination. For example, a rule describing airline luggage size limitations might require that the sum of three attribute values be less than some constant. The compiler, on encountering such a condition element, would allocate an extra attribute field for

the working memory element, and cause the appropriate node processor to store in this field the sum of the other three attribute values, before propagating the new token.

## 3 Virtual Machines in the FFP machine

The FFP Machine is a fine-grained MIMD parallel computer [Magó *et al.* 84, Magó 85, Magó *et al.* 87], which has been designed to execute the functional programming languages of John Backus [Backus 78]. It contains a very large number of simple programmable cells, each having a few hundred bytes of local storage for instructions and data. These cells are organized as a linear array, the *L array* of *L cells*, in which programs and data, represented as strings of symbols, reside and are manipulated. L cells communicate through a tree-structured network of message processors.

The marked difference between the FFP Machine and similarly fine-grained machines lies in the concept of *partitioning*. The FFP Machine can be divided, at run-time, into arbitrary disjoint groups of contiguous cells, called *virtual machines*. The division is done rapidly, without preparation, according to syntactic structure in the strings of symbols in the L array. The cells of each virtual machine can communicate among themselves, but they are disconnected from, and unaffected by, communication among the cells in other groups. Furthermore, the sizes of the virtual machines are independent, being determined dynamically by the needs of the different computations that arise.

Some noteworthy aspects of the partitioning can be seen in Figure 3, which shows part of a physical machine partitioned to support several virtual machines (running FP programs, in this case). Virtual machines are not aligned with the physical tree structure in the FFP Machine; this avoids wasting resources through fragmentation. Empty cells may be interspersed among the symbols of the programs and data; such cells do not affect the virtual machines formed around groups of symbols and they aid storage management (to be described below). The virtual machines are defined syntactically by the presence of innermost pairs of parentheses (which derives from the innermost reduction rule used in FP languages [Backus 78]). In the example shown, the summation of the inner product does not become innermost, and so executable, until all of the individual products have been computed. This syntactic definition provides a natural, inexpensive scheduling mechanism; tasks can be scheduled by altering the arrangement of parentheses residing in the L array.

Each virtual machine has exclusive use of a set of message processors (proportional to the number of its L cells). These message processors provide each virtual machine with a tightly-coupled synchronised network which can perform various simple operations such as sorting and counting. Each message processor merges two sorted streams of messages according to one or two keys contained in the messages and, in the case of collisions, combines messages according to an included operator. The time needed to communicate one set of messages within a virtual machine is linearly proportional to the number of messages sent and logarithmically proportional to the number of cells in the virtual machine.

A virtual machine also has the remarkable ability to change in size during its operation, in order to acquire empty L cells to hold more data. To enable such storage allocation, the contents of L cells in the entire L array are shifted horizontally by various amounts, all the while maintaining their linear ordering, in such a way that empty L cells are absorbed, and gaps of the appropriate size are created among the L cells that are to support the virtual machine making the request. Following the subsequent partitioning operation in which new sets of message processors are allocated to the newly positioned virtual machines, the virtual machine spanning the gap will be able to store data in these newly acquired cells. The time required for this *storage management* operation is linearly proportional to the largest distance traveled by (the contents of) any L cell. Interleaved empty L cells reduce the interaction between separate requests and so improve behavior. This situation occurs when sufficiently many empty cells are interspersed among active cells so that separate storage requests do not interact and so aggregate. Empty cells are created when virtual machines finish their operation and need fewer cells to represent the result than they contain; the remainder are erased and so become ready for any subsequent storage allocation.

The single restriction on virtual machines is that they consist of a *contiguous set* of the L cells. This is necessary so that the tree network can be partitioned into non-interfering subnetworks. More importantly, the specification and creation of partitions within a tree is much simpler (and hence cheaper in time and hardware) than it would be for richer network structures. This restriction surfaces as a constraint on the programmer: to avoid communication contention, concurrent processes must be mapped onto the L array in a way that preserves the locality in the computation.

Partitioning the physical machine into virtual machines is of negligible cost: it involves passing control messages of three bits from nodes to their parents, performing a few logical operations and setting three switches that configure internal communication channels. (Each internal node may need to support up to three distinct virtual machines and is designed to do so simultaneously).

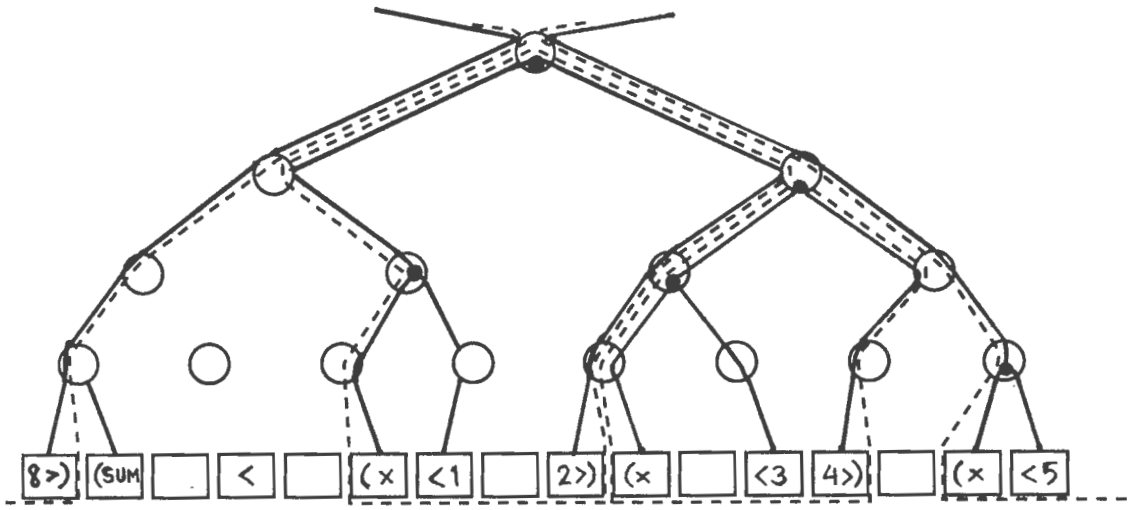



**Figure 3.** FFP Machine partitioned to support separate virtual machines

Virtual machines are rarely balanced, however their depth is typically only a few levels deeper than necessary for the number of cells they contain. Virtual machines suffer no alignment constraints. A virtual machine may begin at any point in the linear array of cells and no fragmentation arises from its having an arbitrary length. Thus, for example, a tree with a thousand cells can support two virtual machines, one containing four hundred cells and the other six hundred.

This flexibility in size and placement enables the processing power of a virtual machine to correspond more precisely to the needs of a particular computation, specifically, the amount of data being manipulated by a particular function. This flexibility should overcome some of the difficulties cited for DADO implementations of OPS5 [Gupta 84]: The number of DADO processors assigned to each rule corresponded to a physical subtree in the machine (thus being a power of two), this size was the same for all rules and was determined before execution began. The position and size of individual virtual machines as described here are independent of each other (except for the single overall limit of the number of L cells), and are determined entirely *during execution* according to their individual requirements.

The principal costs in machine operation lie in the communication and storage management phases. The costs of partitioning, loading programs into the L cells and executing local instructions in the cells are assumed to be negligible.

## 4 Virtual machines supporting the RETE network

The RETE network described above is implemented in a straightforward way by a corresponding network of abstract node processors. The node processors and their communication with each other are supported by a series of different sets of virtual machines created from the same L cells by partitioning. By using a series of virtual machine groupings, as demonstrated in Figure 5a, different stages of computation can be optimised individually: global communication can be supported by a few large groupings while concurrent operations can exploit many small independent groupings [Middleton *et al.* 86]. The cells supporting a given node processor can, when necessary, be further divided into several virtual machines to support distributed operations such as associative search.

### The Abstract Node Processors.

In keeping with the "logic in memory" philosophy of fine-grained systems, the node processors are described according to their data structures, the node's processing power being distributed evenly among the data. For brevity, only the simple two-input node in the RETE network is described; the operation of other nodes can be inferred from it. This two-input node, shown in Figure 4, checks every pairing of left tokens with right tokens, to see whether they satisfy the tests specified in some rule's left hand side. It does this incrementally, comparing each newly arrived token against all the tokens that have already arrived from the other side, and saving that token for comparison with subsequent new tokens.

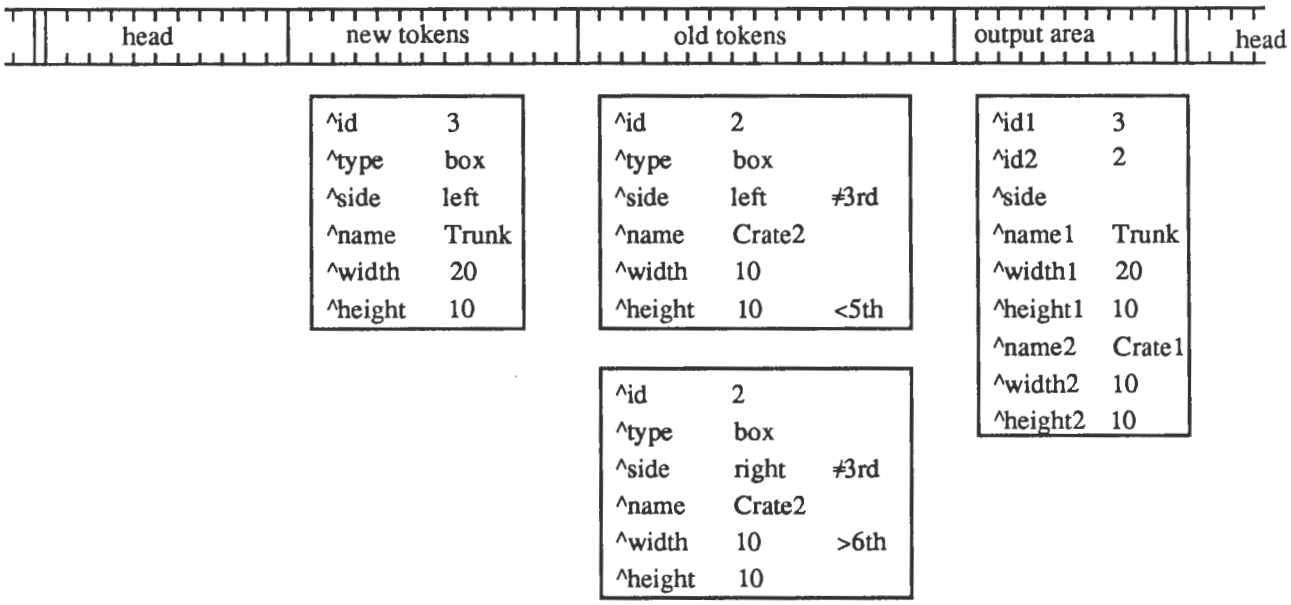



**Figure 4.** Data storage for the associative two-input node processor.

The two-input node processor contains four structures: the *head* holding information describing the particular tests of the corresponding node in the RETE network; the list of *new tokens*, ones which have recently arrived at this node; the list of *old tokens*, which corresponds to the combined left and right memories of a RETE node; and the *output area* for holding the tokens created by the matching process. Tokens are stored simply as lists of attribute values; there are no pointers being used to build more complex structures.

The node processor matches one new token against all the old tokens using five steps (for which approximate estimates of the time spent are given).

(1) A new token is selected from the new token list, taking a small constant time, $c_{select}$, independent of the number of new tokens or their sizes.

(2) The new token is broadcast to the old tokens as a sequence of attribute values. The L cells holding the old tokens, having local copies of the tests relevant to their particular attribute values, can extract and compare attribute values from this stream of messages with their own attribute values. This takes a time proportional to the size of the new token, $O(token\_size)$, plus the depth of the communication network, which is approximately $O(lg(node\_memory\_size))$.

(3) Each group of L cells holding one old token determines if all the tests were satisfied. (For this step, the L cells of the node processor are further partitioned in order to avoid communication contention between cells holding different tokens). This take a time approximately proportional to the size of a token (ignoring size differences between left tokens and right tokens), $O(lg(token\_size))$.

(4) For each old token in which all the tests succeeded, an output token is created combining that token with the new token. Assuming there are $S$ successful matches, this takes $O(S \times token\_size)$ for storage allocation and $O(S \times token\_size + token\_size)$ for communicating the old tokens and new one respectively.

(5) The new token is converted to an old token by broadcasting the tests stored in the head to it. Each attribute in the new token saves the tests that apply to it in a form suitable for its later use. This communication operation takes $O(token\_size + lg(node\_memory\_size))$.

Combining the five approximate costs for these steps, the time for the matching operation is dominated by the term $O(S \times token\_size + token\_size)$. The size of the node, predominantly the L cells holding the new and old tokens, appears as a logarithmic factor which is dominated

by the other terms. Thus the time taken by a node is relatively independent of the number of tokens in its memory.

Each test consists of two selectors and a predicate. The selectors specify attribute values within the left and right tokens as arguments to the predicate. Since a token is a list of working memory elements each of which is a list of attribute values, a selector contains a pair of indices, although compilation might flatten tokens into simple lists of attributes and thus change the selector to a single value.

The left and right memories of a RETE node having been merged into the old token list, a `left/right` attribute is added to each token to distinguish which of the two RETE subtrees sent it. Two tokens may match if their `left/right` attributes differ, this constraint being imposed in the same way as the explicit tests derived from left hand sides. Old tokens contain, as well as a list of attributes, the set of tests to be performed.

Figure 4 shows the node processor for the two-input node in Figure 2, after it has received and saved two tokens containing working memory element 2 in its old token list: one received from its left subtree and the other from its right subtree. On receiving the token containing element 3 from the left subtree, it compares it with all old tokens, trying to satisfy two constraints: first, that the `left/right` attribute in the new and the old token differ, and second, that the `height` attribute in the right-hand token is less than the `width` attribute in the left-hand one.

If a test fails, the L cell sets a *veto* flag. In a subsequent communication stage, the cells of the node processor are further partitioned into smaller virtual machines, one for each old token. The machines containing each old token can determine, in parallel, whether their token is consistent with the newly-arrived token, by the absence of any veto flags.

At this point, the new result tokens (possibly zero or many) are created in the output region of the node processor. The number of matches is counted and the number of cells needed to hold the results is computed and allocated in this region. The cells holding the new token and those holding old tokens which match transmit their contents to the newly allocated cells which receive and store these symbols in such a way that the output tokens result. Simple manipulation of values by the cells in the output area occurs at this point. For example, the `left/right` attribute in the created token would be set to indicate whether this node is the left or right child of the node receiving these resulting tokens, as determined during rule compilation.

**Communication between Node Processors.**

Newly formed tokens are now transferred from the output areas of node processors to the input areas of the parents (in the RETE network) of those nodes. The grouping of L cells into node processors is suspended, and the L cells are re-grouped to support transferring tokens between node processors. Partitioning the L cells into many groups at this stage avoids contention between independent transfers (or at least minimizes it, depending on the mapping of the RETE network).

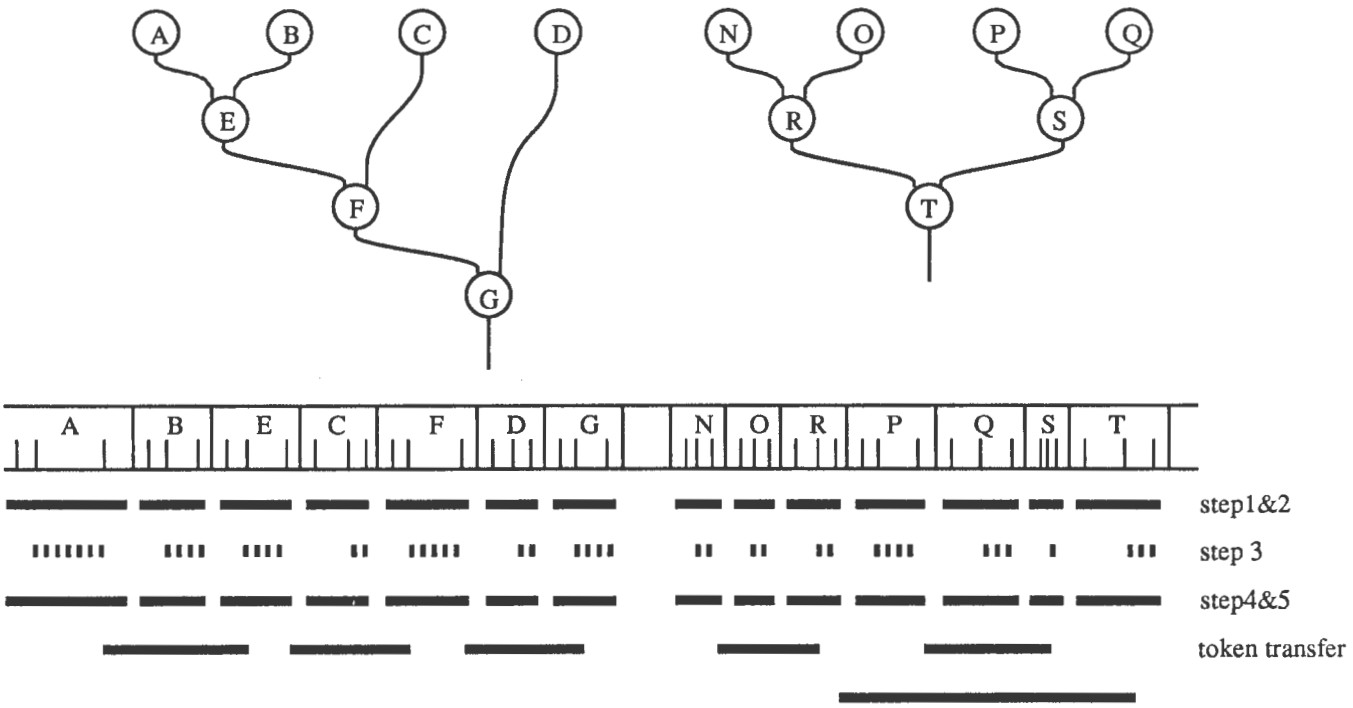
Figure 5 shows two RETE networks containing three two-input nodes, and a corresponding embedding of the network in the L array. For both networks, the sequence of cell groupings is shown; the first three groupings correspond to a single matching cycle and the subsequent groupings support the communication between nodes.

Figure 5a shows a conventional, skew tree of nodes. The depth of such a tree, and consequently the pipeline delay through a sequence of node processors, is equal to the number of condition elements in the corresponding rule. However, such a tree can be embedded in the linear L array so as to allow all communication arcs in the RETE network to be active simultaneously. Figure 5b shows a balanced tree of nodes which might be more suitable for parallel processing since the pipelined delay is logarithmic in the number of condition elements. However, such a tree can no longer be efficiently embedded in the L array. Several groupings of L cells are necessary to support tokens traversing RETE network arcs: as many as the depth of the tree.

L cells are allocated in the parent node's new token list and the output tokens in the children nodes are broadcast, after which the cells in the output regions can be reclaimed. Since this communication and storage allocation is expensive and is performed twice for all created tokens, a reasonable optimisation is to discard the output area from nodes, and create tokens directly in the new token lists in their parent nodes.

**Token deletion.**

Deleting working memory elements and tokens from the network has so far been ignored. There are two ways that deletion may occur (other than the natural firing of rules which removes the winning token from the conflict set). An action may remove (or modify) a working memory element, in which case all the tokens containing that working memory element must be removed, or an action may create a working memory element which, by matching a negated condition element in the left hand side of a rule, prohibits any combination of working memory elements (which might previously have provided a legal match for that left hand side) from entering the conflict set. As with the conventional RETE algorithm, working memory elements in this system have unique identifiers which are in fact timestamps of their creation; these are used for deletions. Lack of space precludes a detailed description of how tokens are deleted, however




a) Skew RETE tree needs one communication cycle b) Balanced RETE tree needs several cycles

**Figure 5.** Virtual machines to support RETE network of node processors.

it should be noted that various associative techniques similar to those used above allow such tokens to be found and deleted in parallel. The L cells holding these elements and tokens immediately become available for storage management.

## 5 Conclusions

Recent debate has suggested that the Rete algorithm, implemented with limited if any parallelism, is the best way to support pattern matching in OPS5, and, by implication, probably in other production-system-style languages as well. The matching process described here is intended to show that a large amount of *useful* fine-grained parallelism is available in the matching process, specifically in combination with the advantages of the RETE algorithm itself.

This matching process exploits parallelism in several ways. First, all nodes in the RETE network can run in parallel if they contain a token in their input queue. The pipelining aspect of the operation, that each node passes on matches involving one new token before starting to match the next new token, is intended to exploit this parallelism between nodes. Other parallel implementations of RETE also exploit this parallelism using some dozens of processors sharing a common memory to contain the working memory elements. Communication between nodes in this scheme, occurring between very tightly coupled processors, should be no slower than message passing between powerful processors, which often requires significant operating system intervention. Since the tokens passing between nodes are much larger (they contain complete working memory elements rather than just pointers to a single shared memory), this scheme uses copying instead of sharing to avoid contention for access to such a memory.

This scheme exploits parallelism within each node, an opportunity which cannot be taken by approaches using coarse grain processors. Matching a given new token is relatively independent of the number of old tokens stored in the node, as shown in Section 4: matches can be performed in a natural associative fashion since a node's processing power increases directly with the size of its token memory. All the nodes in the RETE network can match one of their new tokens in basically the same time; the single principal difference arises with different success rates in different nodes. This counters one of the arguments against fine-grained implementations: that the significant variance in node processing

times, due to the different sizes of token memories, justifies using a few powerful processors.

Working memory elements can be deleted from the entire RETE network in a single operation. Ordinary and "killer" tokens can be deleted from each node memory in a single associative operation in that node, while the propagation of effects from such an operation will depend on the depth of the RETE network.

It is unclear what factors limit the effectiveness of this fine-grained approach to pattern matching in OPS5. Storage allocation takes linear time in the number of tokens created (certainly, for each node, and in the worst case, for all the nodes together) and the constants of proportionality are greater than those for conventional memory allocation. This system implicitly stores tokens with one attribute value per L cell, which may require more processors than is practical. Possible optimisations include packing several attributes in a single cell, propagating only those attributes used later in the RETE network and sharing common condition elements, as sequential RETE algorithms commonly do.

There are two factors which prevent this system from being amenable to more detailed analysis. Most importantly, the behavior of an OPS5 program with regards to these factors is very heavily dependent on the nature of the particular rules and input data. Secondly, the behavior of storage management in the FFP Machine depends heavily on the particular distribution of storage requests in relation to the empty cells that have arisen during the computation. Therefore, the next step is to determine through simulation to what extent these constraints limit the speed improvements provided by the concurrency.

### Acknowledgements

We are grateful to Dave Nicol and Sherry Tomboulian for comments they made following an early presentation of this work.

### References

J. Backus, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs", *Communications of the ACM*, Volume 21 No. 8, pp. 613-641, August 1978.

L. Brownston, R. Farrell, E. Kant and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley, 1985.

C. L. Forgy and A. Gupta, "Preliminary Architecture of the CMU Production System Machine", *Nineteenth Hawaii International Conference on System Sciences*, pp. 194-200, Honolulu, January, 1986.

C. L. Forgy and S. J. Shepard, "Rete: A fast match algorithm", *AI Expert*, pp. 34-40, January 1987.

G. Magó and D. Middleton, "The FFP Machine - A Progress Report", *Proceedings of the International Workshop on High-Level Computer Architecture*, pp. 5.13-5.25, May 1984, Los Angeles, California. Reprinted in *IEEE Tutorial on Computer Architecture* by D. Gajski, V. Milutinović, H. Siegel and B. Furht, and in the *IEEE Selected Reprints on Dataflow and Reduction Architectures* by S. Thakkar.

G. Magó and D.F. Stanat, "The FFP Machine", UNC-CH Computer Science Technical Report 87-014. To appear in *Topics in High-Level Language Computer Architecture*, by V. Milutinović, Computer Science Press.

D. Middleton and B. T. Smith, "FFP Machine support for Language Extensions", *Nineteenth Hawaii International Conference on System Sciences*, pp. 59-66, Honolulu, January, 1986.

J. Quinlan, "A Comparative Analysis of Computer Architectures for Production System Machines", CMU Technical Report CMU-CS-85-178.

S.J. Stolfo and D.P. Miranker, "DADO: A Parallel Processor for Expert Systems", *Proceedings of the 1984 International Conference on Parallel Processing*, pp. 74-82, August 1984.

A. Gupta and C. L. Forgy, "Measurements on Production Systems" CMU Technical Report CMU-CS-83-167.

A. Gupta, "Implementing OPS5 Production Systems on DADO", *Proceedings of the 1984 International Conference on Parallel Processing*, pp. 83-91, August 1984.

G. Magó, "Making Parallel Computations Simple: The FFP Machine", *Proceedings of COMPCON '85*, pp. 424-428, 1985. Reprinted in *Computers for Artificial Intelligence Applications*, IEEE Computer Society, 1986.

# An Expert Advisor for Fourth Generation Software

Douglas Skuce

*Department of Computer Science,*
*University of Ottawa*
*Ottawa, Canada*
*doug@uotcsi2.bitnet*

## Abstract

A prototype "advisor" system which answers typical "how" and "why" questions about a fourth generation report writer has been developed using commercial expert system tools. The system features natural language input, both forward and backward reasoning, a "causal" reasoner specialized for reasoning about user's code, and a code synthesizing component. KEE, ART, and Prolog have been used, and we provide a discussion of their relative merits. We make a distinction between "shallow" and "deep" knowledge representations; both approaches were used in the project. We conclude that such tools can be used directly (the shallow approach) for a useful but not penetrating coverage of typical questions, but that even this would require a number of person-years to develop a reasonably useful system.

**Keywords**: *expert system; human interface; question answering*

## Introduction

This paper describes a project undertaken by the University of Ottawa in collaboration with Cognos, Inc., one of Canada's leading software companies. The Advisor Project was begun in the fall of 1985 to explore the potential of incorporating state-of-the-art expert system technology into certain of Cognos' future fourth generation languages (4GL).The aim of the Advisor project was to create a prototype software advisor or assistant for the user of a typical 4GL[1]. For the prototype we chose QUIZ, Cognos' fourth generation report writer, as the experimental subject, because it is a mature and well understood product, it is typical of such 4GL languages, and there is a large body of users with a well documented history of problems encountered using the product. This documentation was used extensively as a guide to the kinds of questions the system ought to be able to answer. A secondary goal was to explore a number of different approaches to implementing such a system that are currently available. These may be partitioned in several ways: base language (e.g. Lisp or Prolog) vs higher-level tool (e.g. KEE [KEE 86]); natural language (nl) oriented vs rudimentary natural language ability; shallow vs deep semantic representation. We have carefully considered all of these alternatives, and have experimented with most of them. We offer some conclusions from these comparisons.

[1]Skuce [87], Skuce and Tauzovich [87], Tauzovich [87], Skuce, Stanley and Tauzovich [88], Szpakowicz [87], Constant et al [86], and Delisle [87] give further details.

QUIZ is the report writer of POWERHOUSE, Cognos' application development system [QUIZ 85]. A QUIZ program is a declarative specification of the desired report. It starts with an ACCESS statement which declares the files to be used and the linkages between them. Other major statement types include the REPORT statement which defines what is to be reported, and the SELECT statement which specifies the data selection criteria. A GO statement indicates that the specification is complete, and that execution can begin. A report-writing module then produces the actual report.

### Approach

We consulted the records of Cognos' telesupport group for documentary evidence of typical problems encountered by QUIZ users. These queries were from real end users of QUIZ, who were attempting to use it to solve their data-processing problems. Their calls to the telesupport group are hence typical of questions that any advisor should be capable of answering. We screened the questions for obvious irrelevancies, and then classified them according to a system which we developed. [Constant et al 87] The questions fell into six main categories as follows:

HDI (how do I do such-and-such?) 51%
WHY (why is such-and-such happening?) 20%
SYN (what is the syntax for ...?) 13%
ERR (what does this .... error message mean?) 7%
HYP (what would happen if I ...?) 4%
DEF (what is the meaning of...?) 4%

Since the HDI questions alone accounted for more than half the questions, it was decided to make answering this type of question one major sub-project. A second sub-project was initiated for the second most frequent category, WHY. Only these two categories were considered of real interest to AI researchers, since the others can be handled in straightforward ways .

We made a detailed analysis of some 2000 questions. Cognos technical writers were a secondary beneficiary of this exercise. They received a weighted list of six topics that clearly gave many users difficulties, and which therefore required an improved treatment in the reference manuals. In addition, we derived lists of technical terminology that was not clearly or consistently used in the manuals, a major source of confusion both for users and ourselves as knowledge engineers. We reduced this set to a representative set of some 220 questions by eliminating irrelevant or difficult questions.

We then used these questions as a guide in designing our question answering strategy. The main thrust of this strategy was to use each question type as a focus for knowledge encoding. The concepts and rules needed for this and similar questions were added incrementally to the knowledge base. By "similar", we mean questions obtained by substitutions of other noun and verb concepts which were semantically similar. At present, we have thus covered about 30% of the 220 questions. The process of extending this coverage tends to go progressively faster, since less and less new concepts need to be added (questions overlap), and more "landmarks" exist to provide clues as to where to add new knowledge.

The project was divided into three subprojects. Each was treated relatively independently during the initial stage, termed P1, which was at 14 months from the beginning of the project. This stage was a major watershed in the project and is the focus of this paper. In the HDI subproject, we used a "shallow" approach, whereas the WHY module was built using a "deep" approach. We use these terms as follows. In a *shallow* approach, no attempt is made to design a specialized knowledge structure to thoroughly represent all the essential detail of the system being modeled. Rather, a simple, general-purpose representation, in our case frames, is the main knowledge structure. It contains only those details which are necessary to support the rules needed to answer the questions. In a *deep* approach, a knowledge representation structure adequate for virtually all aspects of the subject is selected first, which necessarily will have such features as logical adequacy (expressive power at least similar to first order logic), the ability to make definitions, and facilities to accomodate natural language phenomena such as ambiguity and anaphora. Another way of stating this difference is that in our shallow approach we directly used the facilities of KEE in which the frames support the rules, whereas in our deep approach, the knowledge representation was more complex than just a frame system, and the deduction more general than that of KEE or similar expert systems. The relative merits of these approaches are discussed in the final section of the paper; a more complete discussion of some of these comparative issues is given in [Skuce 87], [Skuce and Tauzovich 87], and [Skuce, Stanley and Tauzovich 88].

The three subprojects were:

- a parser capable of parsing a simplified version of any question asked by the user of the system, coupled with a simple user interface;
- an HDI question-answering system, implemented in KEE using a shallow approach;
- a WHY or causal question-answering system, implemented in Prolog using a deep approach.

Each of these is described in the following sections of the paper. Figure 1 show the overall architecture of the system.

## The Parser and User Interface

The parser was written in Quintus Prolog and runs on a SUN/3 workstation. LESK (Language for Exactly Stating Knowledge), the subset of English that the parser is designed to accept, is based on earlier work [Skuce 83] on simple English-like knowledge acquisition languages. The unit of input to the question-answering system is termed a *query*, consisting of a question accompanied by zero or more assumptions, which could include a fragment of QUIZ code and/or a QUIZ error message. LESK is sufficient to cope with a rewording of the assumptions and questions that were present in the question set. The parser outputs a successfully parsed query in the form of a parse tree for each statement, represented either as Prolog lists or as Lisp s-expressions, depending on the type of question (HDI questions were handled by a separate system, using KEE and Lisp; see below). If the parser fails to parse an input fragment, an interactive dialogue is initiated with the user to attempt to resolve the problem. The user is given the choice of respecifying the fragment or of undertaking a more detailed dialogue via which a new word can be defined in the parser's lexicon. The design of the parser is based on definite clause grammars and is capable of parsing any input fragment in less than 0.5 seconds.

The user interface, also written in Prolog, initializes all the communications mechanisms, controls the Lisp machine (see below) as a slave, and allows the user to enter, edit, and submit all the various components of a query to the appropriate question-answering system. This interface is adequate for research needs, but would have to be redesigned in any real advisor.

## The HDI Question-Answering Subystem

The How-Do-I question-answering subsystem (HDI) was the one into which the most effort has been directed, mainly because we wished to explore the capabilities of available large-scale AI tools, and also because it was the most common question type. We chose KEE (Knowledge Engineering Environment, [KEE 86]) on the Xerox 1186. At the time the choice was made (May 1986), KEE was the only such tool running on the hardware we had, SUNs and 1186s. Below we will discuss the advantages and disadvantages of this tool, and will compare it with its close competitor, ART [ART 86]. ART became available on the SUN too late (February 1987) to use in the project.

We used KEE's major features in the HDI part of the system: frame-based representation of knowledge; a forward/backward-chaining, rule-driven inferential system; and object-oriented methods written in Interlisp. These three mechanisms were combined in various proportions to answer a submitted query in two main stages. Figure 2 show the steps in HDI question answering.

### Stage 1

The first stage of KEE processing (box 1 in Figure 2) was termed the *semantic interpreter*. Only methods were used, mapping the query (a set of assumptions plus a question) into a set of units representing the semantic structure. These units correspond to a particular meaning of each noun and verb. A major problem was the determination of the roles (cases) in general, and the role structure for each verb, since there does not seem to have been adequate study of this problem in the literature. We desired to keep the semantic processing in this stage as generic (independent of subject) as possible, but often found it necessary to invent roles which were specific to QUIZ; our roles are hence either generic or QUIZ-specific. The semantic interpreter attempts to map phrases which modify a verb into appropriate case roles, descending into subclasses to find the most specific interpretation. We experimented with various approaches to resolving ambiguities, from taking unannounced defaults to explicitly querying the user. Which is best depends on who the user is.

We represented plural noun phrases as classes in KEE, and singular noun phrases as instances of classes. The meaning of, e.g. 'every' or 'a', depends on its context, and rules in the stage which follows the semantic interpretation must

either take special action or rely on a default interpretation. For example, the default treatment for a plural noun was to treat it as a number of "typical" instances, e.g. "how do I print every sort key" will show how to print a typical two or three, followed by "etc...".

### Stage 2

The second stage, termed the *code knowledge* stage (box 2 in Figure 2), which follows the semantic interpretation, is comprised virtually entirely of forward rules. These are triggered by acts and objects (our terminology for KEE units representing verbs and nouns) generated in the semantic interpretation stage. They create KEE units representing the syntactic components of the desired QUIZ program. We developed a three-phase approach to developing the answer, termed *generic answering*. A generic answer is one that does not give all the detail - actual QUIZ code - but gives enough hints that most users could supply the detail. For example, a human when asked the question "how do I report an item only after a subtotal?" might reply "use a FOOTING AT statement". If the asker were still unclear, the answerer would provide more detail. We do the same. A question is first answered with a "use... with...option" type of answer, which we term *generic*. This is often sufficient, and it is much easier to write rules to generate such answers than to supply all the actual minutiae of the code.

For some of our generic answers, we have gone further and supplied rules that actually synthesize the QUIZ code required, a second phase of code knowledge. The user may request this by a mouse action, which we term *completing* the answer (box 3 in Figure 2). However, given a certain number of weeks or months of an expert's time to develop rules, we would prefer that they be spent more on the generic answer rules, so that more user's questions can be handled in less detail (but nevertheless satisfactorily for most users). The detailed code synthesis rules can always be added later. When all components of QUIZ code in a completed answer have been generated as KEE units, methods attached to each are called to display a normal linear version of the program in correct syntax.

The third phase of code knowledge is termed *extending* the generic answer (box 4 in Figure 2). Usually, a question is about some lower syntactic level of QUIZ, e.g. a part of a statement. By extending, we mean a generic or complete answer about the whole statement that would be necessary, given the current assumptions, or even the whole program (all statements). The user may request such extension information by mouse action.

We found it useful to partition the rules into a number of substeps, activated sequentially, which is a common technique in forward chaining programming. For example, first we activate rules for disambiguation of the query, followed by rules that create non-referenced objects as KEE units (i.e. representations of certain QUIZ objects that are participating in the situation but were not explicitly referred to). Next, we activate rules that construct the generic answer. Finally, if the user wishes, we activate rules that construct the units representing the actual code, i.e. syntactic objects (boxes 3 and 4).

### The P1 Prototype

An initial prototype, P1, was developed in five months, before a true QUIZ expert was brought in, by AI people who knew the basics of QUIZ. This was to develop the strategy for processing the questions, together with the necessary coding, so that the expert would arrive with a stable methodology in place. The expert inherited the P1 approach in March 1987, and extended it considerably without changing the methodolgy, while the remainder of the team explored new directions (see below). After a four month learning curve, she was able to add new knowledge to the system directly, but still required assistance in making major decisions on how to structure the hierarchy and in writing Lisp code.

We feel that this approach, in which the expert is brought in only after a stable design has been arrived at by the knowledge engineering group acting initially as pseudo-experts, is desirable in applications where the experts are scarce, and the knowledge engineers can become pseudo-experts.

P1 was capable of answering tens of queries on the topics of accessing files, reporting and sorting. The emphasis was entirely on developing the methodology; no attempt was made to enter sufficient knowledge to answer all the questions. Figure 3 lists some of these queries. Achieving this required approximately 85 KEE units, representing both nouns and verbs, plus about 30 QUIZ syntactic component specifications (also units), supported by some 70 rules and about 10 KEE methods. This was developed in about 15 person-months, beginning with no knowledge of KEE or the Interlisp environment. The latter consumed probably 40% of our learning effort, since KEE cannot be used effectively in this type of application without Lisp programming. (P1 uses some thirty pages of Lisp functions). Although performance was not an issue for this prototype, it took the KEE/Interlisp-based mechanisms typically 10 to 20 seconds to answer a question on the 1186. One of the issues that we intend to address is the question of what sort of response might be considered reasonable for an advisory system.

We turn next to the second sub-system, the QAUZ system.

### The QAUZ Causal Reasoning Subsystem

The primary role of the QAUZ subsystem is to provide explanations of unexpected or puzzling features of QUIZ reports, since most "why" questions arise as a result of unmet expectations. Queries normally include background information consisting of partial QUIZ code and a number of LESK statements that describe the problem. In order to take advantage of as much information contained in a query as possible, the system explores the individual elements of the query and tries to establish causal links among them. The elements of the query are first converted into an asssertional form suitable for the subsequent deduction process by a "compiler". The system parses the QUIZ statements in the query and then uses *forward* chaining rules to determine the effects the givens in the query have on the result. When analyzing descriptions of the report however, it performs *backward* deduction to determine the features of the code, or other possible or stated assumptions, that could have caused the result. Reasoning in both directions is performed whenever both code and descriptions of results are present in the query.

QAUZ has an unusual ability for most expert systems: it can discover contradictions in queries and deal with them effectively. This ability to handle negative knowledge is necessary in an application where questions are often about failure to obtain an expected result, or where a naive user will often give inconsistent assumptions. The details of the query-answering algorithms are given in [Tauzovich 87]. Figure 4 diagrams the QAUZ subsystem. We believe that approaches based on causal reasoning in this manner will become

increasingly important as the demand for robustness and apparent "intelligence" increases [Brown 84].

Reasoning forward from the QUIZ code is done by associating each statement in a QUIZ program with a rule that describes its causal effect on the appearance of the resulting report. To be able to reason about these causal relationships, we developed an Augmented Syntax Language, which incorporates the syntax-describing conventions from the QUIZ manual (keywords, arguments, optional and repeated entries) into *causal* rules that define the semantic effects of the syntactic components.

Reasoning backward from the results obtained, described by LESK statements, we use rules which specify a result as the causal "conclusion", having LESK descriptions of the internal state of QUIZ (caused in turn by the user's code) in their premises. Thus we may go entirely backward, from results to code, entirely forward, from code to results, or, in a typical query, combine both. The system takes appropriate action if the code and described results do not properly correspond, attempting to explain to the user where the discrepancy lies. While processing a question, the QAUZ system constructs an internal representation in the form of assertions which model the actual state of the QUIZ system. This is to detect all the various dependencies and causal mechanisms that may be contributary factors to answering the "why" question. Even if a focused answer cannot be reached by the system, a browsable display of the internal representation of the problem frequently yields sufficient information to enable the user to answer the query.

In November 1986, QAUZ could answer questions from every kind of WHY question in the question sample. Figure 6 lists some of these.

We have classified the approach used in QAUZ as "deep" because the knowledge representation was developed specifically for this purpose, in Prolog, rather than attempting to fit the problem into a given structure such as KEE provides. As a result, the frame structure and deduction engine had to be coded from scratch, using Prolog purely as a programming language. We discuss further the relative merits of this in the final section of the paper.

## Related Work

Very little work has been done on software advisors. The article [Carroll and McKendree 87] discusses general design goals for advisors, and cites most of these systems up to about 1986. We know of no work that specifically uses an expert system tool or that performs rule-based causal reasoning as we have done. Perhaps the best known software advisor is the Unix Consultant [Wilensky et al 86]. This system emphasizes natural language understanding, and answers questions of similar syntactic difficulty to ours about how to do common operations in Unix. It is not rule-based, since it does not use expert system techniques; its causal reasoning uses frames and semantic nets. Its strength lies in its ability to understand linguistic speech acts, i.e. to model the goals and intentions of the user. Its knowledge of Unix seems to have a comparable depth to that in our system.

DCL [Shrager and Finin 82] is an advisor that helps a novice learn the VAX/VMS operating system. DCL does not answer questions directly, rather it monitors the user's actions and gives advice when appropriate. The system is based on a catalog of inefficient plans novices often use. The DCL network represents causal knowledge as triples: a user goal, a commonly-used inefficent method, and a better method that has the same effect. As well, Digital Equipment Corp. has been developing a methodology for operating system consultants [Billmers and Carifico 85]. Their first research system, TEACHVMS, helps novices who are familiar with TOPS20 learn VMS. It is a forward-chaining system, implemented directly in C. It accepts TOPS20 commands, and produces the equivalent VMS commands. A second system, TVX, builds on the first to produce a generic "shell" for encoding knowledge about any operating system. It separates "generic" operating system knowledge from "specific" knowledge for one system, e.g. VMS. It uses a backward rule-based approach to synthesizing plans to accomplish the user's goal. It does not explicitly represent causality, nor does it have a significant natural language ability.

A somewhat broader category of system might be generically termed "intelligent front ends", though the term "assistant" is often used as well. We take this term to be quite general, encompassing any system that acts as an interface to some other system such as a database. The FRED system [Jacobson et al 86] is a database front end. Their methodology is similar to our HDI system, though they do not mention using an expert system tool; apparently rule-based knowledge is a relatively small part of the system. The articles by [Berry and Broadbent 86/87], [Bundy 85], [Spark Jones 85], and [McKeown 84] are all general discussions of the problems in integrating natural language and expert system technology in intelligent front ends.

There are a number of systems that may be termed "programmer's assistants", i.e. which help in the program development process. The function of these is more similar to that of the HDI component of the Cognos Advisor than to the QAUZ component, in that they perform code synthesis rather than analysis. Typical of these is the Programmer's Apprentice [Waters 86], which acts as a "clerk" to a programmer. The PROUST system [Johnson 85] can identify novice-level semantic errors in a Pascal program. It is based on a catalog of typical plans and goals, and performs causal reasoning about the user's intended goals. The Unix Computer Consultant [Douglass and Hegner 82] is another Unix assistant. The front end accepts a natural language query which is translated into a formal query language similar to "logical" database query languages. These systems typically use a deep representation.

## Discussion and Conclusions

Our primary conclusions have been the following:

**1.** A major distinction has to be made between problems that can be adequately handled with a shallow model, e.g. the frame and rule structures that today's expert system tools supply, and those that need a deep model, which these systems, with the possible exception of ART, can't adequately support without using Lisp or Prolog extensively. In our project, a useful variety of questions was supportable by KEE plus Lisp (the parser in Prolog could easily be rewritten in Lisp), and indeed, a shallow advisor based on the P1 approach has been shown to be feasible. However, it would probably not be economically sensible to use a tool like KEE for this purpose. Cheaper tools that require less resources are now available that, while not as flexible, are probably adequate. The primary role of a system like KEE then is probably one of prototyping, for which it was certainly very useful, but for delivery purposes, the resouces it requires are still extensive, even in 386-based versions. This situation will improve considerably in the next few years, of course.

Prolog systems today are more advanced (e.g. windowing) than two years ago, but to build a system such as ours entirely in Prolog one would still have to code many of the facilities that good expert system tools provide from scratch. We know of no expert system shell written in Prolog that remotely approaches the capabilites of KEE or ART. For example, forward chaining in Prolog, except for small rule bases, would require writing a serious forward chaining engine, a major problem.

How then does one decide in advance whether one has a shallow problem, committing to a system such as KEE or ART used without much enhancement, vs a deep problem, that either needs considerable enhancement to an existing tool, or the use of no tool at all, i.e. coding directly in Lisp or Prolog? Clear guidelines for this problem would be very valuable. We offer the following "rules".

- any problem in which more than trivial natural language input is desired is probably a deep problem. Advisors can easily fall in this category.
- any problem in which the most thorough representation of domain-specific knowledge is clearly essential to acceptable performance is definitely deep.
- any problem that involves reasoning about sets, more complex mathematical concepts such as logical relationships, temporal relations between events, or procedural knowledge is very probably deep.
- any problem which requires some form of deduction which is not supported by existing tools, as in our QAUZ module, is definitely deep.
- any problem which does not require natural language and for which experts are content to express their knowledge in the language of a KEE or ART-like notation is probably shallow.

**2.** The two approaches used in the HDI and QAUZ modules are difficult to compare. A proper comparison would require that two equally qualified groups undertake the exact same task to allow a more precise comparison. While this would clearly be an interesting experiment, few could afford it. The two modules were similar enough however that we feel confident in drawing some conclusions. The particular reasoning needs of the QAUZ module were sufficiently unique that they could not be met with the built-in deduction mechanisms provided by existing expert system tools, with the possible exception of ART. (ART's capabilities border between shallow and deep.) Hence a deep approach was necessary. If we rebuilt the QAUZ system in ART, which has more powerful deductive abilities than KEE, its deduction engine might be sufficient. We could of course make use of the frame system directly, plus other benefits such as graphics, truth maintenance, and viewpoints. Thus we possibly could have implemented QAUZ in ART, but probably not in KEE. However, the excessive resource problem would remain.

The reasoning needs for the HDI module (shallow approach) were met with a tool like KEE. Had we implemented it in Lisp or Prolog directly, we would have had to reinvent much of the machinery that these tools provide, particularly forward chaining, a questionable prospect. However if one wishes to take a deep approach, without using Lisp or Prolog directly, among the expert system tools, only ART seems adequate as a platform. We do not recommend attempting a deep version of a HDI-type advisor, except as a research project, at this time.

In the HDI module, we used methods in the semantic interpreter stage, and rules in the code knowledge stage, to gain experience with both. Our general feeling is that rules are much easier to understand and maintain than methods, which contained often complex Lisp code. However some of our rules had considerable LISP code in the conclusion, making them a kind of hybrid: a "pattern-invoked" function. This was necessary because we found that while the rule pattern language in KEE was adequate for most rule premises, it was not for many conclusions. (A similar comment applies, to a lesser degree, to ART rules.) Hence, were we to redo the semantic interpreter in KEE, we would replace the methods with rules, some of which would be quite "hybrid".

**3.** We have also been impressed with the relative difficulty of representing natural language semantics with respect to programming language semantics. We spent many hours debating what the role structure for our verbs should be. This is, of course, one of the key problems in natural language semantics research.

A closely related problem derives from how carelessly people use natural language, even in supposedly well-written documents like technical manuals. We were often frustrated by imprecise, ambiguous, inconsistent, and even illogical technical terminology in the QUIZ manual, which nevertheless reflects the actual usage of these terms as they have evolved over a number of years without anyone seriously attempting to control them. (It should be pointed out, however, that Cognos manuals are considered excellent by contemporary standards.) This project has impressed us with the need to take terminological control very seriously. The question of what tools and techniques to use for this, which involves establishing and controlling terminology uniformly throughout a large organization, is an interesting one which we are investigating. It seems clear that techniques such as described by Sowa [Sowa 84], may be beneficial.

**4.** Turning away from the problems of tools and natural language, we would suggest that the development of software advisor systems of the kind described here, using a shallow approach, should be a thrust of expert system research. We believe that such systems are certainly attainable in the near term. Such systems would be very useful, and would enhance the productivity - and hence marketability - of virtually any complex software product. We expect more activity in this area; the two last international expert systems conferences in Avignon had only one paper on the subject: our own [Szpakowicz et al 86].

**5.** Our experience allows us to estimate that a shallow system which could handle many hundreds of simple user questions for a product of the complexity of QUIZ would involve about 500 KEE units, in the order of 1000 to 2000 rules, and a simple parser with a vocabulary of about 1000 nouns and 100 to 200 verbs. Developing such a system would take a number of person-years.

Finally, we believe that the marriage of natural language and expert system technology is inevitable, and that a wide variety of systems, not only front ends or advisors, can be be based upon it.

## Acknowledgements

The Advisor project was a industry-university project between Cognos Incorporated and the University of Ottawa, funded by grants from the National Research Council of Canada, The Natural Sciences and Engineering Research Council of Canada, and the Ontario Ministry of Colleges and Universities, and Cognos itself.

## REFERENCES


ART 86
ART *Reference Manual (version 3.0)*, Inference Corporation, Los Angeles, CA.

Berry and Broadbent 86/87
Berry, D. and Broadbent, D., Expert Systems and the Man-machine Interface *Expert Systems* , 3 pp. 228-233 and 4 pp. 18-27 (two parts)

Billmers and Carifco 85
Billmers, M.A., and Carifco, M.G., Building Operating System Consultants, Proc. Second IEEE Conference on Artificial Intelligence Applications, Miami Beach, pp. 449-454.

Brown 84
Brown, J.S., The Low Road, the Middle Road, and the High Road. In: *AI Business*, P.H. Winston, K.A. Pendergast (eds.), the MIT Press, MA, Cambridge.

Bundy 85
Bundy, A. Intelligent Front Ends. In: Bramer, M. (ed.) *Research and Development in Expert Systems,* Cambridge University Press, New Rochelle, pp. 192-203.

Carroll and McKendree 87
Carroll, J. and McKendree, J., Interface Design Issues for Advice-Giving Expert Systems. *Comm. ACM* 30, pp. 14-30.

Constant et al 87
Constant, P., Matwin, S., and Szpakowicz, S., Question-driven Approach to the Construction of Knowledge-based Software Advisor Systems. Proc. Third Annual Conference on Artificial Intelligence Applications, Orlando, FL.

Delisle 87
Delisle, S., A Natural Language Interface for an Expert Advisor System. M.C.S. thesis, Department of Computer Science, University of Ottawa.

Douglass and Hegner 82
Douglass, R. and Hegner, S., An Expert Consultant for the UNIX System: Bridging the Gap Between the User and Command Language Semantics, Proc. Fourth National Conf. of the Canadian Society for Computational Studies of Intelligence, Saskatoon, pp. 92-96.

Jacobson et al 86
Jacobson, G., Lafond, C., Nyberg, E. and Piatetsky-Shapiro, G. An Intelligent Database Interface. *IEEE Expert*, Summer 1986, pp. 65-78.

Johnson 85
Johnson, W.L., PROUST: A System Which Debugs Pascal Programs, Proc. Expert Systems in Government Symposium, McLean, VA, p. 157 (October).

KEE 86
KEE Reference Manual, Version 3.0, Intellicorp, Mountain View, CA, (1986).

McKeown 84
McKeown, K. Natural Language for Expert Systems: Comparisons with Database Systems. Proc. COLING 1984, pp. 190-193.

QUIZ 85
Quiz Version 5.01 *User's Guide*, Cognos, Inc. Ottawa, 1985.

Shrager and Finin 82
Shrager, J., and Finin, T., An Expert System that Volunteers Advice, Proceedings of the National Conference on Artificial Intelligence, pp. 339-340.

Skuce 83
Skuce, D. The LESK Tutorial. Department of Computer Science, University of Ottawa, Tech. Report TR-83-03.

Skuce 87
Skuce, D., A Comparison of Conceptual Graphs and Frame Systems. To appear in: *Conceptual Graphs for Knowledge Systems*, Sowa, J., Foo, N., and Rao, A. (eds.), Addison Wesley, Reading, MA, 1987.

Skuce and Tauzovich 87
Skuce, D. and Tauzovich, B., Causal Reasoning in an Advisor for Fourth Generation Software. To appear in: *Expert Systems.*

Skuce, Stanley and Tauzovich 88
Skuce, D., Stanley, R., and Tauzovich, B. An Expert Advisor for Commercial Fourth Generation Software. To appear in: *International Journal of Expert Systems.*

Sowa 84
Sowa, J. *Conceptual Structures: Information Processing in Mind and Machine*, Addison Wesley, Reading, MA.

Spark-Jones 85
Spark-Jones, K., Natural Language Interfaces for Expert Systems: An Introductory Note. In: Bramer, M. (ed.), *Research and Development in Expert Systems.* Cambridge University Press, New Rochelle, pp. 85-94.

Szpakowicz et al 86
Szpakowicz, S., Matwin, S., and Skuce, D., QUIZ Advisor: A Consultant for a Fourth Generation Software Package. Proc. of the 6th International Workshop on Expert Systems and Their Applications, Avignon, France (1986).

Tauzovich 87
Tauzovich, B., Causal Reasoning in a Software Advisor. Ph.D. thesis, Department of Electrical Engineering, University of Ottawa (1987).

Waters 86
Waters, R.C., KBEmacs: Where's the AI? *The AI Magazine* 7, pp. 47-56.

Wilensky et al 86
Wilensky, R., Mayfield, J., Albert, A., Chin, D., Cox, C., Luria, M., Martin, J., and Wu, D., UC - A Progress Report, Report No. UCB/CSD 87/303, Computer Science Division, University of California, Berkeley, (July).

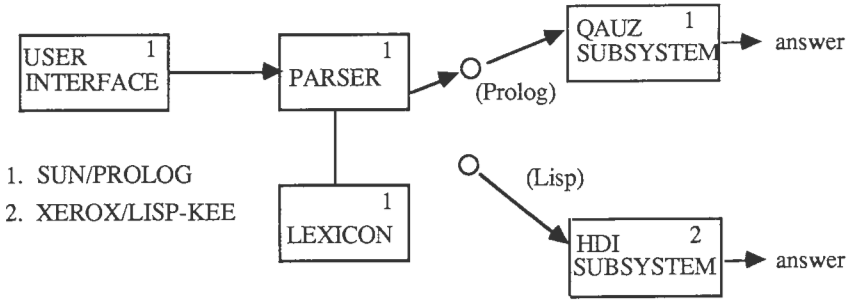



**Figure 1: Advisor System Architecture**

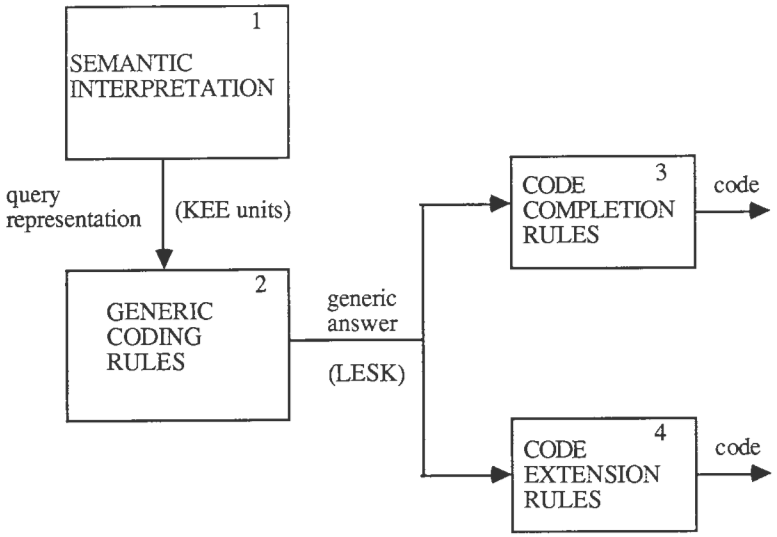



**Figure 2: The HDI Subsystem**
**Stages 3 and 4 Optional**

hdi report a record item at a sort key to a subfile?

hdi print an average of an item at a control break?

hdi average an item in a footing?

hdi output a page skip at a control break?

hdi print an item at the beginning of a line?

hdi sort in descending order?

hdi sort on some record items?

hdi sort on 2 record items and 1 or more defined items?

hdi link two files?

x is a subfile. hdi link x to x?

hdi access a file using 2 keys?

**Figure 3:** Some "how do I" questions.

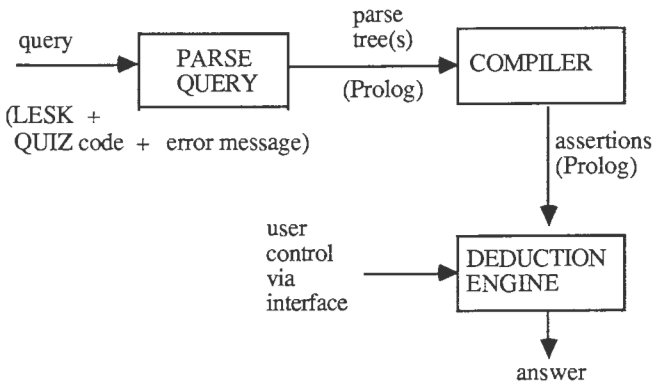



**FIGURE 4:**
**THE QAUZ Subsystem**

> What happens if I use the code

```
ACCESS EMPLOYEES ALIAS STAFF LINK TO BILLINGS OPTIONAL
SELECT  IF  DATAEJOINED GT 810101
REPORT ALL
```

{24 lines of assertions are produced}

---

> CODE: REPORT ALL

Why is the column heading is set to the dictionary heading?

{a 23 line backward trace explaining this is produced}

---

> When is the column heading set to the dictionary headings?

{a similar 31 line backward trace ultimately showing
code that will do this is produced}

---

> CODE: ACCESS EMPLOYEES
SELECT EMPLOYEES
REPORT

Why is nothing reported?

{the error in the code is diagnosed}

---

> ASSUMPTIONS: an alias is specified
CODE: ACCESS EMPLOYEES

<any question here>

{the inconsistency between the assumption and the code is noted}

---

> ASSUMPTIONS: 'X' is an item
the type of 'X' is string
'F' is the field of 'X'

> Why is 'F' not truncated?

{a backward trace which shows that something is NOT true
is given as the reason}

---

## APPENDIX

A selection of typical QUAZ queries. The full answers are not shown because they are long.

# Refinement of Scene Interpretation for Object Recognition and Location

K. D. Rueb and A. K. C. Wong
Systems Design Engineering
University of Waterloo

**Abstract**

This paper presents a method for identifying and precisely locating modeled objects through analysis of single or multiple perspective images. The system provides reliable and efficient scene interpretation through incremental refinement of the hypothesized measurement conditions and 3-D environment. Automated visual detection of camera viewpoints permits use of unconventional viewing conditions, such as multiple views per image due to reflective surfaces or mirrors, and removes the need for precise camera positioning. Automatic integration of any number of arbitrary viewpoints permits detection of objects occluded from certain views, as well as providing varying image resolution and wider coverage of the visible workspace.

## 1 Introduction

Development of reliable and flexible methods for automated assembly is primarily limited by the need for precise positioning of the components used. Current assembly methods require use of complex and expensive jigs or fixtures to supply and position parts manipulated by the robot arm. One approach to simplifying robotic assembly is to locate the required parts visually. This approach is currently used under very constrained viewing conditions [1]. Attempts to achieve such capability under natural conditions (no special lighting or constrained camera viewpoints) must deal with missing or extraneous features due to glare or shadow, partial or total occlusion by other objects or even the robot arm itself, as well the fundamental ambiguity resulting from projection to a 2-D image from the original 3-D scene.

In spite of these difficulties, some progress has been made, in particular, with new methods to deal with partial occlusion of objects. In general, success has been achieved through a technique commonly referred to as hypothesis verification. A hypothesis of object identity, position and orientation maps an object model into the image. If a unique combination of object features remains visible, such recognition is possible even in the presence of severe image noise and occlusion of object features.

Such hypothesized object positions may be used directly through voting or clustering to determine object position and identity [2,3]. More commonly however, the hypothesis is used to test known 3-D models against detected image features. This technique has been demonstrated in the analysis of 3-D scenes [4-7] as well as 2-D scenes with extensive occlusion of object features [8-10]. For 3-D scenes ambiguity of feature identification under perspective transformation prevents rapid analysis and typically requires on the order of one minute of processing time per image.

Our goal is to provide a practical system for industrial use. Under such conditions, total processing for each image must be completed in at most a few seconds. We have found that to meet such requirements, it is necessary to exploit the natural constraints of the application environment. The key to such an approach is to use *three-dimensional* constraints which provide a more natural expression of the environment restrictions independent of the camera viewpoint. For example, analysis of an image of a robot workcell should be constrained such that all objects are above or on the surface of the workspace or table. This immediately constrains the scale of valid image features and limits the candidates for selected object features.

To apply such constraints we have developed a method of analysis that we characterize as *hypothesis refinement*. In contrast to previous approaches which essentially use image features to *generate* possible hypotheses, we initially specify a poorly constrained set of hypotheses that are then refined on the basis of detected image features until specific testable hypotheses are obtained. The initial set of hypotheses reflect the constraints on the position and orientation of different objects appropriate for the given application. As additional images are acquired, more information may be applied. As explained later, this approach also provides improved measurement accuracy and interpretation reliability.

## 2 Hypothesis Refinement

Image analysis or interpretation may be viewed as a search of interpretation hypotheses which specify the possible location and orientation of modeled objects within the 3-D

scene. Hypothesis refinement searches the space of possible scene interpretation through refinement of an initially specified set of hypotheses which represent the natural 3-D constraints of the application environment. Some of the most commonly encountered constraint conditions for robotic assembly tasks are listed in Table 1. [1] Each constraint is further partitioned into a set of characteristic views (specified object orientations from which a similar set of features is visible). This provides the initial set of interpretation hypotheses.

| Type | Constraint | Applications |
|---|---|---|
| Known Position | Object position and orientation are constrained to a given range. | Monitor object with known position, verify success of robot pick or place operation, check for presence of part in part feeder. |
| Flat Support | Object is supported by a flat surface. Vertical orientation and position are constrained. Object must be in stable support configuration (vertical projection of center of gravity must lie within the convex hull of surface contact points). | Locate objects on a workspace of the robot workcell or supported by other objects, only minor overlap of components permitted. |
| Bin Picking | Object is contained within a parts bin. Object position is constrained within known limits, orientation is unconstrained. Object identity is known. Select most visible of many parts in bin. | Selection and location of parts in bin for assembly tasks. |

Table 1: Constraint Conditions

The constraints of each hypothesis are then applied to select candidate image features corresponding to key features of the object model. Figure 1 illustrates the selection of candidates for a model feature under flat support conditions. Note that the constraint is three-dimensional — the local image characteristics of selected features are dependent on their location in the scene and the corresponding perspective projection of the model feature.

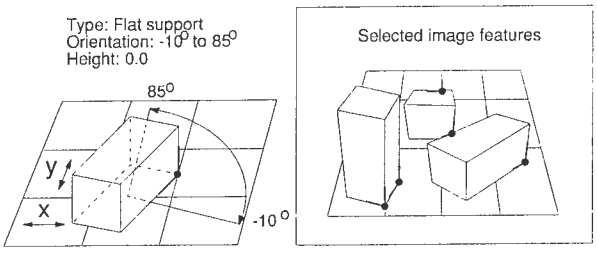



Figure 1: Selection of candidate image features

The precise nature of such 3-D constraints permits the filter to select only 5 candidates from the 90 corner features present in the image. The filter will select any features with identical 3-D configuration as well as any features only apparently similar in the 2-D image projection. Note that partitioning the initial hypotheses into characteristic views does not increase search complexity as the filter will select different candidate features for different object orientation. Consequently, selection of characteristic views streamlines the interpretation by limiting analysis to the set of visible features without requiring expensive analysis of feature visibility.

Selection of appropriate constraints also has an impact on the accuracy and reliability of analysis. For example, a false positive in perspective scene analysis occurs due to random alignment of image features in a configuration matching a projection of the object model. In general, such a match will have any random orientation or position. The tight constraints of the interpretation eliminate all such matches; indeed they are never even considered by the analysis since they are not within the constraints of the initial hypotheses. Most errors which do occur are between objects with similar 3-D shape and appearance. The greater the tolerance to partial object occlusion (achieved by permitting a large set of missed features), the greater the possibility of false recognition.

Secondly, the available constraints can improve measurement accuracy. For example, depth measurement is limited by the image size of the projected object. In contrast, position measurement perpendicular to the camera axis is only dependent on the resolution of the image. Assumptions such as a flat support plane map horizontal position into limits on object depth improving depth resolution. Hence, the constraint of a flat support surface can provide precise positioning of small objects (eg. a bolt or screw) that would be impossible judging solely by object appearance.

# 3 The Knowledge-directed Search

The process of refining an interpretation hypothesis can generally be described by an ordered sequence of well-defined steps or stages. In actual practice, processing is more complex. Input to the system may consist of multiple images or viewpoints, and additional workspaces may be defined with multiple object models and refinement strategies. Analysis of the broad range of possibilities is managed through a *knowledge-directed* search. The search is directed by the knowledge or context of the task or environment. [2] The knowledge-directed search is guided by a rule network consisting of a network of *rule nodes*. Each step by step refinement process, or refinement strategy, is represented as a path through a set of nodes in the network.

At each step of the refinement strategy, new interpretation hypotheses are produced. Such hypotheses determine the constraints on the object position, and specify an assumed correspondence between model and image features. The refinement of a new hypothesis is dependent on i) the search rules (or processes) and domain knowledge associated with the rule node of the search path, ii) the assumed model, feature correspondences, and position constraints of the current hypothesis (referred to as the *context* of the

[1] In the current system, only flat support and known position constraints have been implemented. This is sufficient for only a limited class of assembly operations.

search), and iii) the observed data that is selected to infer a more precise characterization of the hypothesis. The knowledge-directed search represents all such information as a set of search activations (refer to Figure 2).

> Each *search activation* is a relation that uniquely associates a node of the *rule network*, a set of *observed data*, and the interpretation hypothesis (or *context*) of the search.

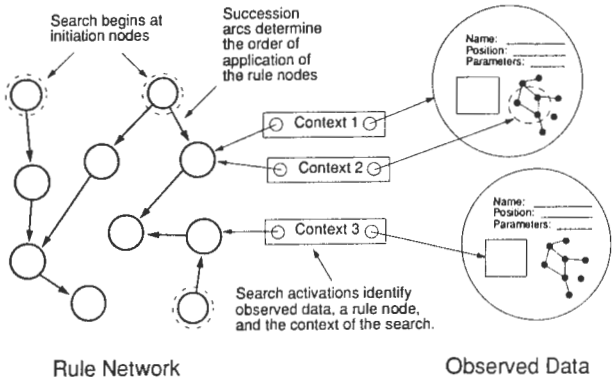



Figure 2: Rule network and search activations

Each node is a instance of a class of rule nodes which perform a specific task (eg. acquire an image, detect image features or specify an object model, constraint and refinement strategy). The rule node classes employed in the present system are listed in Table 2.

| | |
|---|---|
| Image Acquisition | Obtain image from specified source (camera or disk file) and set camera parameters (focal length, imaging plane, focus setting). |
| Line Detection | Obtain line features according to specified limitations (line length, contrast, straightness, resolution). |
| Camera Position | Obtain camera position and orientation relative to position target reference. |
| Feature Detection | Obtain image features (line end points, corner features, 'T' junctions). Index according to image location. |
| Model Definition | Define 3-D object model and interpretation constraints (i.e. search strategies). |
| Hypothesis Refinement | Refine object position and orientation to verify possible hypotheses. |
| System Control | Select appropriate search strategies and object models. Construct a consistent world model from refined and verified interpretation hypotheses. Control low level processing and feature detection. Interact with other processes (eg. control robot arm). |

Table 2: Rule node classes

Each class has an associated set of processes and defined structure for input search context. Each instance of a rule node acts on an input context to produce any number of output context records. In addition, each instance of a rule node has a private internal memory to record specific parameters of the instantiation of the rule node or to compile a history of results or information acquired.

[2]This is similar in spirit to the "knowledge-directed image analysis" of Ballard, Brown and Feldman in which image understanding was directed by a search query from the person using the system [11].

As context information is transmitted through the network additional information is supplied by each node to eventually provide an interpretation of the world viewed by the various cameras of the vision system.

In general, processing is in the sequence: acquire image, detect line features, locate camera position, establish point or corner features, initiate model search strategies, refine and test interpretation hypotheses, and construct a consistent world model. The structure of the associated rule network is illustrated in Figure 3. The network layout is flexible. Any number of instances of each class of node may be defined. For example, there may be any number of image nodes, each image can be assigned specialized edge or feature detectors, and various models or refinement strategies may be defined.

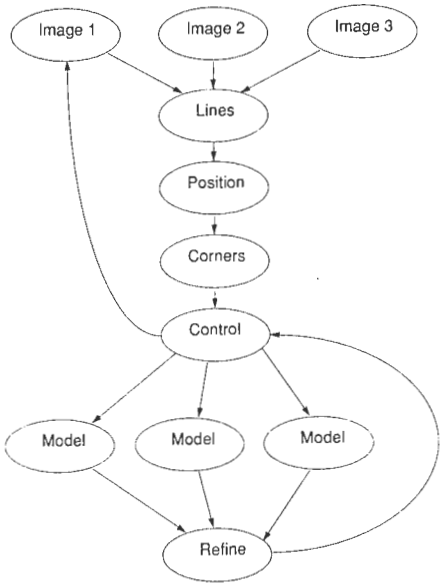



Figure 3: Structure of rule network

## 3.1 Establishing Camera Position

The position of each camera is determined by visually locating a predefined target pattern. The precise coordinates of the chosen target pattern and the defined reference coordinate system are shown in Figure 4. The scale of the

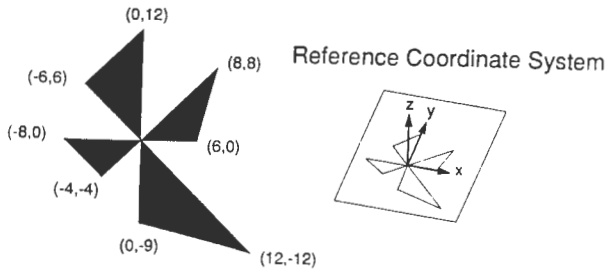



Figure 4: Target pattern

pattern is varied to suit particular applications and camera viewpoints. The unique characteristics of the chosen target pattern permit rapid detection despite the lack of available constraints. Including necessary feature detection, the target may be tracked at a rate of approximately two images per second.

Once the target position has been obtained, one or more workspaces may be defined. Each workspace defines a potential support region within the robot workcell. A workspace is a flat region with specified position, orientation and area. Each workspace produces an associated search context to be propagated through the rule network. In the absence of conflicting evidence, such context are retained. As a result, the target pattern may be removed once all camera positions have been identified.

If more than one target is visible in an image, multiple viewpoint hypotheses and associated workspaces are created. Since the correct transformation of the target cross pattern is known, scene interpretation is possible under any arbitrary viewing conditions. For example, if the target image is reflected in a mirror, the generated viewpoint will provide the correct transformation between left and right handed coordinate systems (the 'mirror image') as well as adjusting for apparent position and orientation. As a result, the reversal of object coordinates in the reflected image is transparent to the subsequent scene interpretation. Consequently, it is possible to provide more complete visual coverage of the workspace by placing mirrors to view areas hidden by other objects in the environment.

### 3.2 System Control

Scene interpretation through hypothesis refinement culminates in the production of verified hypotheses of object identity and position. This interpretation is represented by the context information of a set of unrelated search activations. Some information will be redundant due to various search strategies arriving at a similar result or through overlapping views of different camera viewpoints. The primary role of system control is to collect all such context information and construct a coherent scene interpretation. This interpretation of the 3-D environment provides additional information for analysis of any further images of the environment.

The current scene interpretation is the primary source of *known position* hypotheses. Such hypotheses may be used to test the current scene interpretation in succeeding images or to verify external action such as the placement of an object by the robot arm. In addition, location of objects in the scene will define additional hypothesized workspaces if any of the object surfaces are capable of providing stable support for other objects in the environment.

System control may also be used to limit the complexity of the analysis. For example, it may limit search to only those parts required in the current stage of robotic assembly or may place a limit on the number of instances of each object detected, as required by the needs of the current task.

## 4 Current System Implementation

Our current vision system provides analysis of single or multiple perspective images to locate parts visible in the workspace of a robot workcell. In its present configuration, the system consists of a SUN 3/160 workstation, a Matrox video frame grabber, and from one to three SONY CCD video cameras (see Figure 5). Each digitized video frame can be directly accessed by programs running on the SUN workstation. Approximately ten 512x480 images may be acquired each second. No additional special purpose hardware is required. Image processing, feature detection, and analysis are implemented in software in portable 'C' code and can be run on various computer systems (without modification on systems supporting the Xwindow graphics environment). The system is capable of processing multiple camera/viewpoint input in a few seconds.

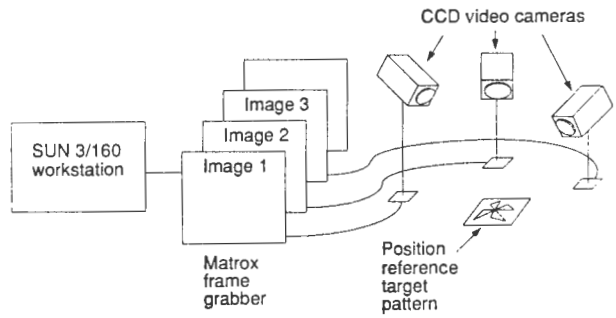


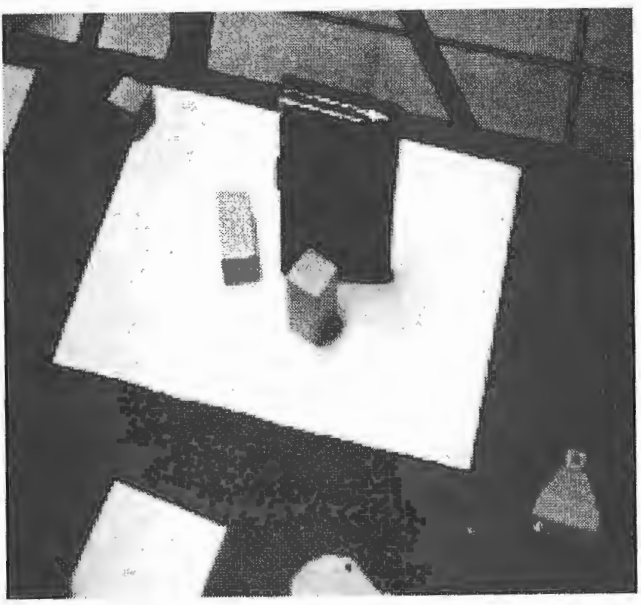

Figure 5: System configuration

## 5 Experimental Results

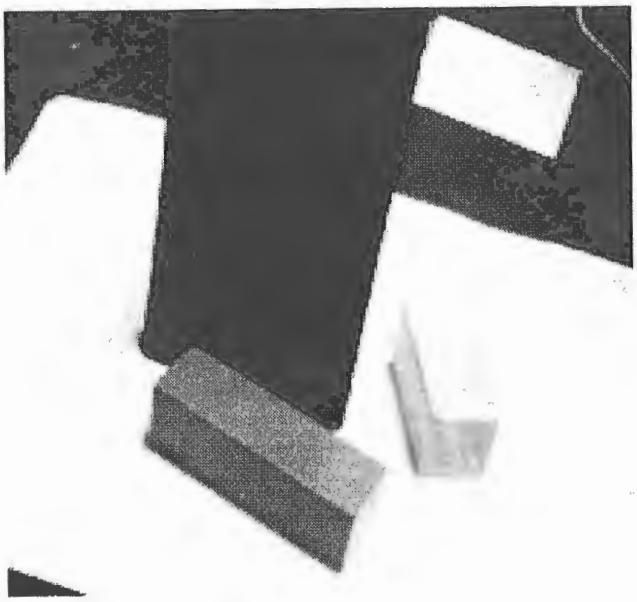
In this section, a selected set of images is presented to demonstrate key features of the vision system. In each case, input is from one or more 512x480 pixel CCD cameras.

The first example illustrates integration of multiple camera input. In this case, the system has been supplied a set of simple block models. Two views of the scene are available as shown in Figures 6(a) and 6(b). Neither viewpoint has access to the entire scene because of an occluding object. Figure 6(c) is the resulting interpretation of the first image. [3] Figure 6(d) presents the second scene. Note that the viewpoint is radically different, both in viewing direction and choice of a 'close-up' of the workspace. Although not shown, the position target was initially displayed to calibrate the positions of each camera. Consequently, it is possible to integrate the scene interpretations. The resulting interpretation is shown as viewed from above the workspace (Figure 6(e)) and from the side (Figure 6(f)). The workspace is shown as a grid of 2x2 cm squares. Note that the occluding object is not detected as no model of the object is available.

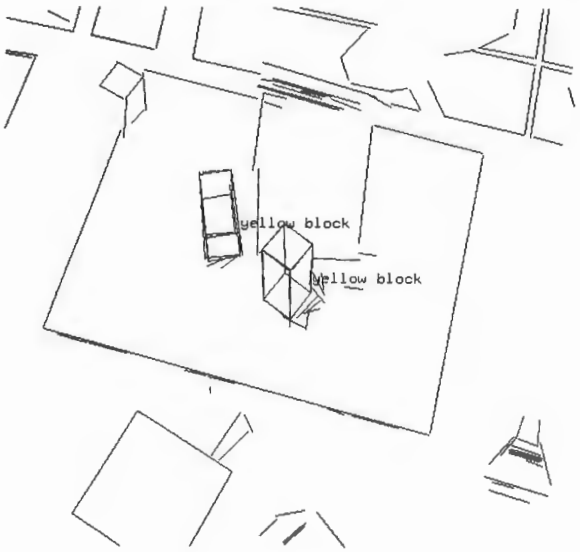
[3] The object model is superimposed on the detected edges and identified by the object model name. All lines of the object model are shown giving the effect of a wireframe or transparent object.



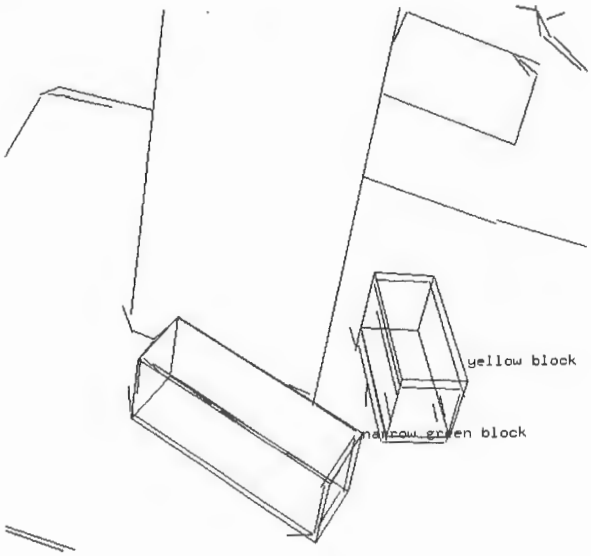
(a) standard view



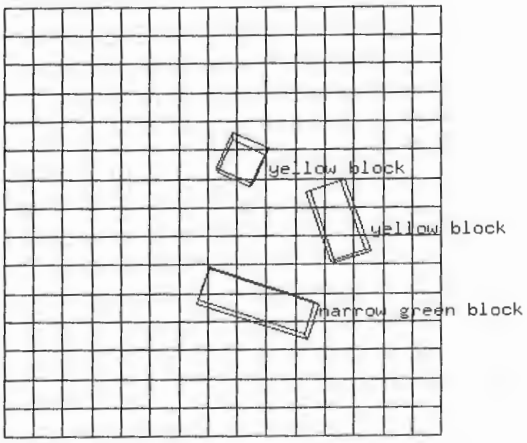
(b) near view



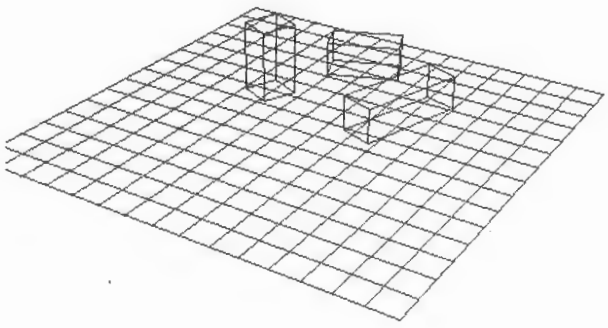

(c) standard scene




(d) near scene




(e) overhead view

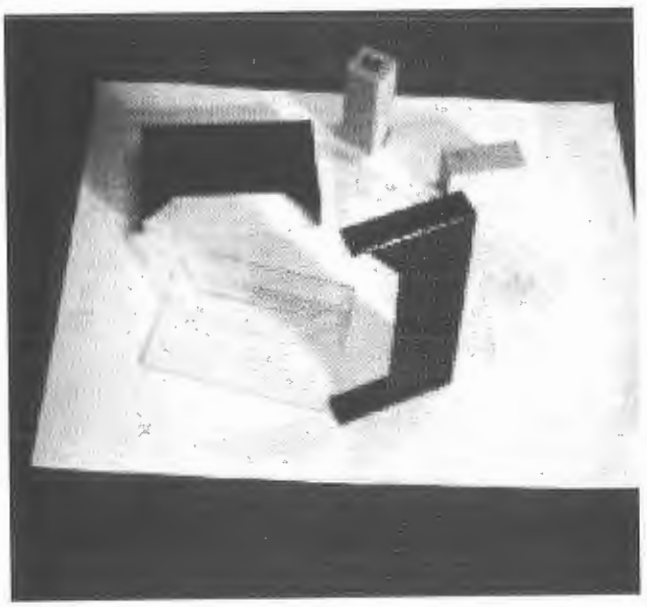


(f) side view

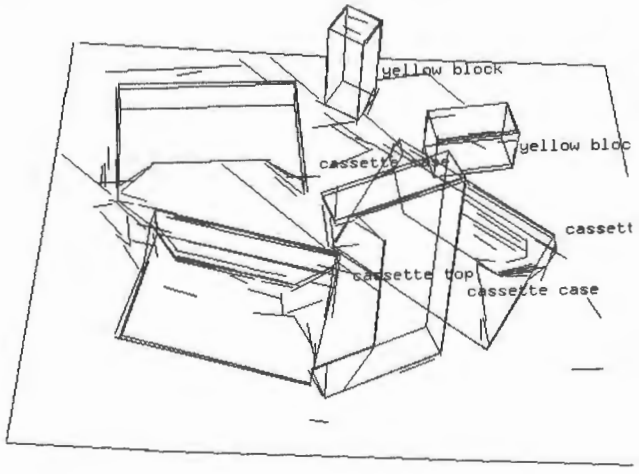
Figure 6: Integration of multiple views

The set of object models is then extended to include the two components of an audio cassette case (the transparent cassette top and black plastic case). Figure 7(a) is a typical image of a scene containing the modeled components. Figure 7(b) is the resulting scene interpretation. Figure 7(c) illustrates the scene interpretation as viewed from above the workspace.

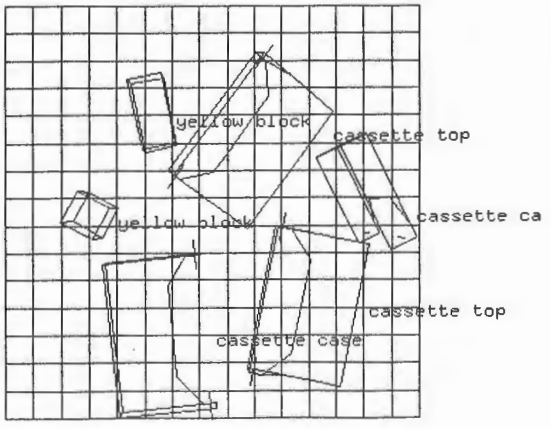
In the final example, a mirror is placed behind the workspace to permit detection of hidden or severely occluded objects (Figure 8(a)). The scene interpretation is shown in Figure 8(b). Note the detection of the block hidden by the stacked block and cassette case in the foreground of the image. Figures 8(c) and 8(d) show an overhead view of the scene interpretation and a view from the opposite side of the workspace (as viewed from 'behind' or 'through' the mirror). The mirror is purposely inclined relative to the table surface so that reflections of symmetrical objects are not mistaken for objects behind or 'inside' the mirror's surface.



(a) original image




(b) scene interpretation




(c) overhead view

Figure 7: Location of cassette case

## 6 Summary

In this paper, we have presented a method for locating objects in a 3-D scene through analysis of single perspective images. As demonstrated in this paper, such analysis can be completed in a few seconds, permitting practical use in a robot workcell. This performance is achieved through exploitation of the natural three-dimensional constraints of the application domain. Constraints on object position are represented as interpretation hypotheses which are refined to produce a final scene interpretation. Camera position is determined visually through location of a position reference target permitting simple and reliable integration of different viewpoints. This permits the use of mirrors or input from multiple camera positions to provide detection of objects occluded in the initial view.

## 7 References


1. Schroeder, H.E., "Practical Illumination Concept and Technique for Machine Vision Applications", *Robots 8*, Detroit, June 1984, pp.27-43.

2. Thompson, D.W., Mundy, J.L., "Three-dimensional Model Matching from an Unconstrained Viewpoint", *Proceedings of 1987 IEEE International Conference on Robotics and Automation,* vol. 1, 1987, pp.208-220.

3. Ballard, D.H., Sabbah, D., "Viewer Independent Shape Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. PAMI-5, no. 6, 1983, pp.653-660.

4. Walter, I., Tropf, H., "3-D Recognition of Randomly Oriented Parts", *SPIE vol. 449*, part 1, 1984, pp.171-178.

5. Lowe, D.G., "Three-Dimensional Object Recognition from Single Two-Dimensional Images", *Technical Report No. 202,* Courant Institute of Mathematical Sciences, New York University, New York, February 1986.

6. Horaud, R., "New Methods for Matching 3-D Objects with Single Perspective Views", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. PAMI-9, no. 3, May 1987, pp.401-412.

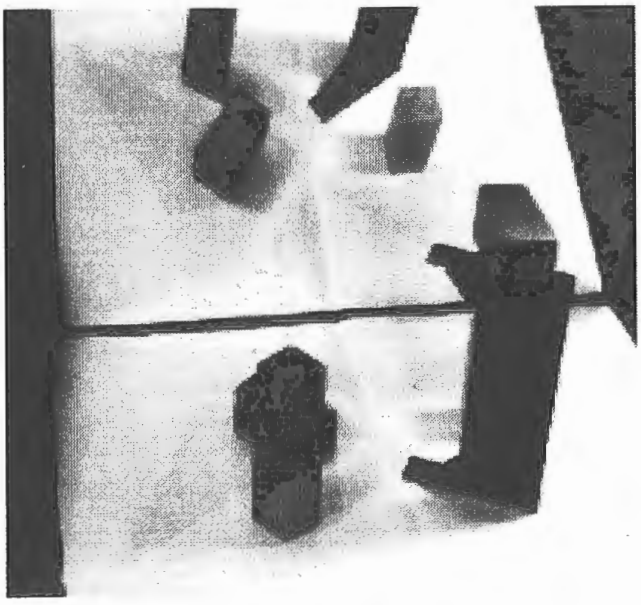
7. Grimson, W.E.L., Lozano-Perez, T., "Localizing Overlapping Parts by Searching the Interpretation Tree", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. PAMI-9, no. 4, July 1987, pp.469-482.

8. Knoll, T.F., Jain, R.C., "Recognizing Partially Visible Objects Using Feature Indexed Hypotheses", *IEEE Journal of Robotics and Automation,* vol. RA-2, no. 1, March 1986, pp.3-13.

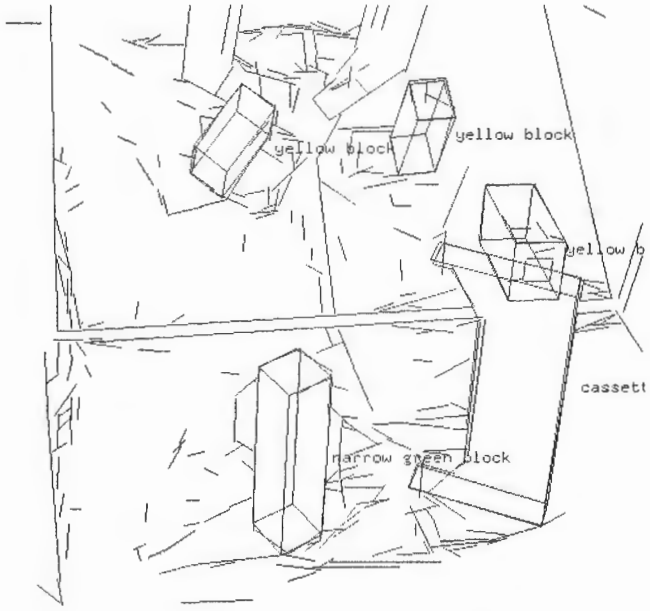
9. Kalvin, A., Schonberg, E., Schwartz, J.T., Sharir, M., "Two-Dimensional, Model-Based, Boundary Matching Using Footprints", *The International Journal of Robotics Research,* vol. 5, no. 4, Winter 1986, pp.38-55.

10. Koch, M.W., Kashyap, R.L., "Using Polygons to Recognize and Locate Partially Occluded Objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. PAMI-9, no. 4, July 1987, pp.483-494.

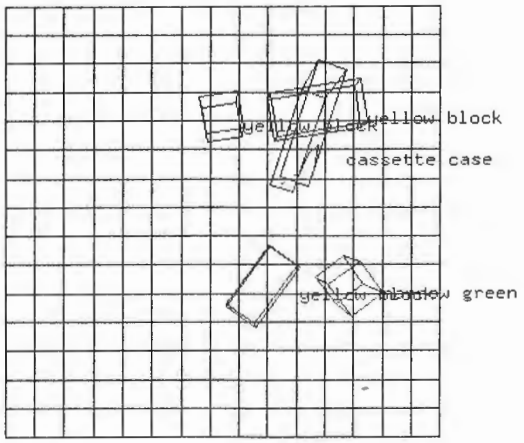
11. Ballard, D.H., Brown, C.M., Feldman, J.A., "An Approach to Knowledge-directed Image Analysis", *Computer Vision Systems*, ed. A.Hanson and E.Riseman, New York: Academic, 1978, pp.271-281.



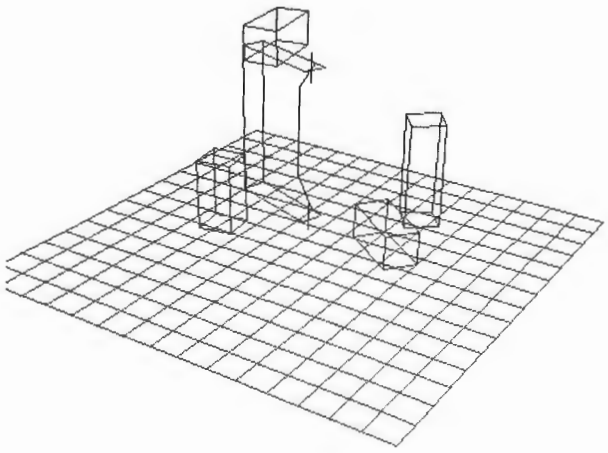
(a) original image




(b) scene interpretation




(c) overhead view



(d) view 'through' mirror

Figure 8: Detection of occluded object in mirror image

# Author Index