J. Little

# PROCEEDINGS


## CONFERENCE 1980

VICTORIA, B. C.

14, 15, 16 MAY 1980

# PROCEEDINGS OF THE THIRD BIENNIAL CONFERENCE OF THE CANADIAN SOCIETY FOR COMPUTATIONAL STUDIES OF INTELLIGENCE

# COMPTE-RENDU DE LA TROISIEME CONFERENCE BIENNALE DE LA SOCIETE CANADIENNE DES ETUDES D'INTELLIGENCE PAR ORDINATEUR

PRESENTED IN COOPERATION WITH
PRESENTE EN COOPERATION AVEC

Canadian Man-Computer Communications Society
Canadian Image Processing and Pattern Recognition Society
The University of Victoria
The University of Alberta

University of Victoria
Victoria, B.C.

14-16 May/Mai 1980

Copies of these Proceedings may be obtained by prepaid order to:

CIPS National Office
5th Floor
243 College Street
Toronto, Ontario
M5T 2Y1

Prepaid price: CSCSI/SCEIO, CM-CCS, and CIPPRS members $22.00
Others $25.00

## Chairman's Message

As Conference Chairman, I take great pleasure in welcoming you to Victoria and this Third Biennial CSCSI Conference. This conference is particularly significant in that it is being held in cooperation with the Canadian Man-Computer Communications Society and the Canadian Image Processing and Pattern Recognition Society. It is hoped that the arrangement is rewarding for all parties, and will continue in future conferences.

The program committee, chaired by L.K. Schubert and assisted by D.R. Barstow, J. de Kleer, A.K. Mackworth, T.A. Marsland, R.C. Perrault and J.R. Sampson have put together an outstanding program. I would like to thank them for their efforts. In addition, I am also grateful to N. Cercone, C. Suen and M. Wein for their contributions to the MCCS and IPPR portions of the program.

Local arrangements have been handled by the University of Victoria, Housing and Food Services, with assistance provided by H. Widdifield, A. Tweedale and C.H. Morgan. You have my gratitude for your most welcome assistance.

The assistance of NSERC for providing financial help with speakers' travel and the Province of British Columbia for a financial contribution to the Salmon Barbeque is also acknowledged.

Lastly, but not least, it is necessary to acknowledge the assistance of the University of Alberta: J. Tartar for allowing the use of the facilities at the U. of A. and particularly Sandra Wilkins for making sure that everything ran so smoothly.

Wayne A. Davis
Conference Chairman

## Message du président

En tant que président de la conférence, j'ai le grand plaisir de vous accueillir à Victoria, pour la troisième conférence biennale du SCEIO. Cette conférence est particulièrement importante en ce sens qu'elle a eu lieu en coopération avec la Canadian Man-Computer Communications Society et la Canadian Image Processing and Pattern Recognition Society. Nous espérons que toutes les organisations seront récompensées de cet arrangement et que celui-ci se poursuivra lois de futures conférences.

Le comité chargé du programme, présidé par L.K. Schubert, et assisté par D.R. Barstow, J. de Kleer, A.K. Mackworth, T.A. Marsland, R.C. Perrault et J.R. Sampson, a réalisé un travail remarquable. J'aimerais les remercier pour leurs efforts. En outre je suis très reconnaissant envers N. Cercone, C. Suen et M. Wein, pour leur contribution aux parties MCCS et IPPR du programme.

L'Université de Victoria, Housing and Food Services, avec l'assistance de H. Widdifield, A. Tweedale et C.H. Morgan, s'est chargée des arrangements locaux. Vous avez toute ma gratitude pour votre assistance qui a été des plus bienvenues.

Je mentionnerai aussi l'assistance du NSERC, qui a subvenu aux frais de voyage des conférenciers, et de la Province de Colombie brittainique, pour sa contribution financière au Salmon Barbeque.

Pour finir, il est nécessaire de faire part de l'assistance qui n'a pas été la moindre de l'Université d'Alberta: tout d'abord celle de J. Tartar pour avoir permis l'utilisation des locaux et de l'équipement de l'Université d'Alberta, et particulièrement celle de Sandra Wilkins pour avoir permis que tout aille pour le mieux.

Wayne A. Davis
Président de la Conférence

## OFFICERS OF THE THIRD BIENNIAL CONFERENCE OF THE CSCSI/SCEIO

General Chairman

Dr. Wayne A. Davis
Department of Computing Science
University of Alberta
Edmonton, Alberta
T6G 2H1

AI Program Chairman

Dr. Lenhart K. Schubert
Department of Computing Science
University of Alberta
Edmonton, Alberta
T6G 2H1

MCCS Program Chairman

Dr. Wayne A. Davis
Department of Computing Science
University of Alberta
Edmonton, Alberta
T6G 2H1

Proceedings Editor

Dr. Jeffrey R. Sampson
Department of Computing Science
University of Alberta
Edmonton, Alberta
T6G 2H1

Program Committee

Dr. David R. Barstow (Yale)
Dr. Johan de Kleer (Xerox PARC)
Dr. Alan K. Mackworth (U.B.C.)
Dr. T. Anthony Marsland (Alberta)
Dr. Raymond C. Perrault (Toronto)
Dr. Jeffrey R. Sampson (Alberta)

## OFFICERS OF THE CSCSI/SCEIO

President

Dr. John Mylopoulos
Department of Computer Science
University of Toronto
Toronto, Ontario
M5S 1A7

Vice-President

Dr. Alan K. Mackworth
Department of Computer Science
University of British Columbia
Vancouver, British Columbia
V6T 1W5

Treasurer

Dr. Wayne A. Davis
Department of Computing Science
University of Alberta
Edmonton, Alberta
T6G 2H1

Secretary

Dr. Steven W. Zucker
Dept. of Electrical Engineering
McGill University
Montreal, Quebec
H3C 3G1

TABLE OF CONTENTS

NATURAL LANGUAGE PROCESSING I



COMPUTER VISION

GAMES, PROBLEMS, AND SEARCH



NATURAL LANGUAGE PROCESSING II



EXPERT SYSTEMS II

MAN-COMPUTER COMMUNICATIONS



* No paper available

# CONSULTATION SYSTEMS FOR PHYSICIANS:
# The Role of Artificial Intelligence Techniques

Edward H. Shortliffe

Departments of Medicine and Computer Science
Heuristic Programming Project
Stanford University School of Medicine
Stanford, California 94305

## ABSTRACT

Computer systems for use by physicians have had limited impact on clinical medicine. When one examines the most common reasons for poor acceptance of medical computing systems, the potential relevance of artificial intelligence techniques becomes evident. This paper proposes design criteria for clinical computing systems and demonstrates their relationship to current research in knowledge engineering. The MYCIN System is used to illustrate the ways in which our research group has attempted to respond to the design criteria cited.

## 1. INTRODUCTION

Although computers have had an increasing impact on the practice of medicine, the successful applications have tended to be in domains where physicians have not been asked to interact at the terminal. Few potential user populations are as demanding of computer-based decision aids. This is due to a variety of factors which include their traditional independence as lone decision makers, the seriousness with which they view actions that may have life and death significance, and the overwhelming time demands that tend to make them impatient with any innovation that breaks up the flow of their daily routine.

This paper examines some of the issues that have limited the acceptance of programs for use by physicians, particularly programs intended to give advice in clinical settings. My goal is to present design criteria which may encourage the use of computer programs by physicians, and to show that AI offers some particularly pertinent methods for responding to the design criteria outlined. Although the emphasis is medical throughout, many of the issues occur in other user communities where the introduction of computer methods must confront similar barriers. After presenting the design considerations and their relationship to AI research, I will use our work with MYCIN to illustrate some of the ways in which we have attempted to respond to the acceptability criteria I have outlined.

### 1.1. The Nature Of Medical Reasoning

It is frequently observed that clinical medicine is more an "art" than a "science". This statement reflects the varied factors that are typically considered in medical decision making; any practitioner knows that well-trained experts with considerable specialized experience may still reach very different conclusions about how to treat a patient or proceed with a diagnostic workup.

One factor which may contribute to observed discrepancies, even among experts, is the tendency of medical education to emphasize the teaching of facts, with little formal advice regarding the reasoning processes that are most appropriate for decision making. There has been a traditional assumption that future physicians should learn to make decisions by observing other doctors in action and by acquiring as much basic knowledge as possible. More recently, however, there has been interest in studying the ways in which expert physicians reach decisions in hopes that a more structured approach to the teaching of medical decision making can be developed [Kassirer 1978, Elstein 1978].

---

[1] This article is based on a longer paper to be published as a book chapter by Academic Press [Shortliffe 1980].

[2] Dr. Shortliffe is recipient of research career development award LM00048 from the National Library of Medicine.

Computer programs for assisting with medical decision making have tended not to emphasize models of clinical reasoning. Instead they have commonly assigned structure to a domain using statistical techniques such as Bayes' Theorem [deDombal 1972] or formal decision analysis [Gorry 1973]. More recently a number of programs have attempted to draw lessons from analyses of actual human reasoning in clinical settings [Wortman 1972, Pauker 1976]. Although the other methodologies may lead to excellent decisions in the clinical areas to which they have been applied, many believe that programs with greater dependence on models of expert clinical reasoning will have heightened acceptance by the physicians for whom they are designed.

### 1.2. The Consultation Process

Accelerated growth in medical knowledge has necessitated greater subspecialization and more dependence upon assistance from others when a patient presents with a complex problem outside one's own area of expertise. Such consultations are acceptable to doctors in part because they maintain the primary physician's role as ultimate decision maker. The consultation generally involves a dialog between the two physicians, with the expert explaining the basis for advice that is given and the nonexpert seeking justification of points found puzzling or questionable. Consultants who offered dogmatic advice they were unwilling to discuss or defend would find that their opinions were seldom sought. After a recommendation is given, the primary physician generally makes the decision whether to follow the consultant's advice, seek a second opinion, or proceed in some other fashion. When the consultant's advice is followed, it is frequently because the patient's doctor has been genuinely educated about the particular complex problem for which assistance was sought.

Since such consultations are accepted largely because they allow the primary physician to make the final management decision, it can be argued that medical consultation programs must mimic this human process. Computer-based decision aids have typically emphasized only the accumulation of patient data and the generation of advice [Shortliffe 1979]. On the other hand, an ability to explain decisions may be incorporated into computer-based decision aids if the system is given an adequate internal model of the logic that it uses and can convey this intelligibly to the physician-user. The addition of explanation capabilities may be an important step towards effectively encouraging a system's use.

## 2. ACCEPTABILITY ISSUES

Studies have shown that many physicians are inherently reluctant to use computers in their practice [Startsman 1972]. Some researchers fear that the psychological barriers are insurmountable, but we are beginning to see systems that have had considerable success in encouraging terminal use by physicians [Watson 1974]. The key seems to be to provide adequate benefits while creating an environment in which the physician can feel comfortable and efficient.

Physicians tend to ask at least seven questions when a new system is presented to them:

(1) Is its performance reliable?

(2) Do I need this system?

(3) Is it fast and easy to use?

(4) Does it help me without being dogmatic?

(5) Does it justify its recommendations so that I can decide for myself what to do?

(6) Does use of the system fit naturally into my daily routine?

(7) Is it designed to make me feel comfortable when I use it?

Experience has shown that reliability alone may not be enough to insure system acceptance [Shortliffe 1979]; the additional issues cited here are also central to the question of how to design consultation systems that doctors will be willing to use.

## 3. DESIGN CRITERIA

The design considerations for systems to be used by physicians can be divided into three main categories: mechanical, epistemological, and psychological.

### 3.1. Mechanical Issues

It is clear that the best of systems will eventually fail if the process for getting information in or out of the

machine is too arduous, frustrating, or complicated. Someday physician-computer interaction may involve voice communication by telephone or microphone, but technology is likely to require manual interaction for years to come. Thus, careful attention to the mechanics of the interaction, the simplicity of the displays, response time, accessibility of terminals, and self-documentation, are all essential for the successful implementation of clinical computing systems.

### 3.2. Epistemological Issues

As has been discussed, the quality of a program's performance at its decision making task is a basic acceptability criterion. A variety of approaches to automated advice systems have been developed, and many perform admirably [Shortliffe 1979]. Thus the capturing of knowledge and data, plus a system for using them in a coherent and consistent manner, are the design considerations that have traditionally received the most attention.

Other potential uses of system knowledge must also be recognized, however. As has been noted, physicians often expect to be educated when they request a human consultation, and a computer-based consultant should also be an effective teaching tool. On the other hand, physicians would quickly reject a pedantic program that attempted to convey every pertinent fact in its knowledge base. Thus it is appropriate to design programs that convey knowledge as well as advice, but which serve this educational function only when asked to do so by the physician-user.

As has been mentioned, physicians also prefer to understand the basis for a consultant's advice so that they can decide for themselves whether to follow the recommendation. Hence the educational role of the consultation program can also be seen as providing an explanation or justification capability. When asked to do so, the system should be able to retrieve and display any relevant fact or reasoning step that was brought to bear in considering a given case. It is also important that such explanations be expressed in terms that are easily comprehensible to the physician.

Since it would be unacceptable for a consultation program to explain every relevant reasoning step or fact, it is important that the user be able to request justification for points found to be puzzling. Yet an ability to ask for explanations generally requires that the program be able to understand free-form queries entered by the user. A reasonable design consideration, then, is to attempt to develop an interface whereby simple questions expressed in English can be understood by the system and appropriately answered.

It is perhaps inevitable that consultation programs dealing with complex clinical problems will occasionally reveal errors or knowledge gaps, even after they have been implemented for ongoing use. A common source of frustration is the inability to correct such errors quickly so that they will not recur in subsequent consultation sessions. There is often a lapse of several months between "releases" of a system, with an annoying error recurring persistently in the meantime. It is therefore ideal to design systems in which knowledge is easily modified and integrated; then errors can be rapidly rectified once the missing or erroneous knowledge is identified. This requires a flexible knowledge representation and powerful methods for assessing the interactions of new knowledge with other facts already in the system.

Finally, the acquisition of knowledge can be an arduous task for system developers. In some applications the knowledge may be based largely on statistical data, but in others it may be necessary to extract judgmental information from the minds of experts. Thus another design consideration is the development of interactive techniques to permit acquisition of knowledge from primary data or directly from an expert without requiring that a computer programmer function as an intermediary.

### 3.3. Psychological Issues

The most difficult problems in designing consultation programs may be the frequently encountered psychological barriers to the use of computers among physicians [Startsman 1972, Croft 1972]. Many of these barriers are reflected in the mechanical and epistemological design criteria mentioned above. However, there are several other pertinent observations:

(1) It is probably a mistake to expect the physician to adapt to changes imposed by a consultation system.

(2) A system's acceptance may be greatly heightened if ways are identified to permit physicians to perform tasks that they have wanted to do but had previously been unable to do [Mesel 1976, Watson 1974].

(3) It is important to avoid premature introduction of a system while it is still "experimental".

(4) System acceptance may be heightened if physicians know that a human expert is available to back up the program when problems arise.

(5) Physicians are used to assessing research and new techniques on the basis of rigorous evaluations; hence novel approaches to assessing both the performance and the clinical impact of medical systems are required.

## 4. KNOWLEDGE ENGINEERING

In recent years the terms "expert systems" and "knowledge-based systems" have been coined to describe AI programs that contain large amounts of specialized expertise that they convey to system users in the form of consultative advice. The phrase "knowledge engineering" has been devised [Michie 1973] to describe the basic AI problem areas that support the development of expert systems. There are several associated research themes:

(1) Representation of Knowledge. A variety of methods for computer-based representation of human knowledge have been devised, each of which is directed at facilitating the associated symbolic reasoning and at permitting the codification and application of "common sense" as well as expert knowledge of the domain.

(2) Acquisition of Knowledge. Obtaining the knowledge needed by an expert program is often a complex task. In certain domains programs may be able to "learn" through experience or from examples, but typically the system designers and the experts being modelled must work closely together to identify and verify the knowledge of the domain. Recently there has been some early experience devising programs that actually bring the expert to the computer terminal where a "teaching session" can result in direct transfer of knowledge from the expert to the system itself [Davis 1979].

(3) Methods of Inference. Closely linked to the issue of knowledge representation is the mechanism for devising a line of reasoning for a given consultation. Techniques for hypothesis generation and testing are required, as are focusing techniques. A particularly challenging associated problem is the development of techniques for quantitating and manipulating uncertainty. Although inferences can sometimes be based on established techniques such as Bayes' Theorem or decision analysis, utilization of expert judgmental knowledge typically leads to the development of alternate methods for symbolically manipulating inexact knowledge [Shortliffe 1975].

(4) Explanation Capabilities. For reasons I have explained in the medical context above, knowledge engineering has come to include the development of techniques for making explicit the basis for recommendations or decisions. This requirement tends to constrain the methods of inference and the knowledge representation that is used by a complex reasoning program.

(5) The Knowledge Interface. There are a variety of issues that fall in this general category. One is the mechanical interface between the expert program and the individual who is using it; this problem has been mentioned for the medical user, and many of the observations there can be applied directly to the users in other knowledge engineering application domains. Researchers on these systems also are looking for ways to combine AI techniques with more traditional numerical approaches to produce enhanced system performance. There is growing recognition that the greatest power in knowledge-based expert systems may lie in the melding of AI techniques and other computer science methodologies [Shortliffe 1979].

Thus it should be clear that artificial intelligence, and specifically knowledge engineering, are inherently involved with several of the design considerations that have been suggested for medical consultation systems. In the next section I will discuss how our medical AI program has attempted to respond to the design criteria that have been cited.

## 5. AN EXAMPLE: THE MYCIN SYSTEM

Since 1972 our research group at Stanford University[1] has been involved with the development of computer-based consultation systems. The first was designed to assist physicians with the selection of antibiotics for patients with

---

[1]Several computer scientists, physicians, and a pharmacist have been involved in the development of the MYCIN System. These include J. Aikins, S. Axline, J. Bennett, A. Bonnet, B. Buchanan, W. Clancey, S. Cohen, R. Davis, L. Fagan, F. Rhame, C. Scott, W. vanMelle, S. Wraith, and V. Yu.

serious infections. That program has been termed MYCIN after the suffix utilized in the names of many common antimicrobial agents. MYCIN is still a research tool, but it has been designed largely in response to issues such as those I have described. The details of the system have been discussed in several publications [Shortliffe 1976, Davis 1977, Scott 1977] and may already be well known to many readers. Technical details will therefore be omitted here, but I will briefly describe the program to illustrate the ways in which its structure reflects the design considerations outlined above.

### 5.1. Knowledge Representation and Acquisition

All infectious disease knowledge in MYCIN is contained in packets of inferential knowledge represented as production rules [Davis 1976]. These rules were acquired from collaborating clinical experts during detailed discussions of specific complex cases on the wards at Stanford Hospital. More recently the system has been given the capability to acquire such rules directly through interaction with the clinical expert[1].

MYCIN currently contains some 600 rules that deal with the diagnosis and treatment of bacteremia (bacteria in the blood) and meningitis (bacteria in the cerebrospinal fluid). These rules are coded in INTERLISP [Teitelman 1978], but routines have been written to translate them into simple English so that they can be displayed and understood by the user. For example, one simple rule which relates a patient's clinical situation with the likely bacteria causing the illness is shown in Fig. 1. The strengths with which the specified inferences can be drawn are indicated by numerical weights, or certainty factors, that are described further below.

### 5.2. Inference Methods

#### 5.2.1. Reasoning Model

Production rules provide powerful mechanisms for selecting those that apply to a given consultation. In MYCIN's case the rules are only loosely related to one another before a consultation begins; the program selects the relevant rules and chains them together as it considers a particular patient. Two rules chain together if the action portion of one helps determine the truth value of a condition in the premise of the other. The resulting reasoning network, then, is created dynamically and can be seen as a model of one approach to the patient's problem.

---

RULE300

[This rule applies to all cultures and suspected infections, and is tried in order to find out about the organisms (other than those seen on cultures or smears) which might be causing the infection]

If: 1) The infection which requires therapy is meningitis, and
2) The patient does have evidence of serious skin or soft tissue infection, and
3) Organisms were not seen on the stain of the culture, and
4) The type of the infection is bacterial

Then: There is evidence that the organism (other than those seen on cultures or smears) which might be causing the infection is staphylococcus-coag-pos (.75) streptococcus-group-a (.5)

Figure 1 - A Sample MYCIN Rule

---

MYCIN's strategy in rule selection is goal-oriented. The program "reasons backwards" from its recognized goal of determining therapy for a patient. It therefore starts by considering rules for therapy selection, but the premise portion of each of those rules in turn sets up new questions or subgoals. These new goals then cause new rules to be invoked and a reasoning network is thereby developed. When the truth of a premise condition is best determined by asking the physician rather than by applying rules (e.g., to determine the value of a laboratory test), a question is displayed. The physician enters the appropriate response and the program continues to select additional rules. Once information on the patient is obtained, some rules will fail to be applicable; in this way the invoked applicable rules will provide a customized patient-specific reasoning network for the case under consideration.

Portions of a sample consultation session are shown in Fig. 2. The physician's responses are in capital letters and follow a double asterisk.

---

[1] This capability was implemented in rudimentary form in early versions of the system [Shortliffe 1976] but was substantially broadened and strengthened by Davis in his Teiresias program [Davis 1979].

Note that the physician can indicate that he does not know the answer to a question (Question 13) and MYCIN will proceed to do the best it can on the basis of the data available. Typographical or spelling errors are automatically corrected (Question 12) with the presumed meaning printed at the left margin so that the physician knows what assumption has been made. Note also that the physician's responses are generally single words.

```
--------PATIENT-538--------
1) Patient's name:
** PT538
2) Age:
** 45
3) Sex:
** MALE
4) Race:
** CAUC
=CAUCASIAN
5) Are there any cultures for Pt538
   which may be related to the present
   illness, and from which organisms
   have been grown successfully in
   the microbiology laboratory?
** NO
6) Are there any pending cultures?
** YES

   --------CULTURE-1--------
   7) From what site was the specimen
      for CULTURE-1 taken?
   ** CSF
   8) Please give the date and time
      when the pending csf culture
      (CULTURE-1) was obtained.
   ** 11/28/78
   9) Was a smear examined at the time
      that the specimen for the pending
      csf culture (CULTURE-1) was
      obtained?
   ** YES
   10) Were any organisms seen on the
      smear of the specimen submitted
      as the pending csf culture
      (CULTURE-1)?
   ** YES

      --------ORGANISM-1--------
      11) The stain (Gram, India ink,
         or Ziehl-Neelsen acid-fast) of
         ORGANISM-1:
      ** GRAMNEG
      12) Is ORGANISM-1 a rod or coccus
         (etc.):
      ** ROOD
=ROD
      13) What is the form of the
         individual organisms (e.g.
         Lancet-shaped for cocci,
         fusiform for rods, etc.)?
      ** ??
      Expected responses are: FUSIFORM,
      PLUMP or DUMBBELL-SHAPED
      Enter HELP for list of options.
      ** UNKNOWN
   14) Any other organisms noted on the
      smear?
   ** NO
```

Figure 2
The Beginning Of A Sample Consultation

In Fig. 3 the form of MYCIN's final conclusions and therapy recommendations are demonstrated. Note that the program specifies what organisms are likely to be causing the patient's infection and then suggests a therapeutic regimen appropriate for them. There are also specialized routines to calculate recommended drug doses depending upon the patient's size and kidney function.

```
My therapy recommendations will be
designed to treat for organisms that
either are very likely or, although less
likely, would have a significant effect
on therapy selection if they were
present. It is important to cover for
the following probable infection(s) and
associated organism(s):

INFECTION-1 is MENINGITIS

  + <ITEM-1>  E.COLI
  + <ITEM-2>  PSEUDOMONAS-AERUGINOSA
  + <ITEM-3>  KLEBSIELLA-PNEUMONIAE
  + <ITEM-4>  DIPLOCOCCUS-PNEUMONIAE

   41) Does Pt538 have a clinically
      significant allergic reaction to
      any antimicrobial agent?
   ** NO
   42) Do you have reason to suspect that
      Pt538 may have impaired renal
      function?
   ** NO
   43) Pt538's weight in kilograms:
   ** 70

[REC-1] My preferred therapy
        recommendation is:
   In order to cover for Items <1 2 3 4>:
        Give the following in combination:
        1) AMPICILLIN
        Dose: 3.5g (28.0 ml) q4h IV
              [calculated on basis of 50
              mg/kg]
        2) GENTAMICIN
        Dose: 119 mg (3.0 ml, 80mg/2ml
              ampule) q8h IV [calculated
              on basis of 1.7 mg/kg] plus
              consider giving 5 mg q24h
              intrathecally

Since high concentrations of penicillins
can inactivate aminoglycosides, do not mix
these two antibiotics in the same bottle.
```

Figure 3
Example of MYCIN's Recommendations

### 5.2.2. Management of Uncertainty

The knowledge expressed in a MYCIN rule is seldom definite but tends to include "suggestive" or "strongly suggestive" evidence in favor of a given conclusion. In order to combine evidence regarding a single hypothesis but derived from a number of different rules, it has been necessary to devise a numeric system for capturing and representing an expert's measure of belief regarding the inference stated in a rule. Although this problem may at first seem amenable to the use of conditional probabilities and Bayes' Theorem, a probabilistic model fails to be adequate for a number of reasons we have detailed elsewhere [Shortliffe 1975]. Instead we use a model that has been influenced by the theory of confirmation, and have devised a system of belief measures known as certainty factors. These numbers lie on a -1 to +1 scale with -1 indicating absolute disproof of an hypothesis, +1 indicating its proof, and 0 indicating the absence of evidence for or against the hypothesis (or equally weighted evidence in both directions). The relationship of the model to formal probability theory and the methods for combining evidence from diverse sources (rules and user estimates) have been described [Shortliffe 1975]. Although the system has served us well to date, it does have several recognized inadequacies [Adams 1976] and can only be seen as a first step towards the development of a coherent theory for the management of uncertainty in complex reasoning domains.

## 5.3. Explanation

Perhaps the greatest advantage of the rules used in MYCIN is the way in which they facilitate the development of mechanisms for explaining and justifying system performance. These capabilities also contribute greatly to MYCIN's educational role [Clancey 1979]. The explanation program has two options, one of which is limited in scope but fast and easy to use. This option is used during a consultation session and allows examination of the reasoning steps that have generated a specific question. The second option is more powerful but it involves language processing and is therefore less reliable and more time consuming. This question-answering capability is designed to allow full access to all system knowledge, both static (i.e., facts and rules obtained from experts) and dynamic (i.e., conclusions reached by the program for the specific consultation session). Each system is briefly described below.

### 5.3.1. Answers To Questions About The Reasoning Status

During the consultation, MYCIN may ask questions which the physician finds puzzling, either because they appear irrelevant or because the details of their applicability are unclear. When this occurs, the physician can digress momentarily and seek an explanation of the question[1]. Instead of answering the question MYCIN has asked, the user enters the word "WHY" and the program responds with a translation of the rule which

---

```
35) Is this a hospital-acquired
    infection?
** WHY

[i.e. WHY is it important to determine
whether the infection was acquired while
the patient was hospitalized?]

[2.0] This will aid in determining the
organisms (other than those seen on
cultures or smears) which might be causing
the infection.
   It has already been established that
        [2.1] the infection which requires
              therapy is meningitis, and
        [2.2] organisms were not seen on
              the stain of this pending
              csf culture, and
        [2.3] the type of the infection is
              bacterial, and
        [2.4] neurosurgery is not one of
              the relevant items from the
              history of pt538, and
        [2.5] the age of pt538 is greater
              than 10 days
     Therefore, if
        [2.6] the infection was acquired
              while the patient was
              hospitalized
     then:
           there is evidence that the
           organism (other than those
           seen on cultures or smears)
           which might be causing the
           infection is e.coli (.75)
           staphylococcus-coag-pos (.3)
           pseudomonas-aeruginosa (.3)
           klebsiella-pneumoniae (.5)

[back to question 35...]
**
```

Figure 4
Example of the WHY Command

---

[1]The mechanisms for examining the reasoning status using "WHY" and "HOW" commands were largely the work of Davis in his Teiresias program [Davis 1979]. The techniques he developed are general in their applicability and have been implemented in nonmedical domains as well.

generated the question. An example of this feature is shown in Fig. 4. Note that MYCIN begins its response by phrasing in English its understanding of the "WHY question" asked by the physician. It then displays the relevant rule, specifying which conditions in the premise are already known to be true and which conditions remain to be investigated. In many cases this single rule displayed is an adequate explanation of the current line of reasoning and the physician can then proceed with the consultation by answering the question.

The user can alternatively continue to investigate the current reasoning by repeating the "WHY" command several times. Each additional "WHY" is interpreted by MYCIN as a request for display of the next rule in the current reasoning chain. For example, in Fig. 4 another "WHY" would be equivalent to asking "Well then, why are you trying to determine the organisms which might be causing the infection?" After responding to each "WHY", MYCIN returns to the current question and awaits the physician's response.

The "HOW" command is used in conjunction with the "WHY" command. Note that MYCIN places a unique number before each condition in the premise of a displayed rule (Fig. 4). If the user does not understand how a specified condition was met (or will be met), the "HOW" command, followed by the identifying number of the premise clause in question, can be used. Hence "HOW" requires an argument as shown in Fig. 5. This example follows immediately on the "WHY" example in Fig. 4; thus the argument "2.3" refers to the third condition in the premise of RULE545 in Fig. 4 and "2.5" refers to the fifth. MYCIN's responses to "HOW" commands are self-explanatory. Note also that components of all explanations are given identifiers so that, in Fig. 5, the command "HOW 3.1" would have resulted in a display of RULE526.

Other features of the capability to assess the reasoning status during a consultation are described elsewhere [Davis 1977, Scott 1977, Davis 1979]. Two points require emphasis here, however. First, these features require no English language processing since the user gives simple single-word commands; hence the capability is fast, efficient, and reliable. On the other hand, questions are limited to the immediate reasoning environment surrounding the question asked by MYCIN. Hence general access to other system knowledge or other conclusions previously made by the program is not possible. To provide a more general explanation capability, MYCIN must be able to respond to questions expressed in simple English as described below.

---

```
   [back to question 35...]
   ** HOW 2.3

[i.e. HOW was it established that the type
      of the infection is bacterial?]

The following rules concluded about the
      likelihood that the type of the
      infection is bacterial:

              Rule          Certainty
     [2.7]    RULE148       (.19)
     [2.8]    RULE500       (.51)
     [2.9]    RULE501       (.95)
     [3.0]    RULE502       (.97)
     [3.1]    RULE526       (.98)
     [3.2]    RULE504       (.97)
     [3.3]    RULE524       (.98)

Thus, it has been established that the
     type of the infection is bacterial.

   [back to question 35...]
   ** HOW 2.5

[i.e. HOW was it established that the age
     of pt538 is greater than 10 days?]

In answer to question 2 you said that the
     age of pt538 is 34 years. Thus, it
     has been established that the age of
     pt538 is greater than 10 days.

   [back to question 35...]
   **
```

Figure 5
Examples of the HOW Command

---

### 5.3.2. Answers To Questions About The Consultation And Knowledge Base

MYCIN's general question-answering feature is automatically made available to the user at the end of each consultation session. After a therapy recommendation has been given, the physician may still have several questions regarding details of the consultation and the basis for diagnostic and therapeutic advice. Fig. 6 shows an example of a simple English question which MYCIN understands and attempts to answer. Our approach to natural language understanding is very simplistic [Scott 1977] but often adequate if questions are kept short without subordinate clauses. In many cases questions are best answered simply by retrieving relevant rules and displaying them in a format suitable for the question being asked. In Fig. 6 MYCIN offers a detailed explanation of how it had decided that the patient might be infected with an E. coli. All rules are specified by

number so that the physician can have them displayed if full details are desired.

---

** WHY DID YOU TREAT E.COLI?

ITEM-1 is E.coli associated with meningitis. I decided to treat ITEM-1 on the basis of clinical evidence only.

The following rules concluded about: the likelihood that e.coli is one of the organisms (other than those seen on cultures or smears) which might be causing the infection

RULE545 was used to conclude that e.coli is one of the organisms (other than those seen on cultures or smears) which might be causing the infection (.74).
RULE557 was used to conclude that e.coli is one of the organisms (other than those seen on cultures or smears) which might be causing the infection (.93).

The parameters most frequently used in these rules:
the infection which requires therapy
whether a smear of the culture was examined
whether organisms were seen
the type of the infection

Figure 6
Response To A Free Text Question

---

Fig. 7 demonstrates the way in which a physician can ask general questions of the knowledge base (static knowledge); neither the questions nor MYCIN's responses refer to any specific consultation session. Note that MYCIN has a sense of its limitations and makes it clear that it has no knowledge specific to pneumonia. Also, a large dictionary includes synonyms and hence permits MYCIN to recognize that "pneumococcal" refers to "pneumococcus", which in turn is the same as "diplococcus-pneumoniae".

Despite their utility, the language understanding features demonstrated in Figs. 6 and 7 have significant limitations. The associated computer code is large, complex, and very slow. Furthermore, we have found that it takes novice users several sessions before they learn the best ways to phrase questions so that MYCIN will interpret them properly.

## 5.4. Evaluating MYCIN

As work on MYCIN has progressed, we have begun to devise techniques for analyzing formally the system's

---

** HOW DO YOU DECIDE THAT AN ORGANISM IS AN E.COLI?
The following rules conclude about: the likelihood that the identity of the organism is E.coli
555, 304, 243, 3
Which of the rules do you wish to see?
** NONE

** WHAT IS THE TREATMENT FOR PNEUMOCOCCAL PNEUMONIA?
The MYCIN system currently can treat the following infections: BACTEREMIA MENINGITIS CYSTITIS. Because the knowledge base is incomplete, all other infections will be treated like bacteremia.

For treatment of DIPLOCOCCUS-PNEUMONIAE in BACTEREMIA, the drugs of choice (ranked by relative efficacy, assuming sensitivity) are:
1st choice:
PENICILLIN or AMPICILLIN
2nd choice:
NAFCILLIN, CEPHALOTHIN, ERYTHROMYCIN, or CLINDAMYCIN
3rd choice:
TETRACYCLINE, CHLORAMPHENICOL, or VANCOMYCIN

** WHAT DOSE OF ERYTHROMYCIN DO YOU RECOMMEND?
When ERYTHROMYCIN is given for BACTEREMIA, the recommended dosage is:
if age is > 12 yrs then 12 mg/kg q6h IV
else if age > .08 yrs then 5 mg/kg q6h IV
else if age > .02 yrs then 5 mg/kg q12h IV

Figure 7
Requests For MYCIN's General Knowledge

---

performance. It must be emphasized, however, that the decision making performance is only one aspect of overall system acceptability; as I have discussed, many of the most significant problems occur when attempts are made to encourage physicians to use a program, even after it has been shown to reach good decisions.

The details of the evaluation studies will not be presented here[1], but a number of specific points are of interest. First any evaluation is difficult because there is so much difference of opinion in this domain, even among experts. Hence, it is unclear how to select a "gold standard" by which to measure the system's performance.

---

[1] See [Yu 1979a] for the details of the bacteremia evaluation, and [Yu 1979b] for the data on MYCIN's performance selecting therapy for patients with meningitis.

Actual clinical outcome cannot be used because each patient of course is treated in only one way and because a poor outcome in a gravely ill patient cannot necessarily be blamed on the therapy that had been selected.

Second, although MYCIN performed at or near expert level in almost all cases, the evaluating experts in one study [Yu 1979a] had serious reservations about the clinical utility of the program. It is difficult to assess how much of this opinion is due to actual inadequacies in system knowledge or design and how much is related to inherent bias against any computer-based consultation aid. In a subsequent study we attempted to eliminate this bias from the study by having the evaluators unaware of which recommendations were MYCIN's and which came from actual physicians [Yu 1979b]. In that setting MYCIN's recommendations were uniformly judged preferable to, or equivalent to, those of five infectious disease experts who recommended therapy for the same patients.

Finally, those cases in which MYCIN has tended to do least well are those in which serious infections have been simultaneously present at sites in the body about which the program has been given no rules. It is reasonable, of course, that the program should fail in areas where it has no knowledge. However, a useful antimicrobial consultation system must know about a broad range of infectious diseases, just as its human counterpart does. Even with excellent performance managing isolated bacteremias and meningitis, the program is therefore not ready for clinical implementation.

There will eventually be several important questions regarding the clinical impact of MYCIN and systems like it. Are they used? If so, do the physicians follow the program's advice? If so, does patient welfare improve? Is the system cost effective when no longer in an experimental form? What are the legal implications in the use of, or failure to use, such systems? The answers to all these questions are years away for most consultation systems, but it must be recognized that all these issues are ultimately just as important as whether the decision making methodology manages to lead the computer to accurate and reliable advice.

## 6. CONCLUSION

Although I have asserted that AI research potentially offers solutions to many of the important problems confronting researchers in computer-based clinical decision making, the field is not without its serious limitations. However, AI has reached a level of development where it is both appropriate and productive to begin applying the techniques to important real world problems rather than purely theoretical issues. The difficulty lies in the fact that such efforts must still dwell largely in research environments where short term development of systems for service use is not likely to occur.

It is also important to recognize that other computational techniques may meld very naturally with AI approaches as the fields mature. Thus we may see, for example, direct links between AI methods and statistical procedures, decision analysis, pattern recognition techniques, and large databanks. As researchers in other areas become more familiar with AI, it may gradually be brought into fruitful combination with these alternate methodologies. The need for physician acceptance of medical consultation programs is likely to make AI approaches particularly attractive, at least in those settings where hands-on computer use by physicians is desired or necessary. This paper has attempted to explain why the wedding of AI and medical consultation systems is a natural one and to show, in the setting of the MYCIN system, how one early application has responded to design criteria identified for a user community of physicians.

## REFERENCES


Adams, J. B. "A probability model of medical reasoning and the MYCIN model." Math. Biosci. 32,177-186 (1976).

Clancey, W.J. Transfer of Rule-Based Expertise Through a Tutorial Dialogue. Dictoral dissertation, Stanford University, September 1979. Technical memo STAN-CS-79-769.

Croft, D. J. "Is computerized diagnosis possible?" Comp. Biomed. Res. 5,351-367 (1972).

Davis, R. and King, J. "An overview of production systems." In Machine Representation of Knowledge (E. W. Elcock and D. Michie, eds.), New York: Wiley, 1976.

Davis, R., Buchanan, B. G., and Shortliffe, E. H. "Production rules as a representation for a knowledge-based consultation system." Artificial Intelligence 8,15-45 (1977).

Davis, R., "Interactive transfer of expertise: acquisition of new inference rules." Artificial Intelligence, 12,121-157 (1979).

deDombal, F. T., Leaper, D. J., Staniland, J. R., et al. "Computer-aided diagnosis of acute abdominal pain." Brit. Med. J. 2,9-13 (1972).

Elstein, A. S., Shulman, L. S., and Sprafka, S. A. Medical Problem Solving: An Analysis of Clinical Reasoning. Cambridge, Mass.: Harvard Univ. Press, 1978.

Gorry, G. A., Kassirer, J. P., Essig, A., and Schwartz, W. B. "Decision analysis as the basis for computer-aided management of acute renal failure." Amer. J. Med 55,473-484 (1973).

Kassirer, J. P. and Gorry, G. A. "Clinical problem solving: a behavioral analysis." Anns. Int. Med. 89,245-255 (1978).

Mesel, E., Wirtschafter, D. D., Carpenter, J. T., et al. "Clinical algorithms for cancer chemotherapy - systems for community-based consultant-extenders and oncology centers." Meth. Inform. Med. 15,168-173 (1976).

Michie, D. "Knowledge engineering." Cybernetics 2,197-200 (1973).

Pauker, S. G., Gorry, G. A., Kassirer, J. P., and Schwartz, W. B. "Towards the simulation of clinical cognition: taking a present illness by computer." Amer. J. Med. 60:981-996 (1976).

Scott, A. C., Clancey, W., Davis, R., and Shortliffe, E. H. "Explanation capabilities of knowledge-based production systems." Amer. J. Computational Linguistics, Microfiche 62, 1977.

Shortliffe, E. H. and Buchanan, B. G. "A model of inexact reasoning in medicine." Math. Biosci. 23,351-379 (1975).

Shortliffe, E. H. Computer-Based Medical Consultations: MYCIN, New York: Elsevier/North Holland, 1976.

Shortliffe, E. H. Buchanan, B. G. and Feigenbaum, E. A. "Knowledge engineering for medical decision making: a review of computer-based clinical decision aids." PROCEEDINGS of the IEEE, 67,1207-1224 (1979).

Shortliffe, E.H. "Medical consultation systems: designing for doctors." In Communication With Computers (M. Sime and M. Fitter, eds.), Academic Press, London, 1980 (in press).

Startsman, T. S. and Robinson, R. E. "The attitudes of medical and paramedical personnel towards computers." Comp. Biomed. Res. 5,218-227 (1972).

Teitelman, W. INTERLISP Reference Manual, XEROX Corporation, Palo Alto, Calif. and Bolt Beranek and Newman, Cambridge, Mass., October 1978.

Watson, R. J. "Medical staff response to a medical information system with direct physician-computer interface." MEDINFO 74, pp. 299-302, Amsterdam: North-Holland Publishing Company, 1974.

Wortman, P. M. "Medical diagnosis: an information processing approach." Comput. Biomed. Res. 5,315-328 (1972).

Yu, V. L. Buchanan, B. G. Shortliffe, E. H. et al. "Evaluating the performance of a computer-based consultant." Comput. Prog. Biomed. 9,95-102 (1979a).

Yu, V. L. Fagan, L. M. Wraith, S. M. et al. "Computerized consultation in antimicrobial selection - a blinded evaluation by experts." J. Amer. Med. Assoc. 242,1279-1282 (1979b).

# Finding Common Paths as a Learning Mechanism[1]

Pat Langley
Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

## ABSTRACT

In this paper I describe ACTG, a production system language designed to model the learning process. The language incorporates a propositional network for storing declarative knowledge, and employs five interacting learning mechanisms to generate new condition-action rules on the basis of experience. I focus on one of these, a technique for finding common paths through the propositional network. The method is applied to learning simple procedures for algebra and integration, inducing meaning to sentence mappings, and discovering complex functions.

## 1. Introduction

In recent years, Artificial Intelligence researchers have developed a number of programs capable of expert behavior in limited domains (e.g., DENDRAL [3], INTERNIST [10]). Each of these systems has drawn upon a large store of knowledge about its specialty, and for this reason considerable effort has been involved in their construction. More recently, researchers like Mitchell [8] and Michalski [7] have considered how such systems might acquire their own knowledge bases.

In this paper I focus on the task of learning procedures and other forms of rules from examples. I begin with an example of such learning in the domain of algebra. Next I describe a programming language, ACTG, designed for the construction of learning systems. After this, I consider one of ACTG's learning mechanisms in detail, the discovery of common paths through a propositional network. Finally, I consider the generality of this technique, along with some of its limitations.

$$3x + 2 = 8$$
$$3x + 2 - 2 = 8 - 2$$
$$3x = 6$$
$$3x / 3 = 6 / 3$$
$$x = 2$$

Table 1. Sample solution to an algebra problem.

[1]This work was supported in part by NSF Grant SPI-7914852, in part by NSF Grant IST 7918266, and in part by ARPA Grant F44620-73 C0074. I would like to thank John Anderson, Paul Kline, H. A. Simon, and Robert Neches for discussions leading to the ideas presented in this paper.

## 2. Learning Algebra from Examples

Imagine a student presented with the worked-out solution to an algebra problem like $3x + 2 = 8$, as presented in Table 1. Given this information, the student might formulate a set of specific condition-action rules like that shown in Table 2. Each of these rules corresponds to a step in the solution process, and the possession of these rules would let the student resolve the sample problem. Next, imagine that an analogous problem like $2x + 1 = 9$ is presented, along with its solutions. The solution steps for these two problems are isomorphic; they differ only in the numbers which are used. Thus, this example would lead to a second set of rules like those in Table 2, structurally similar but containing different numbers.

**If you have just written $3x + 2 = 8$,**
**then write $3x + 2 - 2 = 8 - 2$.**

**If you have just written $3x + 2 - 2 = 8 - 2$,**
**then write $3x = 6$.**

**If you have just written $3x = 6$,**
**then write $3x / 3 = 6 / 3$.**

**If you have just written $3x / 3 = 6 / 3$,**
**then write $x = 2$.**

Table 2. Specific rules learned from the sequence in Table 1.

At this point, the student might realize that he has two very similar sets of rules, and attempt to generalize. The result might be a set of rules like that shown in Table 3. Here the numbers which differed in the specific rule sets have been replaced by variables which can match any number; all variables start with the letter v. Note that two of the general rules include conditions which were not found in their specific counterparts. These conditions are necessary to determine the values of some of the variables found in the action sides. The mechanism leading to the discovery of such conditions is discussed in Section 4.

The interested reader should see Neyes [9] for a fuller treatment of the task of learning algebra from examples. Although I will be using examples from algebra throughout the paper, that domain is not the focus of the present research. Rather, the goal is to develop a small set of general mechanisms which can lead to learning in many domains. As a result, these techniques will be data-driven instead of model-driven, and will have a more syntactic than semantic flavor. Section 5 is devoted to testing the generality of one of these methods.

---

**If you have just written v1 x + v2 = v3,**
**then write v1 x + v2 - v2 = v3 - v2.**

**If you have just written v1 x + v2 - v2 = v3 - v2,**
**and v2 + v4 = v3,**
**then write v1 x = v4.**

**If you have just written v1 x = v4,**
**then write v1 x / v1 = v4 / v1.**

**If you have just written v1 x / v1 = v4 / v1,**
**and v1 × v5 = v4,**
**then write x = v5.**

---

Table 3. More general rules for solving algebra problems.

## 3. An Overview of ACTG

ACTG is a production system language designed to model learning processes. Below I summarize the features that ACTG shares with other production system languages. After this I discuss some of its unique characteristics, including its propositional network, its conflict resolution scheme, and its automatic learning mechanisms.

### 3.1. The ACTG Production System

ACTG has a number of similarities to earlier production system languages. A program is stated as a set of condition-action rules, or **productions**, like those in Tables 2 and 3. If the conditions of one of these rules are true, then the associated actions may be carried out. These conditions and actions may refer to descriptions of the environment, or to purely internal structures like goals. A production system operates in **cycles**. Each cycle a true production is selected and its actions are applied; these actions change the state of the world so that other rules become true, and the system iterates.

The ACTG language draws on Forgy's [4] techniques for the efficient storage and matching of production conditions. These are stored in a discrimination network which takes advantage of common tests for the presence of symbols and for shared variables. A separate discrimination net is created for use by the generalization process discussed below. Approximately half of ACTG consists of MACLISP code borrowed from Forgy's [5] OPS4 production system language.

### 3.2. The Propositional Network

ACTG differs from many of its predecessors by incorporating a long-term **propositional network**. While productions are used primarily to represent procedural knowledge, propositions are used to store declarative or factual knowledge. Although ACTG propositions may be arbitrary list structures, all examples in this paper will be simple lists. A proposition is composed of two types of symbols: **content elements** and **syntax elements**. For example, a number fact might be stored in a proposition like **3 + 2 = 5**, where **3**, **2**, and **5** are content elements and **+** and **=** are syntax elements. The user may specify different syntax elements for different domains; all non-syntactic elements are treated as content elements.

During a given cycle, some subset of the propositional network may be considered **active**. It is against this set of active propositions that the conditions of productions are tested. The activation of a proposition decays by a user-modifiable factor after each cycle. If its activation drops below a user-controlled threshold, it is deleted from active memory. When this happens, all matches relying on the proposition are removed from the set of potentially applicable rules, or **conflict set**.

Once a proposition has been stored in the network, it may be retrieved through a process of **spreading activation**. When a proposition is added to active memory, its activation is divided equally among its various content elements. All propositions containing a given element are found, and the activation associated with that element is divided among these propositions according to their **trace strengths**. The trace strength of a proposition is a function of the number of times it has been added to active memory. If the activation spread to a proposition in this manner exceeds the threshold mentioned above, it is added to memory and contributes to the match process.

### 3.3. Conflict Resolution

In many cases, more than one production will be matched by the contents of active memory, or a single production may have multiple true **instantiations**. Normally a set of domain-independent heuristics are used for determining the order in which these instantiations are applied. McDermott and Forgy [6] have considered a number of such **conflict resolution rules**. Early languages simply ordered productions and selected the true rule with the highest priority. More recent languages, such as Forgy's OPS4, have used **recency** information to order the conflict set.

ACTG uses the related strategy of computing the **total activation** of an instantiation. This is found by summing the activations of the propositions matching a production's conditions. Although similar to recency-based approaches, this heuristic also provides a preference for special case rules, since productions with more conditions will tend to have a higher total activation. ACTG also associates a **strength** with each production, which may change over time. On each cycle, the **product** of the strength and total activation of each instantiation is calculated, and the actions of the instantiation with the highest product are carried out.

### 3.4. The ACTG Learning Mechanisms

One advantage of the production system formalism is the simplicity and relative independence of the condition-action rules. This suggests that production systems should be well-suited for modeling incremental learning. ACTG draws on five interacting learning mechanisms in addressing this issue, most of which it shares with Anderson, Kline, and Beasley's [1] ACTF language. The most basic of these is the **designation** process. This allows the creation of a new production as one of the actions of an existing production. Usually, the production responsible for designation is a very general one, while the productions it creates are quite specific. The productions shown in Table 2 are examples of rules which would be built through the designation process.

A second mechanism leads to the **strengthening** of productions through firing, or upon their recreation through any of the learning processes. Since the strengths of productions play a major role in conflict resolution, productions which have proved useful in the past will tend to be preferred, as will those which have been relearned many times. Since some rules may be in error, an inverse process of **weakening** can occur when a production is identified as the source of a bad result. In addition, the discovery of an error can lead to a call on the **discrimination** process. Here the recent firings of the responsible production are examined. If some proposition is found to have been present at the successful firings, but absent at the errorful ones, this proposition is added as an extra condition in a new, more conservative version of the rule. Taken together, the strengthening/weakening process and the discrimination mechanism give learning systems written in ACTG the potential for error recovery, though this aspect of the language has not been explored in detail.

A fourth process leads to **generalizations** like the first and third rules in Table 3. ACTG creates such a general rule whenever a new production is added that is isomorphic, or identical in structure, to an existing production. This general production has variables in place of those constant terms which differed in the two specific rules. Occasionally, there may be more than one mapping between two rules, so that multiple generalizations can result. Syntactic elements are never replaced by variables, and must be identical for two rules to be considered isomorphic.

Now consider the second and fourth rules in Table 2, and suppose two rules are added which are isomorphic to them. As described above, the generalization process would lead to a pair of general rules with unbound variables in their action sides. In other words, there would be variables in the actions which are not mentioned in any of the conditions. The application of such a rule would leave ACTG in confusion, not knowing what actions to take. A fifth learning mechanism, designed to deal with such situations, is described in the next section.

## 4. Finding Common Paths

Upon finding a generalization with unbound action terms, ACTG looks for additional conditions which will determine the values of those terms. This problem can be cast as a search for analogous paths through the propositional network, connecting the actions of the specific productions to their respective conditions. Vere [11] has discussed a similar notion of finding recurring relations. In this section I consider the details of the path-finding process. Below I provide some definitions which should clarify later discussions. Next I present some techniques for constraining the search for common paths. Finally, I discuss the merits of two alternate organizations for this search.

### 4.1. A Definition of Analogous Paths

One of the central notions in propositional networks is **adjacency**. Two propositions may be considered adjacent if they share one or more non-syntactic symbols. For example, the propositions 5 + 4 = 9 and 4 + 3 = 7 are adjacent only through the symbol 4, provided one treats the symbols + and = as syntactic elements. Adjacency is important in determining the spread of activation, but it is also necessary for the path-finding process. A **path** through the propositional network may be defined as a sequence of one or more propositions, in which each proposition is adjacent to its predecessor in the sequence. For example, the sequence [5 + 4 = 9, 4 + 3 = 7, 3 × 2 = 6] would constitute a path of length three; I will use this notation for paths throughout the rest of the paper.

Upon examining various pairs of adjacent propositions, one finds that some pairs are adjacent in **analogous** ways. For example, the propositions 3 + 2 = 5 and 2 + 4 = 6 are adjacent in an analogous manner to the pair considered above. This is because the symbol 2 occurs in the same positions in these propositions as 4 occurs in the earlier ones; I will represent a pair of elements x and y which occupy analogous positions as [x, y]. Similarly, one can define **analogous paths** as a pair of paths through the network which have analogous adjacencies between their successive propositions. For example, the path [3 + 2 = 5, 2 + 4 = 6, 4 × 2 = 8] is analogous to the path shown above. Finally, one can compute a **generalized path** in which the differing terms in two analogous paths have been replaced by variables. The generalized path for the above pair would be [v1 + v2 = v3, v2 + v4 = v5, v4 × 2 = v6], where variables start with a v.

### 4.2. Constraining the Search

In order to find additional conditions on a rule with unbound action terms, ACTG searches for analogous paths as defined in the last section. The search originates from those symbols which differ in the conditions of the specific rules, and continues until analogous connections have been found to all of the actions leading to unbound variables. Once a set of analogous paths have been discovered, the resulting generalized paths are added to the general rule as an extra set of conditions.

Naturally, paths containing multiple propositions must sometimes be handled. If the propositional network has a high connectivity, as it does in the storage of arithmetic facts, then the branching factor of the search may be quite high. In addition, multiple action terms may have to be connected, so that many paths must be found. ACTG uses four techniques for directing its search down useful paths. I discuss these in some detail below.

- *ACTG extends analogous paths only through symbols which differ.* Suppose the system is considering the analogous paths [1 + 2 = 3] and [1 + 5 = 6], where the original conditions differed by the pair [3, 6]. Then extending the paths through [2, 5] into [1 + 2 = 3, 2 × 3 = 6] and [1 + 5 = 6, 5 × 6 = 30] would be allowed, while going through the pair [1, 1] to [1 + 2 = 3, 1 + 4 = 5] and [1 + 5 = 6, 1 + 7 = 8] would be prohibited.

- *ACTG extends paths only through symbols not bound earlier in those paths.* Suppose the system is considering the length two paths given above. The newest propositions differ in all three of their content elements, giving the pairs [2, 5], [3, 6], and [6, 30]. However, since the first two of these are mentioned earlier along the paths, extensions may occur only through the last pair.

- *ACTG searches for generalized paths which lead to unique matches.* For example, suppose the original conditions differed by the pair [2, 3], while the actions differed by the pair [6, 9]. Consider the two analogous paths of length one, [2 + 4 = 6] and [3 + 6 = 9]. These would lead to the generalized path [v1 + v2 = v3], where only v1 was bound in the original condition. Given a value for v1 (say 2), this condition will match any of a number of propositions (e.g., 2 + 1 = 3 or 2 + 5 = 7), so that the action to be taken is not uniquely determined. Based on this knowledge, ACTG would reject this particular pair of analogous paths.

- *ACTG considers only propositions which were in active memory* at the creation time of the production with which the path is associated. Since production designation is usually based on data found in active memory, activation has a chance to spread from these data before a production is created. The more closely a proposition is associated with the data leading to a new production, the more likely it will be activated. Thus, this is a useful heuristic for filtering out irrelevant information.

Taken together, these techniques should reduce the combinatorics inherent in the search for analogous paths, as well as limiting the search to paths which have some usefulness to the system.

### 4.3. Two Search Strategies

Although I have proposed a number of techniques for eliminating certain paths from consideration, a large number of paths remain which must be systematically explored. The two basic strategies for exhaustive search, **depth-first** and **breadth-first** search, presented themselves as likely candidates. Routines for both strategies were implemented in ACTG, and experiments were carried out. The main attraction of depth-first search was its ease of implementation in a recursive language such as MACLISP. In addition, the **activation level** of propositions provided a heuristic for directing the search down potentially useful paths. This mechanism was implemented in high hopes. Although I expected considerable search and backup to be involved, I fully expected the appropriate paths to be found.

However, this was not the case. Consider two specific but isomorphic algebra rules: $3x + 2 = 8 \rightarrow 3x = 6$ and $2x + 1 = 9 \rightarrow 2x = 8$. The only analogous unbound pair of action symbols is [6, 8], while the analogous pairs of condition symbols are [3, 2], [2, 1], and [8, 9]. Using a depth-first search strategy with a maximum depth of five propositions, ACTG successfully found an analogous pair of paths connecting the pair [8, 9] to the pair [6, 8]. These paths were [3 + 5 = 8, 2 + 3 = 5, 2 × 2 = 4, 2 + 4 = 6] and [3 + 6 = 9, 3 + 3 = 6, 2 × 3 = 6, 2 + 6 = 8].

Unfortunately, the resulting rule is completely spurious, even though all of the constraints mentioned in the last section were used in finding it. The rule may be stated as:

> **If you have written** v1 x + v2 = v3,
> **and** 3 + v4 = v3, **and** v5 + 3 = v4,
> **and** 2 × v5 = v6, **and** 2 + v6 = v7,
> **then write** v1 x = v7

or, in a somewhat simpler form, $v1\,x + v2 = v3 \rightarrow v1\,x = 2 \times (v1 - 6) + 2$. This rule predicts the correct actions for the two examples on which it is based. However, if the new rule is given an entirely new example (e.g., $5x + 2 = 7$), it may give an incorrect answer (e.g., $5x = 4$). Changing the order in which propositions were considered led to a different but equally spurious rule.

Fortunately, the breadth-first search scheme was more successful. This considers all analogous paths of length one first, then extends these into length two paths, and so on. Since in this case a length one path pair existed ([6 + 2 = 8] and [8 + 1 = 9]), longer paths were never examined. In summary, although the breadth-first strategy may have to consider many alternatives for rules of any complexity, the guarantee of finding the shortest possible path seems a decided advantage in its favor.

## 5. Generality of the Path-Finding Process

In this section I consider some domains to which the path-finding mechanism can be applied. The first of these, learning integration from examples, is quite similar to the algebra example. The second domain, learning simple mappings from meanings to sentences, shows the technique is not restricted to finding numerical relations. Next, I show that the method can be used to discover rather complex functions, like that describing Balmer's series. Finally, I consider some difficulties which arise in applying the technique to learning rules for factoring algebraic expressions.

### 5.1. Learning Integration

Consider two specific rules for simplifying integral expressions: $\int 6x^2 \rightarrow 2\int 3x^2$ and $\int 8x^1 \rightarrow 4\int 2x^1$. When ACTG attempts to generalize from this pair of isomorphs, it finds two unbound terms in the action of its new production. These result from the analogous action symbols [2, 4] and [3, 2]. As in the algebra example, ACTG searches through its network of numerical relations, which includes such propositions as $1 + 1 = 2$ and $2 \times 3 = 6$. As before, it ignores connections through such syntactic symbols as +, ×, and =.

The first pair of useful analogous paths are [2 + 1 = 3] and [1 + 1 = 2]; these facts tell ACTG the coefficient it should retain within the scope of the integral. The second pair of relations are [2 + 1 = 3, 2 × 3 = 6] and [1 + 1 = 2, 4 × 2 = 8], which extend the initial paths; these facts tell the system what coefficient goes outside the integral, where the pair [3, 2] corresponds to the already bound internal coefficient. The resulting rule may be stated as:

> **If you have written** $\int v1x^{v2}$,
> **and** v2 + 1 = v3,
> **and** v3 × v4 = v1,
> **then write the expression** $v4\int v3x^{v2}$.

Note that not all of the terms in the new conditions have been replaced with variables. The number 1 was retained because it occupied analogous positions in both paths.

### 5.2. Learning Meaning/Sentence Mappings

The path-finding process can also be employed to let ACTG learn mappings from descriptions of the world to English sentences. For example, suppose the system has a set of facts in its knowledge base about various subset relations, and the words for different concepts. Assume these are represented as separate propositions, say **(Sam is-the-word-for *Sam)** and **(*ball-1 is-a *ball)**, where concepts are preceded with a * and words are not. Now suppose a propositional description of an

event enters active memory, (*throw agent *Sam recipient *dog-1 object *ball-1), accompanied by the sentence **Sam throw s the ball to the dog**. Assume that the relations **is-the-word-for**, **is-a**, **agent**, **recipient**, and **object** are all syntactic elements. If the appropriate designating production exists, this data will lead to the first production in Table 4, which reproduces the sentence whenever the event recurs.

Next, suppose that a similar event occurs, and is represented in active memory by an analogous proposition, (*give agent *Mary recipient *cat-1 object *string-1). In addition, let this proposition be accompanied by a sentence of the same form as the first, **Mary give s the string to the cat**. This leads to the second production in Table 4, and at this point ACTG would attempt to generalize from its two rules. However, the general rule in its initial form has four unbound action terms, so the path-finding process is called. In this case, four independent paths are required, each connecting one action term to a different term in the condition side. These paths are: [(throw is-the-word-for *throw)] and [(give is-the-word-for *give)]; [(John is-the-word-for *John)] and [(Mary is-the-word-for *Mary)]; [(*dog-1 is-a *dog),(dog is-the-word-for *dog)] and [(*cat-1 is-a *cat),(cat is-the-word-for *cat)]; and [(*ball-1 is-a *ball),(ball is-the-word-for *ball)] and [(*string-1 is-a *string),(string is-the-word-for *string)]. The resulting production is presented below its specific precursors in Table 4.

---

**If you see**
(*throw agent *Sam recipient *dog-1 object *ball-1),
**then say Sam throw s the ball to the dog.**

**If you see**
(*give agent *Mary recipient *cat-1 object *string-1),
**then say Mary give s the string to the cat.**

**If you see**
(vaction agent vagent recipient vrec object vobject),
and vword1 is-the-word-for vaction,
and vword2 is-the-word-for vagent,
and vrec is-a vconcept1,
and vword3 is-the-word-for vconcept1,
and vobject is-a vconcept2,
and vword4 is-the-word-for vconcept2,
**then say vword2 vword1 s the vword4 to the vword3.**

---

Table 4. Productions mapping meanings onto sentences.

### 5.3. Discovering Balmer's Law

A third application of the path-finding mechanism is the discovery of complex functions, such as that describing Balmer's series for the hydrogen spectrum. This series[2] may be stated as **9/5, 16/12, 25/21, 36/32, __**. To find one of the fractions, take the square root of the numerator in the preceding fraction, add one to this number and square the result to determine the current numerator. The denominator of a fraction is always four less than its numerator. If **p** is a fraction's position in the sequence, then the fraction may be expressed as $(p + 2)^2/[(p + 2)^2 - 4]$.

Suppose that upon receiving successive entries in the series, a designating production creates specific rules predicting a fraction in terms of its predecessor, such as **9/5 → 16/12** and **16/12 → 25/21**. Since the rules have identical form, first the generalization process, and then the path-finding process are evoked. Searching through the same numerical data base as was used in the algebra and integration examples, ACTG arrives at the general rule presented in Table 5. The last four conditions in this production correspond to the information given verbally above. Using these constraints, the rule will correctly predict the fraction for any position in terms of its predecessor.

---

**If the last fraction was v1 / v2,**
and $v3 \times v3 = v1$, and $v3 + 1 = v4$,
and $v4 \times v4 = v5$, and $v6 + 4 = v5$,
**then the next fraction will be v5 / v6.**

---

Table 5. General rule for Balmer's series.

### 5.4. A Failure of the Method

In factoring algebraic expressions, the goal is to find a set of simplified expressions which, when multiplied together, give the original expression. Suppose ACTG creates two specific factoring rules from examples it has been given, say $6x^2 + 5x - 4 \rightarrow (2x - 1)(3x + 4)$ and $3x^2 + 2x - 8 \rightarrow (3x - 4)(1x + 2)$. Since these are structurally similar, the system would attempt to generalize, but would generate unbound action terms based on the pairs [2, 3], [1, 4], [3, 1], and [4, 2]. The path-finding process would try to connect these to one or more of the analogous condition symbols [6, 3], [5, 2], and [4, 8]. Ideally,

---

[2] In fact, Balmer had only decimal values to examine. According to Banet [2], much of Balmer's insight was the realization that these numbers approximated the ratios of integers given above. ACTG's path-finding process has nothing to say about this aspect of Balmer's discovery.

ACTG would find the following four pairs of analogous paths: [2 × 3 = 6] and [3 × 1 = 3]; [1 × 4 = 4] and [4 × 2 = 8]; [5 + 3 = 8, 2 × 4 = 8] and [2 + 4 = 6, 3 × 2 = 6]; and [5 + 3 = 8, 1 × 3 = 3] and [2 + 4 = 6, 4 × 1 = 4]. Unfortunately, a difficulty arises in reaching this goal.

Taken together, the resulting generalized paths satisfy the uniqueness criterion discussed earlier. However, none of the generalized paths do so individually. Moreover, if this criterion were abandoned, then the first two paths would be sufficient to bind all four action terms. But the resulting rule would be overly general, leading to actions like $6x^2 + 5x \cdot 4 \rightarrow (6x \cdot 2)(1x + 2)$, since there are multiple factorings for 6 and 4. Thus, the uniqueness criterion remains useful, but its implementation in ACTG seems overly restrictive. In the context of breadth-first search, the system could be modified to link paths together for additional constraints. However, the combinatorics would allow this only if a small number of paths were being considered simultaneously. This problem clearly requires more thought, and a search is underway for other learning situations in which this difficulty arises.

## 6. Summary

In this paper I described ACTG, a production system language designed for the study of learning systems. This language incorporates a declarative propositional network, a spreading activation mechanism, and a unique conflict resolution scheme. ACTG also supports a number of automatic learning mechanisms, including the ability to strengthen or weaken existing rules, and the ability to form discriminant and general versions of these rules. The conflict resolution scheme of ACTG should interact with these learning techniques to allow recovery from errors, but this prediction has not been tested.

After summarizing the main characteristics of ACTG, I focussed on one of the learning mechanisms. This was a path-finding process which was called upon when generalization led to unbound variables in the action side of a rule. Once a pair of analogous paths had been found, a generalized version of these paths was incorporated into the condition side of the new rule. In its pure form, this network search would have been computationally intractable, so four techniques were used to direct the search along useful paths. Finally, I considered applications of this method to the learning of mathematical procedures, the induction of meaning to sentence mappings, and to the discovery of complex functions. Future work will concentrate on testing this technique in more complex domains, and on studying its interaction with the other ACTG learning mechanisms.

## References


[1] Anderson, J. R., Kline, P. J., and Beasley, C. M. Complex learning processes. In R. E. Snow, P. A. Federico, and W. E. Montague (eds.), **Aptitude, Learning, and Instruction: Cognitive Process Analyses**, 1979, in press.

[2] Banet, L. Evolution of the Balmer series. **American Journal of Physics**, 1966, 34, 496-503.

[3] Feigenbaum, E.A., Buchanan, B.G., and Lederberg, J. On generality and problem solving: A case study using the DENDRAL program. **Machine Intelligence 6**. Edinburgh University Press, 1971.

[4] Forgy, C. L. On the efficient implementation of production systems. Doctoral dissertation. Computer Science Department, Carnegie-Mellon University, 1979.

[5] Forgy, C. L. OPS4 User's Manual. Computer Science Department, Carnegie-Mellon University, 1979.

[6] McDermott, J. and Forgy, C. L. Production system conflict resolution strategies. In D. A. Waterman and F. Hayes-Roth (eds.), **Pattern-directed inference systems**. New York: Academic Press, 1978, 177-199.

[7] Michalski, R. S. Toward computer-aided induction. Report No. 874, Department of Computer Science, University of Illinois, Urbana, 1977.

[8] Mitchell, T. M. Version spaces: A candidate elimination approach to rule learning. **Proceedings of the Fifth International Joint Conference on Artificial Intelligence**, 1977, 305-310.

[9] Neves, D. M. A computer program that learns algebraic procedures by examining examples and working problems in a textbook. **Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence**, 1978, 191-195.

[10] Pople, H. E. The formation of composite hypotheses in diagnostic problem solving: An exercise in synthetic reasoning. **Proceedings of the Fifth International Joint Conference on Artificial Intelligence**, 1977, 1030-1037.

[11] Vere, S.A. Induction of relational productions in the presence of background information. **Proceedings of the Fifth International Joint Conference on Artificial Intelligence**, 1977, 349-355.

# BACON.4: The Discovery of Intrinsic Properties[1]

Gary L. Bradshaw
Pat Langley
Herbert A. Simon
Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

**Abstract**

BACON.4 is a production system that discovers empirical laws. It is not intended as a detailed model of human discovery processes, but instead attempts to model some of the general processes that can lead to discovery in a number of domains. BACON.4, like its predecessor BACON.3, uses a few simple heuristics to handle a broad range of tasks. These heuristic rules, which are easily implemented in a production system format, detect constancies and trends in data, and lead to the formulation of hypotheses and the definition of theoretical terms. No hard distinctions are made between the data and the hypotheses that explain them, so that laws can be discovered involving an arbitrary number of variables. BACON.4, unlike BACON.3, can discover relevant properties of objects that have been input in nominal form. Numerical values of these properties, consistent with the data, are assigned to each of the nominal values. These quantities can then be used in new hypotheses to discover empirical laws based on the discovered properties. BACON.4 has shown its generality by rediscovering Snell's Law, Black's Law, the law of gravitational attraction, and one form of the law of conservation of momentum.

## 1. Introduction

Centuries ago, physicists like Kepler and Galileo began to discover laws that described the physical world. Such discovery is presently being modeled in Artificial Intelligence with a number of systems. These include DENDRAL [1], meta-DENDRAL [2], and AM [3]. Each of these discovery systems draws on a large amount of knowledge about the domains in which they worked.

Another approach, represented by Langley's BACON.3 program [4], models the discovery process without requiring a large body of domain-specific knowledge. This approach uses a few weak, but general, heuristics that may be applied to discover laws in many domains. BACON.3 showed its generality by rediscovering such laws as the ideal gas law, Kepler's third law, Ohm's law, and Coulomb's law. Although BACON.3 could include variables taking symbolic values into its laws (e.g., the type of metal used in a battery), such variables could only be used in that system as conditions restricting the scope of numerical laws.[2] BACON.4 can move beyond this by hypothesizing a numerical property of a nominal or symbolic variable, associating an appropriate numerical constant with each nominal value, and using these numerical constants to discover a more general law which includes this new property. We will call the numerical variables **intrinsic properties** of the nominal variables.

In this paper, we first discuss how such an intrinsic property is postulated during the discovery of Snell's law. Next, we review the operation of BACON.3, and discuss BACON.4's added heuristics, which allow it to propose intrinsic properties. Next we discuss BACON.4's generality as a discovery system by showing how it finds several other laws. Finally, we show some limitations of the present version of BACON.

## 2. An Example: Discovering Snell's Law

Snell's law relates the angle of incidence, **i**, and the angle of refraction, **r**, of a ray of light as it meets a smooth, transparent interface between two media. The general form of Snell's law is:

$$\text{Sine } i / \text{Sine } r = n_1 / n_2$$

where $n_1$ and $n_2$ are indices of refraction for each medium relative to a common reference medium. **Vacuum** is the usual reference medium, with an index of refraction arbitrarily assigned as 1.0. With such a standard, **water** has an index of refraction of **1.33**, **oil 1.47**, **quartz 1.54**, and ordinary **glass 1.66**.

Suppose one is given experimental control over the angle of incidence (a numerical variable)[3] and the types of the two media (which are nominal variables), and one gathers data by varying first the angle of incidence, followed by the first medium,

[1]This research was supported in part by NIMH Grant MH 07722, in part by NSF Grant SPI 7914852, in part by NSF Grant IST-7918266, and in part by ARPA Grant F44620 73 C0074. We would like to thank Glenn Iba for discussions about ideas presented in this paper.

[2]A nominal or symbolic variable is a variable that takes on names or labels as values. Thus, a nominal variable, "material," may take on values like lead, water, etc.; a nominal variable "object," may take on values like "Object A", "Object B," and so on. Values of numerical properties may sometimes be associated with values of nominal variables, thus: "the density of lead is 13.34 gr/cm$^3$", "the mass of Object A is 3 grams"; etc.

[3]The present version of BACON cannot calculate sine values from angles. Thus sine values are provided to the program. This serves to simplify the problem for BACON.4 by reducing the space through which it must search.

followed by the second medium. Table 1 shows some of the data obtained in this manner when the second medium is **vacuum**. The index of refraction of a medium is an intrinsic property of the medium that is not apparent at the beginning of the experiment. Without being given prior knowledge of the existence of such a property, BACON.4 discovers it as it tries to describe the data.

| SECOND MEDIUM | FIRST MEDIUM | Sine i | Sine r | $\frac{\text{Sine i}}{\text{Sine r}}$ |
|---|---|---|---|---|
| Vacuum | Water | .5000 | .3759 | 1.33 |
| Vacuum | Water | .7071 | .5317 | 1.33 |
| Vacuum | Water | .8660 | .6511 | 1.33 |
| Vacuum | Oil | .5000 | .3401 | 1.47 |
| Vacuum | Oil | .7071 | .4810 | 1.47 |
| Vacuum | Oil | .8660 | .5819 | 1.47 |
| Vacuum | Quartz | .5000 | .3247 | 1.54 |
| Vacuum | Quartz | .7071 | .4592 | 1.54 |
| Vacuum | Quartz | .8660 | .5623 | 1.54 |
| Vacuum | Glass | .5000 | .3012 | 1.66 |
| Vacuum | Glass | .7071 | .4250 | 1.66 |
| Vacuum | Glass | .8660 | .5217 | 1.66 |

TABLE 1. DATA OBEYING SNELL'S LAW

Consider the first three rows of Table 1. As **sine i** increases, **sine r** increases. Since a function of the form **sine i / sine r = k** would lead to such a trend, BACON.4 calculates the values of the ratio **sine i / sine r** (see Section 3 for a review of BACON.4's discovery heuristics). In fact, the values of this term have a constant value 1.33 whenever the first medium is **water** and the second medium is **vacuum**. Constancies with different values occur for **vacuum** paired with **oil**, **vacuum** paired with **glass**, and **vacuum** paired with **quartz**.

At this point the data cannot be summarized further numerically. However, since the nature of the first medium influenced the value of **sine i/sine r**, and because values of the first medium are interchangeable with values of the second medium, BACON.4 assumes that there exists some property of the two media (which we will call the **index of refraction**) that affects the experimental outcome. Because the values of **sine i/sine r** are solely dependent on the values of the first and second medium, BACON.4 assumes that the values of the ratio reflect the different values of the property. We can assign to each of the values of the second medium the value of the ratio **sine i/sine r**. Thus the index of refraction for **water** is **1.33**, **oil** is **1.47**, and so on.

Because the first medium has remained the same, an appropriate value for vacuum cannot be specified. There are several potential ways to find an appropriate index of refraction for vacuum. The first way might be to conduct a second experiment where vacuum appeared as one of the values of the second medium. The relation between vacuum and the other media would allow us to calculate an appropriate index. A simpler approach, which is used by BACON.4, is to exclude vacuum from further experiments, and simply use media for which BACON.4 already knows the appropriate indices of refraction.

| SECOND MEDIUM | FIRST MEDIUM | Sine i | Sine r | $\frac{\text{Sine i}}{\text{Sine r}}$ |
|---|---|---|---|---|
| Water | Oil | .5000 | .4523 | 1.11 |
| Water | Oil | .7071 | .6398 | 1.11 |
| Water | Oil | .8660 | .7835 | 1.11 |
| Water | Quartz | .5000 | .4318 | 1.16 |
| Water | Quartz | .7071 | .6107 | 1.16 |
| Water | Quartz | .8660 | .7479 | 1.16 |
| Water | Glass | .5000 | .4006 | 1.25 |
| Water | Glass | .7071 | .5665 | 1.25 |
| Water | Glass | .8660 | .6938 | 1.25 |

TABLE 2. NEW DATA OBEYING SNELL'S LAW

Table 2 shows data BACON.4 collects after the index-of-refraction property has been postulated. In this table, the second medium is **water**, while the first medium takes on the values **oil**, **glass**, and **quartz**. The program again calculates the values of **sine i / sine r**, which again have constant values for **water** paired with **oil**, **water** paired with **quartz**, and **water** paired with **glass**.

| SECOND MEDIUM | FIRST MEDIUM | Index of Refraction | $\frac{\text{Sine i}}{\text{Sine r}}$ | $\frac{\text{Sine i}}{\text{n1.Sine r}}$ |
|---|---|---|---|---|
| Water | Oil | 1.47 | 1.11 | .752 |
| Water | Quartz | 1.54 | 1.16 | .752 |
| Water | Glass | 1.66 | 1.25 | .752 |
| Oil | Water | 1.33 | 0.90 | .680 |
| Oil | Quartz | 1.54 | 1.05 | .680 |
| Oil | Glass | 1.66 | 1.13 | .680 |
| Quartz | Water | 1.33 | 0.86 | .649 |
| Quartz | Oil | 1.47 | 0.95 | .649 |
| Quartz | Glass | 1.66 | 1.13 | .649 |
| Glass | Water | 1.33 | 0.80 | .602 |
| Glass | Oil | 1.47 | 0.89 | .602 |
| Glass | Quartz | 1.54 | 0.93 | .602 |

TABLE 3. SECOND LEVEL SUMMARY OF SNELL'S LAW

Table 3 shows the second level summaries generated by BACON.4, including the indices of refraction for the first media. The first three rows summarize the data shown in Table 2. At this point, BACON.4 detects a monotonic increasing relationship between the term **sine i / sine r** and the value of $n_1$, the index of refraction for the first medium. It therefore defines a new theoretical term, **sine i**/$n_1$ × **sine r** which has a constant value of 1.11. Other values of this term are calculated as BACON.4 collects further data, and these values are shown in the remaining rows of Table 3. Notice that this term takes on constant values for each value of the second medium. These constancies are shown in Table 4.

| SECOND MEDIUM | Index of Refraction | $\frac{\text{sine i}}{\text{n1 . sine r}}$ | $\frac{\text{n2 . sine i}}{\text{n1 . sine r}}$ |
|---|---|---|---|
| Water | 1.33 | .752 | 1.00 |
| Oil | 1.47 | .680 | 1.00 |
| Quartz | 1.54 | .649 | 1.00 |
| Glass | 1.66 | .602 | 1.00 |

TABLE 4 FINAL SUMMARY OF SNELL'S LAW

In Table 4, only the second medium plays the role of a condition, so BACON.4 introduces the values of $n_2$, the index of refraction for the second medium. Now BACON.4 notes that the value of sine $i/n_1 \times$sine r varies inversely with the index of refraction of the second medium. Based on this, BACON.4 defines a new theoretical term, $n_2 \times$sine $i/n_1 \times$sine r, which has a constant value 1.0. This relationship is equivalent to sine $i/$sine $r = n_1/n_2$, which the reader will recognize as Snell's law.

## 3. BACON.3 and BACON.4

In the last section, we summarized BACON.4's discovery of Snell's law. BACON.4 is an extension of BACON.3, described by Langley [4]. Yet BACON.3 could have discovered Snell's law only if it was explicitly provided with the index of refraction for each medium. In this section, we will review the two programs, and consider the reason for the difference.

### 3.1. A Review of BACON.3

BACON.3 represented its data in terms of conjunctions of attribute-value pairs called **descriptive clusters**; clusters corresponded to rows like those in Tables 1, 2, 3, and 4. The program used a small set of heuristics called **regularity detectors** for finding constancies and trends in the data. in data. The central heuristic could be stated:

**If the dependent variable a has the value v**
**in a number of clusters at level L,**
**then create a cluster at level L + 1 in which a is v,**
**and which has all of the conditions**
**held in common by the lower level clusters.**

Once created, a higher level description could contribute to yet higher level regularities. Much of BACON.3's power derived from this ability to recursively treat hypotheses as new data.

The program used another set of heurstics for noting trends in numerical data. One of these may be summarized as:

**If the values of dependent variable a1 increase**
**as the values of variable $v_2$ decrease**
**in a number of clusters at level L,**
**then note a monotonic decreasing relation**
**between a1 and a2,**
**and calculate the slope of a1 with respect to a2.**

Additional rules further analyzed these data, and defined theoretical terms based on the results. If a constant slope was found, BACON.3 defined new terms for the slope and intercept of the line. Otherwise, the product or the ratio of the variables was considered, depending on the direction of the relation and the signs of the numbers involved. Once defined, a theoretical term could be used in formulating hypotheses or in specifying yet more complex definitions.

BACON.3 was well-suited for discovering purely numerical laws. However, since it could incorporate nominal variables only as conditions on such laws, it would fail to make much progress on Snell's data. Below we describe BACON.4's solution to this dilemma.

### 3.2. BACON.4 and Intrinsic Properties

BACON.4's trend detectors can only operate on numerical variables. At the start of an experiment, the only relevant information known about certain variables might be that these variables take on symbolically different (i.e., nominal) values. BACON.3 could only use such variables to limit the scope of a law.

However, an experimenter has knowledge about the variables being used in an experiment. In the case of Snell's law, the experimenter knows that values of the first and second media are both drawn from a common set: the set of transparent media. In the present terminology, the first and second media are **interchangeable**: one set of values can be substituted for another. It should be obvious that the values of the second medium and the angle of incidence of the first medium are drawn from different sets, and so are non-interchangeable. When BACON.4 is told that two variables can take on interchangeable values, it considers defining a new property associated with these variables.

BACON.4 contains only three more OPS4 productions [5] than the BACON.3 system upon which it was built. The first of these postulates a property associated with a set of variables. It may be paraphrased in English as

**If $v_2$ is an independent nominal variable**
**and if the values of the numerical**
**dependent variable $v_1$ change**
**as the values of variable $v_2$ change,**
**then propose an intrinsic property**
**associated with $v_2$ and**
**with variables whose values**
**can be interchanged with $v_2$.**

The second production is responsible for associating numerical values with nominal ones, after an intrinsic property has been defined. The third production adds the values of the new property to descriptive clusters when they will be useful. Although all of the examples in the present paper deal with

exactly two variables which can take on interchangeable values, BACON.4 is implemented to deal with an arbitrary number of such variables, as well as with multiple sets of them. The program can also deal with the simpler case of associating numerical values with isolated nominal variables.

## 4. The Generality of BACON.4

In an earlier section, we outlined BACON.4's discovery of Snell's law. Below we present evidence for the generality of the program and its heuristic for postulating intrinsic properties. First we summarize the system's discovery of Black's law for temperature mixtures and the concept of **specific heat**. Next we discuss Cavendish's experiment on gravitational attraction, followed by an experiment exploring conservation of momentum, in which BACON.4 postulates two notions of **mass**.

### 4.1. Postulating Specific Heat

In the 1860's, Joseph Black began systematically to mix liquids of different temperatures together, and to observe the final temperatures of these mixtures. If we let $t_1$ and $t_2$ represent the initial temperatures of the two liquids, $m_1$ and $m_2$ stand for their masses, and $t_f$ be the final temperature, then Black's law may be stated as:

$$t_f = (c_1m_1t_1 + c_2m_2t_2)/(c_1m_1 + c_2m_2).$$

The symbols $c_1$ and $c_2$ represent the **specific heats** of the liquids being mixed. The specific heat is a numerical value associated with a particular liquid that summarizes the role the liquid plays in Black's equation. For example, if we let the specific heat of **water** be 1.0, then the value for **mercury** is 0.0332, and the value for **ethyl alcohol** is 0.456.

BACON.4's discovery of Black's law results from a straightforward application of the techniques discussed earlier. The system starts with experimental control of $t_1$, $t_2$, $m_1$, $m_2$, and the two liquids being used. The first four of these variables take on numerical values, while the last two take nominal ones. The single dependent variable is $t_f$, the final temperature. Upon varying the values of $t_2$, the program notes a linear relationship between this term and $t_f$. Accordingly, BACON.4 defines theoretical terms for the slope and intercept of this line, $s_{t_f,t_2}$ and $i_{t_f,t_2}$. Later, BACON.4 will discover a relation between these terms and the other variables. Although the program does not express things in this manner, the values of the intercept may be calculated as $c_1m_1t_1t_f/(c_1m_1t_1 + c_2m_2t_2)$, and the slope may be found from $(t_f - t)/t_2$.

When the values of $t_1$ are varied as well, the program finds that the values of $i_{t_f,t_2}$ change, but that the values of $s_{t_f,t_2}$ are unaffected. A ratio term, $i_{t_f,t_2}/t_1$, is defined and found to have a constant value for the current values of $m_1$, $m_2$, and the two liquids. Upon altering the values of $m_2$, the values of this new term are affected, but so are those of the original slope term, $s_{t_f,t_2}$. Since the values of $s_{t_f,t_2}$ increase as those of $i_{t_f,t_2}/t_1$ decrease, the product of these terms is considered. This theoretical term is not constant, but varies directly with the mass, so that the term $t_1s_{t_f,t_2}/m_2i_{t_f,t_2}$ is defined. This term has a constant value dependent on the values of $m_1$ and the two liquids. When $m_1$ is varied, BACON.4 finds a monotonic decreasing relationship between $m_1$ and the new term, leading to the product $m_1t_1s_{t_f,t_2}/m_2i_{t_f,t_2}$, which has a constant value dependent only on what two liquids are used.

At this point, BACON.4 can do nothing except associate the values of this constant product with the liquids with which they are associated. The program defines a new intrinsic property, which we may call **specific heat**. It designates a specialized version of this concept for each of the two original nominal terms, which we may call $c_1$ and $c_2$. Upon incorporating the values of $c_2$, BACON.4 finds a monotonic increasing relationship and defines the term $m_1t_1s_{t_f,t_2}/c_2m_2i_{t_f,t_2}$. This has a constant value dependent only on the first liquid used. When the values of $c_1$ are related to these summary data, a decreasing relation is found. The resulting term, $c_1m_1t_1s_{t_f,t_2}/c_2m_2i_{t_f,t_2}$, has the constant value 1.0 under all circumstances. Substituting in the definitions of the slope and intercept terms discussed earlier, one finds that BACON.4's summary of this fact is equivalent to Black's law.

### 4.2. BACON.4 and Gravitational Mass

In the late 1790's, Henry Cavendish designed an apparatus to measure the value of G, the universal constant of gravitation. This apparatus consisted of 1) an object attached to an arm which is suspended from a quartz fiber, 2) a second object which moves toward the suspended object, and 3) a mirror and light source which measures the resultant torque produced by attraction between the two objects. BACON.4 can use data collected in this experiment to find the law of gravitational attraction. If we let $m_1$ be the mass of the suspended object, $m_2$ be the mass of the movable object, D be the distance between their centers of mass, and F be the observed force between the two objects, the law of gravitational attraction may be stated as:

$$F = G \times m_1 \times m_2 / D^2$$

If BACON.4 is given only nominal values for a number of pairs of objects $m_1$ and $m_2$, it can discover mass as a property during the experiment as well. Table 5 shows the values of data collected in this experiment when the suspended object is object A.

| SUSPENDED OBJECT | MOVEABLE OBJECT | DISTANCE (meters) | FORCE ($nt \cdot 100$) | F·D ($nt \cdot m \cdot 100$) | $F \cdot D^2$ ($nt \cdot m^2 \cdot 100$) |
|---|---|---|---|---|---|
| A | B | 0.01 | 2.001 | .02001 | .000200 |
| A | B | 0.02 | 0.500 | .01000 | .000200 |
| A | B | 0.03 | 0.222 | .00666 | .000200 |
| A | C | 0.01 | 2.501 | .02501 | .000250 |
| A | C | 0.02 | 0.625 | .01250 | .000250 |
| A | C | 0.03 | 0.278 | .00834 | .000250 |
| A | D | 0.01 | 3.001 | .03501 | .000300 |
| A | D | 0.02 | 0.750 | .01500 | .000300 |
| A | D | 0.03 | 0.334 | .01002 | .000300 |
| A | E | 0.01 | 3.501 | .03501 | .000350 |
| A | E | 0.02 | 0.875 | .01750 | .000350 |
| A | E | 0.03 | 0.389 | .01167 | .000350 |

TABLE 5. DATA OBEYING GRAVITATIONAL LAW

In the first three rows of Table 5, the gravitational force decreases as the distance increases. This leads BACON.4 to calculate the product $F \times D$. Because the new term does not have a constant value, and because $F \times D$ decreases when D increases, BACON.4 calculates a second product, $F \times D^2$. This has a constant value of 0.000200 when the suspended object is object A and the movable object is object B. BACON.4 finds similar constants for the pairs of objects A and C, A and D, and A and E. The values of these constants are dependent upon values that BACON.4 knows are interchangeable. Thus BACON.4 postulates a property, **gravitational mass**, of the two objects, and associates the values of the $F \times D^2$ column with values of the movable object. These values, associated with objects B, C, D, and E, are the true masses of each object multiplied by a constant, $G \times M_a$. Thus all mass values are values relative to the mass of the suspended object used in the initial experiment.

| SUSPENDED OBJECT | MOVEABLE OBJECT | MASS OF OBJECT $M_2$ ($10^{-3}$kg) | $F \cdot D^2$ | $\frac{F \cdot D2}{M_2}$ |
|---|---|---|---|---|
| B | C | .0025 | .00333 | 1.333 |
| B | D | .0030 | .00400 | 1.333 |
| B | E | .0035 | .00467 | 1.333 |
| C | B | .0020 | .00333 | 1.666 |
| C | D | .0030 | .00500 | 1.666 |
| C | E | .0035 | .00583 | 1.666 |
| D | B | .0020 | .00400 | 2.000 |
| D | C | .0025 | .00500 | 2.000 |
| D | E | .0035 | .00700 | 2.000 |
| E | B | .0020 | .00467 | 2.335 |
| E | C | .0025 | .00583 | 2.335 |
| E | D | .0030 | .00700 | 2.335 |

TABLE 6. SECOND LEVEL SUMMARY OF GRAVITATIONAL LAW

Next, BACON.4 collects more data where the suspended object is object B, C, D, or E. Again it finds values of $F \times D^2$ to be constant for each pair of objects. These values are summarized at a higher level of description, shown in Table 6. At this level, BACON.4 can use the values of the mass of the movable object to define a new term, $F \times D^2 / m_2$. These values are constant for each value of the suspended object. This will cause BACON.4 to generate a third level of description, shown in Table 7.

| SUSPENDED OBJECT | MASS OF OBJECT $M_2$ | $\frac{F \cdot D^2}{M_2}$ | $\frac{F \cdot D^2}{M_1 \cdot M_2}$ |
|---|---|---|---|
| B | .00020 | 1.333 | $6.667 \times 10^5$ |
| C | .00025 | 1.666 | $6.667 \times 10^5$ |
| D | .00030 | 2.000 | $6.667 \times 10^5$ |
| E | .00035 | 2.333 | $6.667 \times 10^5$ |

TABLE 7. FINAL SUMMARY OF GRAVITATIONAL LAW

This level of description can be completely summarized by utilizing the mass of the suspended object to define a final term $F \times D^2 / m_1 \times m_2$. This term has a constant value $6.663 \times 10^5$, which is equivalent to the $1 / G \times M_a^2$. The final form of the law could be expressed in terms of usual mass values as: $F \times D^2 / G^2 M_A^2 M_1 M_2 = 1 / G M_A^2$. This is equivalent to the normal form of the law. The peculiar values of the masses assigned to each object derive from the heuristic BACON.4 uses to assign numerical values to objects. BACON.4 does not know the actual masses of each object. If it did, BACON.4 could discover the usual form of the law. Instead it associates values of the masses B, C, D, and E which are relative for a common reference mass, mass A. If relative values were inappropriate, the method would fail (see Section 5 for an example and further discussion on this issue).

### 4.3. BACON.4 and Inertial Mass

BACON.4 can discover mass in a second kind of experiment. Consider the case where two masses, $m_1$ and $m_2$, are connected by a perfectly elastic spring which has no inertia. If the objects are pulled apart and then released, the two masses will enter into harmonic oscillation. One can measure the velocity of the two objects at several points in their period. This situation is a special case of conservation of momentum, where $m_1 \times v_1 / m_2 \times v_2 = 1.0$. For this experiment, BACON.4 has experimental control over nominal values for the first and second masses, as well as over the times at which the observations are made; it observes the dependent values of $v_1$ and $v_2$ under various combinations of the independent values.

In discovering this law, BACON.4 varies the times at which the velocities are measured, and notes that the velocities $v_1$ and $v_2$ increase together. Accordingly, it defines the term $v_1 / v_2$, which does not vary with the time of the observation. Next the program defines a property of the objects, **inertial mass**. The

values of this property for $m_2$ are discovered to vary directly with $v_1/v_2$, so BACON.4 defines the term $v_1/m_2 \times v_2$. This has a constant value dependent upon the first object. BACON.4 finally incorporates mass values of $m_1$, finds these are directly related to $v_1/m_2 \times v_2$, and summarizes the data as $m_1 \times v_1/m_2 \times v_2$, which has a constant value 1.0.

We have shown how BACON.4 can discover mass in two separate experiments. The discovery of mass has been shown to be a necessary step in the discovery of Newtonian mechanics [6]. Mass is a general property that is used in many laws. However, instead of rediscovering mass in several different experiments, BACON.4 can use a property discovered in one experiment to facilitate discovery of a different law.

In the two previous sections, we have shown how two different forms of mass were discovered: inertial mass and gravitational mass. However, if the program that discovered mass in solving the gravitational attraction experiment was told that objects in the conservation experiment were interchangeable with gravitational objects, and if the same objects were used, then the program would already have available mass values for the objects. It would discover that gravitational mass could be used to predict the momentum of the system. In this case, two separate notions about mass would never have been postulated. Similarly, if the order of experiments were reversed, BACON.4 would incorporate inertial mass values into the gravitational experiment, without postulating gravitational mass. This again reflects the data driven nature of BACON.4 The program will only begin to infer properties of objects when no further summarization of the existing data is possible.

## 5. Limitations of BACON.4

In this section, we discuss a case where the present technique of postulating properties is not applicable. After this, we mention some more general limitations of BACON.4 as a discovery system.

### 5.1. BACON.4 and Friction

In an experiment to determine the force required to cause one object to slide on the surface of another, one might vary the composition of the sliding object, the composition of the surface, and the weight of the sliding object. Since the sliding object and surface material are interchangeable, BACON.4 would postulate a property of such materials, which might be thought of as the roughness of a surface.

However, it is well known that the coefficient of friction, $\mu$, is associated not with a single type of material, but with *pairs* of materials. In this case, the values of surface roughness would not lead to finding constancies in the data, and BACON.4 would fail to summarize the data. In the present system, no provision is made to deal with such a circumstance. There are undoubtedly other cases in the literature of chemistry and physics which are similar to this example. In all such cases, BACON.4 will fail to find the generally accepted laws.

The friction example illustrates an important assumption in the present property discovery process. BACON.4 assigns numerical values to one of the interchangeable variables when it does not know the effect of the other interchangeable variable. In the case of gravitational mass discussed earlier, for example, the mass value assigned to object B was equivalent to mass B × mass A × G, the value assigned to object C was equivalent to mass C × mass A × G, and so on. Since mass is a transitive property, the relationship between the values assigned to objects B and C is the correct one. If, as in the friction example, the property is not transitive, the values assigned to object B and C will be inappropriate ones. Of course, there is no way of knowing in advance whether or not the assumption will hold.

### 5.2. Guiding BACON.4's discovery

In the present version of BACON.4, the program is supplied with a certain amount of information about the experiment at hand. The information includes: 1) the variables that the program has to manipulate; 2) appropriate levels of these variables to use; and 3) dependent variables the program can measure. Also, by excluding irrelevant variables, we are reducing the space of potential laws the program must search. Undoubtedly we are guiding the discoveries of BACON.4 by these means. Let us look at each of these problems in turn. We will start by looking at the problem of irrelevant variables.

BACON.3 employed a simple heuristic for dealing with irrelevant variables: if an independent variable is manipulated, but has no effect on the dependent variable(s), it is classified as an irrelevant variable, and its level is held constant. BACON.4 incorporates the same heuristic, so that including irrelevant variables would not prevent it from discovering any of the laws presented here. However, the program would systematically vary the irrelevant variable(s), and so would require more time to discover the laws. Thus the program can be slowed by including extraneous variables, but it cannot be disrupted by them. Our primary interests have not been in this area, so we have no statistics on the extent to which discovery is slowed by introducing one or more extraneous variables.

The question of how to decide which variables to include in an experiment, both independent and dependent, is undoubtedly a serious one. However, this question is outside the realm of the present project, and we have little to say about the matter. The only provision in the present version of BACON which relates to this issue is BACON.4's ability to keep track of properties of variables it has discovered, and to utilize such properties when they are useful. Perhaps some other approach will be more fruitful in addressing the issue of selecting potential variables.

The final concern, that of choosing appropriate levels for independent variables, might perhaps be addressed in a future version of BACON. In order to choose such levels, additional of knowledge must be included in the program. Such knowledge would include: 1) costs associated with achieving different levels of a variable; 2) the potential range a variable might assume; and 3) extraneous effects of a variable in the experiment (i.e., one cannot study water at -20°c). Again this has not been a major thrust of our project.

### 5.3. BACON.4 and Noise

BACON.4 has only primitive facilties for dealing with noise in its data. Suppose the program is comparing two numbers, $n_1$ and $n_2$, to see if they are approximately equal. The system takes the larger of these values and multiplies its absolute value by a **noise factor** f (0.001 in all of the reported runs). The resulting product is both added to and subtracted from $n_1$ to create an interval. If the smaller number $n_2$ falls within this interval (that is, if $n_1 - f \cdot |n_1| < n_2 < n_1 + f \cdot |n_1|$), the two numbers are treated as identical.

This technique was adequate for dealing with the roundoff errors BACON.4 produced while calculating the values of theoretical terms. However, the introduction of realistic noise into the data may require major modifications in the system's control structure. In particular, the regularity detectors may not be powerful enough to propose unique paths through the space of theoretical terms and hypotheses. One alternative is to enable BACON to entertain many hypotheses at once, rejecting some as disconfirming evidence becomes available, but generating more as variants of those that are retained. Such a **beam search** through the space of hypotheses should be sufficiently robust to deal with substantial noise in the data.

The limitations shown in this section are serious ones, and their solution will require considerable time and effort. However, our present experience suggests that most of these problems can be solved by direct extensions of the present program.

## 6. Summary

In summary, BACON.4 is a production system that can rediscover a number of laws from the history of physics. In the process, the program notices regularities in data, defines theoretical terms, postulates properties of symbolic variables, and summarizes its data at various levels of description. We showed how the ability to postulate properties could be used to discover properties such as mass, index of refraction, and specific heat. Thus we have evidence that the technique is a general one.

Further evidence of the generality of the BACON system of programs is evident in that only three productions were added to to create BACON.4. This is due, in part, to the great flexibility afforded by the production system format. It is also due to the generality of the heuristics included in the program. This suggests that the production system format is a suitable one for formalizing discovery programs.

Limitations of the present system include the fact that potentially relevant independent variables must be given to the program, values for these variables must be supplied, the dependent variables must be specified, and the program has only a limited ability to compensate for noise in the data. We hope to be able to overcome some of these problems in future versions of BACON.

## REFERENCES


[1] Buchanan, B.G., Sutherland, G., and Feigenbaum, E.B. Heuristic DENDRAL: A program for generating exploratory hypotheses in organic chemistry. In B. Meltzer and D. Michie (eds.), *Machine Intelligence 4*. New York: American Elsevier Publishing Co., 1969.

[2] Buchanan, B.G., Feigenbaum, E.B., and Sridharan, N.S. Heuristic theory formation. In D. Michie (ed.), *Machine Intelligence 7*. New York: American Elsevier Publishing Co., 1972, 267-290.

[3] Lenat, D.B. Automated theory formation in mathematics. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*. 1977, 833-842.

[4] Langley, P. Rediscovering physics with BACON.3. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, 1979, 505-507.

[5] Forgy, C. L. The OPS4 reference manual. Technical report, Department of Computer Science, Carnegie-Mellon University, 1979.

[6] Simon, H.A. The axioms of Newtonian mechanics. *Philosophical Magazine, Series 7*, Vol. 38, 1947.

# INCREMENTAL DEDUCTION IN A REAL-TIME ENVIRONMENT*

Robert Bechtel
Paul Morris

System Development Corporation

Dennis Kibler

University of California, Irvine

ABSTRACT

STAMMER is a system for identifying objects detected by sensors on board naval vessels and for interactively explaining the identification process. The system operates in a continuing environment where later information may supercede earlier data. New techniques are described for facilitating and revising deductions in this kind of developing situation. Other interesting features of the confidence and explanation mechanisms are discussed.

## INTRODUCTION

A ship at sea has a broad range of data collection devices which can be utilized to aid assessment of the current situation in its immediate vicinity. These devices include on-board sensors like radar and sonar, and communication links with other ships and land bases which can serve as information relays from off-ship sensors. Even with these resources at their disposal, naval personnel can find it hard to maintain an accurate picture of the existing tactical situation. At the lowest level the amount of data available can be staggering, while converting the low level data to useful higher level concepts is an art at best.

A rule-based inference system called STAMMER[1,2] has been devised to help fill the gap between the available information and higher level concepts which are more useful in tactical situation assessment. STAMMER applies its rules to a data base of assertions about ships and aircraft. Relatively static information about facts like ship names and characteristics is available to the system at initialization. From time to time, messages containing information from sensors and communication links are received, and assertions representing the information are added to the data base. Typical messages indicate the detection of an object at some location at some time (e.g. blip spotted at 57.4N, 13.23W at time 0715), but may also report on weather conditions or other factors of interest. When rules fire, they add their conclusions to the data base, and STAMMER reports to the user both in text form and with a graphical situation plot.

An abbreviated transcript containing only the messages (indented) and conclusions follows:

```
      RADAR contact at (63.75 -23.95) Time: 0

REPORT: CONTACT1 was sighted in
          merchant lane LANE2

      SONAR contact at (63.75 -24.09) Time: 15

A0300:  CONTACT1 is somewhat likely  (.15)
          to be a PATROL

      SONAR contact at (63.75 -24.14) Time: 20

A0300: CONTACT1 is somewhat likely (.28)
         to be a PATROL

      SONAR contact at (63.75  -24.19) Time:25
      SONAR contact at (63.75  -24.24) Time:30
```

* This paper reports on work done under contract N00123-76-C-0172 for the Naval Ocean Systems Center, San Diego.

SONAR contact at (63.75 -24.33) Time:40

AO570: CONTACT1 is probably not (-.54)
a MERCHANT

SONAR contact at (63.75 -24.42) Time:50

AO570: CONTACT1 is very probably not (-.8)
a MERCHANT

One of the major conclusions that STAMMER reaches is the identification of a vessel's type from its actions.

Upon receipt of a report from the system, the user may query the data base, trace paths of reasoning, or manipulate the graphic display to help satisfy himself as to the nature and scope of the situation and the appropriateness of the system's reasoning.

The unsolicited nature of data entry suggests a data-driven approach to rule evaluation. For this reason, the rule interpreter operates primarily in a bottom-up, forward-chaining mode. When combined with a facility for user definition of rules, this approach has the additional advantage of permitting the creation and activation of demons to monitor messages for particular situations of interest.

INTERPRETER

The design of the rule interpreter has been strongly influenced by two considerations. First, the form of the rules should be kept simple. To meet this objective, it was decided that conditions of rules should refer as much as possible to states of the external world, rather than internal states of the system. In particular, details such as confidence management and control of repetitious firings are the responsibility of the interpreter, not of the rules themselves. Second, the interpreter should be highly efficient. As the data base becomes large, it is essential to avoid redoing work that has already been done. The potential for duplication of effort is especially great in this environment, because rule-firing attempts recur after every message. It is necessary to suppress attempts in circumstances that have occurred before.

Rules

The data base in STAMMER may be viewed as a set of assertions. An assertion is essentially an N-tuple, where the first element is a relation name. EXAMPLE: (SIGHTING S1 CONNOLE) asserts that S1 is a sighting of the CONNOLE. The rules in STAMMER are primarily inference rules, which are executed by the interpreter to modify the data base. Besides being executed, the rules may be used by other components of STAMMER, such as the explanation system. The internal form of the rules is neutral toward the various uses. Each rule has a condition part and an action part. The condition part is a conjunction of conditions. For inference rules, the action part is a conjunction of conclusions. The conditions and conclusions of the inference rules have the form of assertions in which some of the arguments are replaced by variables. This allows them to function as assertion patterns or templates, with the variables serving as "wildcard" entries. An example of a rule is:

```
(SIGHTING *X *Y)(STORM *Z)(INSIDE *X *Z) ->
   (NOTMERCHANT *Y)
```

The variables are distinguished by an initial asterisk. The rule may be paraphrased as saying that, if *X is a sighting of *Y, and *Z is a storm, and *X is inside *Z, then this is evidence that *Y is not a merchant (ship). Such a rule is founded on the presumption that merchants tend to avoid storms more than other ships.

Conditions of rules may also be Boolean combinations of the elementary forms. In addition, an UNLESS operator is provided.

The task of the rule interpreter is to maintain the following situation: during quiescent periods (i.e. after the system has finished responding to new information), for every combination of assertions in the data base matching the conditions of a rule, the corresponding conclusions must also be present in the data base. Moreover, these must have appropriate confidences, based on the current confidences in the conditions.

The notation for rules resembles that of PROLOG [10], and has a similar natural declarative sense. However, PROLOG executes rules top-down, whereas the interpreter here is bottom-up, and includes a confidence mechanism.

Incremental Deduction

In a bottom-up (forward-chaining) system, an obvious problem is to prevent rules from firing repeatedly on the same data. One solution is to have the rules alter the data base in such a way as to invalidate one or more of their conditions. The rules in such a system might appropriately be referred to as "while-do" rules, since their

behavior resembles that of a WHILE statement in a conventional programming language. A disadvantage of this approach is that the rules are cluttered with additional conditions and conclusions that play essentially a book-keeping role. In addition, the rules no longer have a natural declarative meaning. A second solution is to incorporate an ad hoc mechanism that retains a record of previous firings and intervenes to prevent duplications. Unfortunately, this does nothing to eliminate the wasted effort involved in repeatedly matching initial segments of rules. An example will make this clearer. Consider once again the rule:

```
(SIGHTING *X *Y)(STORM *Z)(INSIDE *X *Z) ->
   (NOTMERCHANT *Y)
```

If the system currently knows about m sightings and n storms, then it had to do (at least) m*n retrievals of the INSIDE relation. Now suppose a new sighting is received. The rule is again potentially applicable. A naive system might now do (m+1)*n retrievals of INSIDE, representing all of the possible combinations. However, only n of these are new; the rest are redone in unchanged circumstances. This is wasted effort. If m is large, the cost may be substantial. Clearly, we would wish that only the new sighting should be used in conjunction with each of the known storms. Similarly, if a report of a new storm is received, then only that storm should be considered in relation to each of the known sightings. The problem is to find a mechanism to achieve this.

Suppose, for the moment, that we could delay a rule from being considered until all the information that might match its conditions is present in the data base. For the example above, this would be at a point when we could be sure there would be no more late reports for the time-span of interest. Assume "executing" a rule means systematically trying out, without repetition, all the possible matches for conditions by items presently in the data base. Then it is only necessary to execute the delayed rule exactly once, and the redundancy feared does not occur. It is not feasible, in general, to delay execution in this manner. A viable alternative, however, is to distribute a single execution over time, interleaved with the other business of the system. Portions of the execution are suspended until needed information arrives. The effect, in terms of CPU time (ignoring overhead), is the same as if the information had been present from the beginning. We call this technique "incremental deduction."

A somewhat related procedure, called "splitting," is outlined in Rieger [14].

Streams

To explain the functioning of the system more fully, it is beneficial to introduce a construct that has been used in experimental programming languages [11,12] (Landin[13] first described the idea.). A stream may be regarded as a distinguished sequence of values, produced in the course of a computation. For example, the history of successive values of a variable constitutes a stream. By providing certain operators, we may manipulate streams to our advantage. In STAMMER we have implemented a mechanism which allows the creation of explicit streams. These may be manipulated by a MAPSTREAM operator analogous to the normal MAPC of LISP. The following imaginary LISP session illustrates how these work:

```
_(SETQ S (NEWSTREAM))          ...create an empty
((NIL) NIL)                       stream S
_(PUTSTREAM S 1)               ...put 1 into stream S
NIL
_(MAPSTREAM S 'PRINT)          ...attach the function
1                                 "print" to stream S
NIL
_(PUTSTREAM S 3)               ..."print" acts on new
3                                 element
NIL
_(MAPSTREAM S '(LAMBDA(X)(PRINT 2*X]
2                              ...new function applies
6                        to all previous stream entries
NIL
_(PUTSTREAM S 5)               ...both functions act
5                                 on new entry.
10
NIL
```

The function NEWSTREAM returns a structure that serves as the means of addressing the stream. A MAPSTREAMed function will immediately be called on the elements already in the stream. In addition, a demon will be attached to the stream structure, so that any subsequent elements will also get acted upon by the function. This provides a natural form of parallellism in an otherwise sequential language. In fact, MAPSTREAM may be regarded as initiating a parallel process that hangs while waiting for input from the stream.

Streams may be viewed as an example of embedding control information in data structures. Kowalski [9] discusses the role of control in algorithms, contrasting it with that of logic.

Assume now that the assertions in the data base are organized into streams, with new information added to the appropriate streams. A simplified definition of the interpreter may be given as follows (in pseudo-ALGOL - the "for each .. in stream .." represents a MAPSTREAM):

```
PROCEDURE interp(conds,actions,bindings)
if conds is NIL then execute(actions,bindings)
 else for each x in stream matching first cond do
  interp(other conds,actions,new bindings);
```

The new bindings consist of the old ones plus those determined by the match on x.

An obvious way to implement streams is by means of coroutines. The demands on space, however, are so onerous that it is preferable to build special purpose mechanisms for particular applications. A stream addressing structure may be considered to have two parts: a history list, containing elements previously put in the stream, and a list of "suspensions," representing the demons waiting for future items to be placed in the stream. In our case, a suspension can be represented economically by a pair consisting of a context (set of bindings) and the "tail" of the rule. Structure sharing may be used heavily to reduce the space occupied by contexts. Apart from the context, the overhead required by a suspension is a single word.

The suspensions saved are associated with conditions of rules. A new item of information may "arouse" a suspension and cause the interpreter to move through a few rule conditions before suspending again. This has done part of the work toward a future firing of the rule. This amortization of the cost of rule firing over several messages is appropriate in the present task, where there may be considerable idle time between significant occurrences.

An implication of this approach is that the problem of rule selection disappears, since each rule is "executed" exactly once. The task of determining which suspensions are relevant to a new item of information is handled automatically by the stream mechanism.

Pulsars

So far, we have ignored the role that confidences play in the interpreter. A potential deduction is suspended at a condition even though a matching item is in the data base, if the confidence in that item is below a threshold value. If the confidence subsequently changes, it may be necessary to revive the deduction at that point. Since the item is already in the relevant assertion stream, it cannot be placed there again to trigger the suspension. We avoid this difficulty by associating a stream of pulses, called a pulsar, with each assertion already in the data base (actually, the pulsar mechanism differs somewhat from that of a normal stream). When the confidence changes, a pulse is added, reviving the suspended deductions.

Truth Maintenance

An assertion which is not believed (i.e. with low confidence) may later be believed due to fresh information. In addition, the UNLESS operator, which permits inferences based on absence of information, gives the logic a non-monotonic character [4]. Thus, assertions which are currently believed may later cease to be believed. Messages may be overturned. The effect of all these is two-fold: rules which failed to fire at one point should later fire, and rules which fired earlier should in some sense be "withdrawn." This is the issue of truth maintenance [3,5,6,8]. The first problem is dealt with in STAMMER by the pulsar mechanism discussed earlier. The second requirement is handled by delayed computation of confidence. When a rule fires, a derivation record is constructed, but no confidence is computed for the conclusion. When a confidence is needed, it is computed using the existing derivation records. Then, when computing the contribution of confidence due to a previous rule application, the current degree of belief in the conditions is taken into account. Thus, "dead derivations" are in effect withdrawn from the system, although their structure remains.

One type of truth maintenance that is described in Doyle [8] is not handled by STAMMER. This is the "conflict of evidence" situation where Doyle's system will attempt to find a culprit and revise the beliefs accordingly. At present STAMMER merely sums evidence algebraically, following the approach of MYCIN [7].

DATA BASE

STAMMER's data base is a collection of assertions, each of which is a statement that some relation holds among some objects. For example, the assertion

(LATITUDE SIGHTING27 54.52)

states that the latitude of sighting27 is 54.52 (north). LATITUDE, SIGHTING27, and 54.52 are all assertion elements. Assertions may involve any number of objects, though each relation has a

fixed number of object slots.

While the data base is a collection of assertions, the assertions are organized into streams. Retrievals produce a stream structure that references assertions, rather than a simple list of assertions.

Retrievals are performed by specifying some or all of the assertion elements which form the desired assertions, with the remaining places "wildcarded." Using "*" for a wildcard, the following retrieval specifications would identify streams containing the sample assertion given earlier:

1. (LATITUDE SIGHTING27 54.52)
2. (LATITUDE SIGHTING27 *)
3. (LATITUDE * 54.52)
4. (LATITUDE * *)
5. (* SIGHTING27 54.52)
6. (* SIGHTING27 *)
7. (* * 54.52)
8. (* * *)

The first retrieval specification totally specifies an assertion, while the rest are partial specifications. Analysis of the rules presently used has shown that the specifications 5-8 are never needed in this application, so they are not stored.

The sample assertion would thus be stored in four distinct streams. We accept this redundant storage in return for quick access via hashing based on the retrieval specification. This method permits retrieval in constant time.

CONFIDENCES

All assertions in the system have a confidence associated with them. The basic confidence calculation is that of MYCIN, where the confidence in an assertion is taken to be the difference between the measure of belief (MB) and the measure of disbelief (MD) in the assertion. Rules have a similar singular measure, called the strength of a rule, which indicates what the confidence in the conclusion would be if the confidence in all conditions was 1.0. More precisely, the contribution of confidence to the conclusion of a rule is given by:

$$MB = MB(\text{conditions}) \times \text{STRENGTH}(\text{rule})$$
$$MD = MD(\text{conditions}) \times \text{STRENGTH}(\text{rule})$$

Note, however, that if MD > MB then the rule does not apply. Ground assertions (those that are not derived from rule applications) have measures of belief and disbelief associated with them directly by whatever creates them. Derived assertions get their measures of belief and disbelief calculated dynamically upon access by tracing through the derivation to ground assertions and combining the measures of belief and disbelief as in MYCIN.

Extensions to the general MYCIN scheme fall into three areas. First, a new logical operator (UNLESS) is introduced which provides non-monotonic capabilities [4] by allowing rules to be based on the absence of information. We interpret a non-existent assertion as equivalent to an identical ground assertion wth a confidence of 0.0. Informally, the semantics of UNLESS permit it to be satisfied by assertions with negative or zero confidence. The formal semantics of the UNLESS operator is given by:

$$MB = \begin{cases} 0 & \text{if } MB\text{-}MD > 0.0 \\ 1 & \text{otherwise} \end{cases}$$
$$MD = \begin{cases} 1 & \text{if } MB\text{-}MD > 0.0 \\ 0 & \text{otherwise} \end{cases}$$

The second and third extensions grow out of the retained derivation record and the dynamic demand-driven calculation of confidence. When the derivation record is retained, it may contain cycles. For example, a rule like

(FRIEND *X *Y) -> (FRIEND *Y *X)

could be used to infer (FRIEND MARY JOHN) if given (FRIEND JOHN MARY). However, once (FRIEND MARY JOHN) is in the data base, the rule may fire again, providing (seemingly) new evidence for (FRIEND JOHN MARY), in an instance of cyclic reasoning. Our solution to this problem is not to restrict the rules but rather to cope with cycles in calculating confidences.

The confidence computation algorithm handles loops in the derivation structure as follows. As it descends recursively, it marks nodes it has seen. This allows it to detect loops. Assume the node A is discovered to begin a cycle. Then A appears twice in the descent path. Rules may supply positive or negative evidence for their conclusions. If the cycle represents positive evidence for A, then the computation proceeds as though the lower appearing A had a confidence of -1. If the cycle indicates negative evidence for A, then the lower A is regarded as having a confidence of +1. We are essentially using the well known mathematician's trick of assuming not X when trying to prove X.

More explicitly, the following six types of cycles may occur:

```
        A --> A          A --> not A
    not A --> A      not A --> not A
unless A --> A   unless A --> not A
```

Note that no rule concludes "unless A." In the lefthand column each lower or antecedent A is given a confidence of -1. In the righthand column each lower A is given a confidence of +1. These assignments yield the appropriate confidence in the consequent, as can be easily checked.

The following symbolic example illustrates the confidence calculation. Assume that we have the rules:

```
R1: A-->B
R2: B-->not C
R3: C-->B
R4: D-->C
```

with strengths s1, s2, s3, and s4, respectively. Graphically the situation is:

```
            s2
           --->not
    s1                  s4
A--->  B          C <--- D
           s3
          <---
```

Assume also that the measure of belief in A and D are a and d respectively (with zero measures of disbelief). Moreover let x#y denote 1-(1-x)(1-y). Then the confidences are given by:

```
conf(B)= (s1*a)#(s3*s4*d)
```

i.e. as though the situation were

```
    s1     s3     s4
A ---> B <--- C <--- D
```

and conf(C)= s4*d - s2*[(s1*a)#s3]

i.e. as though we had

```
    s1     s2
A ---> B --->not     s4
       ^           C <--- D
       |s3
       T
```

where T has confidence 1.

Finally, recalculating confidence in an assertion whenever the confidence is requested provides an automatic update facility for those assertions derived from others whose confidences have changed. This updating of confidence in the conclusion does not require a reevaluation or refiring of the rule.

## EXPLANATION SYSTEM

As important as the ability to make deductions is STAMMER's ability to explain its reasoning. The explanation facility is a part of the user-interaction subsystem. As system builders have little knowledge about the form of interaction users will find convenient, a production system architecture was chosen for defining the interaction subsystem. About two dozen productions were required, divided nearly equally between those interpreting user's commands and those interpreting his queries.

Recall that STAMMER functions by processing each message, reporting to the user all but the most minor conclusions (this is controllable by the user), and then prompting him for any questions he may have. Typical questions that a user might have are:

```
WHAT is the COURSE of SIGHTING3
Is RADAR the SOURCE of SIGHTING32
TELL me about SIGHTING5
WHERE was CONTACT2 at time 115
WHY is A00345  (an assertion)
HOW does rule ID-LANE apply to A0435
WHOSE TYPE is MERCHANT
WHO is HOSTILE
```

These questions forms are redundant, but add to the naturalness of the interaction. In particular, the "TELL me ..." command could be used in nearly all cases, but it often retrieves too much information, overloading the user. To illustrate how simple it is to add or modify these language forms, the internal LISP form of the above "HOW.. " query is:

```
(HOW "does rule" !RULE (= RUL) "apply to"
 !ASSERTION (= NODE): (RULEXP RUL NODE)).
```

In the above questions, the user only types the capitalized words. The system "guides" the user by typing the lower-case words. This guidance aids both the system and the user. The user gains the sense that the system is paying attention to him, anticipating him, and relieving him of some typing. The system need not have the capability to disambiguate such sentences as:

```
What is the course of signting3?
Sighting3 was on what course?
Sighting3 was travelling in which direction?
The heading of sighting3 was?
etc.
```

This form of interaction guarantees that the system will comprehend the question and be able to answer it. The user is not frustrated by finally phrasing a question in an acceptable form, and then having the system respond in some nonsensical manner. Of course there are means by which the user can abort his typein, change it, or determine the possible legal responses.

Most of the questions above deal with simple retrieval of information from the growing data base. Only the "WHY" query asks the system to explain or justify some assertion. Assertions are created by five different mechanisms. Ground assertions are those assertions which require no rule firings. Ground assertions might be i) facts from the technical base, such as "the maximum speed of vessel V is speed S," ii) facts received from messages, such as "the position of object O is latitude LA and longitude LO," iii) predicates requiring a simple computation, such as "X is less than Y," and iv) default assumptions, such as "vessel V is outside all merchant lanes" if the position of V is unknown. The explanation system clearly delineates all of the above types of assertions.

The explanation of a non-ground assertion or inference is more interesting. Associated with each inference is the collection of rules which bear upon it, together with the particular instantiating conditions of each rule. The same rule may be applied more than once, by having different bindings of its conditions, to yield the same conclusion. Since each condition of a rule is itself an assertion, the user can ask for a justification of any condition. This process can go as deep as the user likes until a ground assertion is reached.

Returning, for the moment, to the example given in the introduction, we could see the following interactions occurring after time 30.

```
Question? WHY is A0570
STAMMER applied the rule(s)
SPEED-CHANGED SPEED-CHANGED OUTSIDE-ALL-LANES

Question?PRINT rule SPEED-CHANGED
CONDITIONS:
 *SHIP is a contact
 *SIGHTING is a sighting of *SHIP
 *SIGHTING is the successor of *SIGHTING2
 *SPEED1 is the speed of *SIGHTING
 *SPEED2 is the speed of *SIGHTING2
 *UNLESS*
   *SPEED2 is roughly the same speed as *SPEED1
ACTION:
*SHIP is NOT a MERCHANT
CONFIDENCE: +.5

Question? HOW does rule SPEED-CHANGED
              apply to A0570
Which occurrence? 1
The rule was applied with the assertions:
A0237:CONTACT1 is a contact
A0504:SIGHTING12 is a sighting of CONTACT1
 .
 . STAMMER gives the bindings of the assertions
 .     to the conditions.
```

At one point in time, STAMMER may infer an assertion, say A0001, which it later believes to be false, as the result of later contrary evidence. This unbelieved assertion is not erased from the data base, for that would not allow the user to ask about it. The system informs the user that A0001 is no longer held to be true, although it once was. The user can find out why the assertion was believed by asking about the rules that led to its conclusion.

In addition to provided textual explanation, STAMMER can (if the terminal allows it) provide auxiliary graphic explanation. Maps can be drawn which indicate land masses, merchant lanes, storms, and ships. Although STAMMER will automatically give graphic data when appropriate, the user may always command the system to enter the pictorial mode. Once within this mode, the user can change the center of attention and change the scale of magnification.

Other commands that complete the user interaction subsystem allow the user to add rules, modify confidences, save states, enter INTERLISP, or receive a recapitulation of STAMMER's conclusions.

In summary, the user interaction system is easily modified and easily used. It gives the semblance of a natural language front-end without requiring a great deal of code space, programming time, or execution time. Although some information is withheld from the user (i.e. unreachable by any sequence of questions), such as the current suspended state of many rule firings or the order of rule firings, the user can determine any assertion and derivation that pertains to a conclusion of STAMMER.

CLOSING REMARKS

STAMMER is written in INTERLISP and runs in about 140K of core on a KL-10. The environment of INTERLISP makes it a pleasant and convenient language to code in.

In this paper we have shown how streams can be used to define a clean and efficient rule interpreter for a forward-chaining production system. Naturally occurring rules demanded the use of the non-monotonic operator UNLESS, whose semantics with respect to confidence calculation were defined. To support truth maintenance, the ideas of pulsars and delayed computation of confidences were introduced. Finally, derivation records were used to aid in the in-depth explanation of STAMMER's reasoning. The user interface was designed to provide convenient interaction while preventing ambiguity.

Future Needs

While the "suspension" interpreter is time efficient, it requires a great deal of space. A forgetting mechanism will be needed to discard old or inactive or unimportant suspensions.

Although the confidence mechanism appears to work, all of us hope that a method will be developed for evaluating evidence which is more natural or reason based.

Finally, and perhaps most critically, STAMMER needs to mix its data driven method for deriving conclusions with a goal directed understanding of events.

References

[1] Bechtel, R.J. and P.H. Morris, STAMMER: System for Tactical Assessment of Multisource Messages, Even Radar, NOSC Technical Document 252, Naval Ocean Systems Center, San Diego, May, 1979.

[2] Morris, P.H., Kibler, D.F., and R.J. Bechtel, STAMMER2: A Production System for Tactical Situation Assessment, NOSC Technical Document 298 (in press).

[3] Doyle, J., A Glimpse of Truth Maintenance, Proceedings of the Sixth International Joint Conference on Artificial Intelligence (August 1979), 232-237.

[4] McDermott, D. and J. Doyle, An Introduction to Non-monotonic Logic, Proceedings of the Sixth International Joint Conference on Artificial Intelligence (August 1979), 562-567.

[5] Rosenberg, S., Reasoning in Incomplete Domains, Proceedings of the Sixth International Joint Conference on Artificial Intelligence (August 1979), 735-738.

[6] Elschlager, B., Consistency of Theories of Ideas, Proceedings of the Sixth International Joint Conference on Artificial Intelligence (August 1979), 241-243.

[7] Shortliffe, E.H., Computer-Based Medical Consultations:MYCIN, Elsevier, 1976.

[8] Doyle, J., A Truth Maintenance System, Artificial Intelligence 12, (1979), 231-272.

[9] Kowalski R., Algorithm = Logic + Control, Comm. ACM 22, 7 (July 1979), 424-436.

[10] Warren D., Periera, L. M. and F. Pereira, PROLOG -- The language and its implementation compared with LISP, SIGPLAN/SIGART special issue (Aug. 1977), 109-115.

[11] Arvind, Gostelow K. P. and Wil Plouffe, An Asynchronous Programming Language and Computing Machine, Tech. Rep. 114A, Dept. of Info. and Comp. Sci., U. of Cal., Irvine (Dec 1976).

[12] Ashcroft E. A. and W. W. Wadge, LUCID, A non-procedural language with iteration, Comm. ACM 20, 7 (July 1977), 519-525.

[13] P.J. Landin, A Correspondence between ALGOL-60 and Church's Lambda notation, Part 1, Comm. ACM 8,2 (Feb. 1965), 89-101.

[14] Rieger C., Spontaneous Computation and its Role in AI modeling, in (D. A. Waterman and F. Hayes-Roth Eds.) Pattern-Directed Inference Systems, Academic Press, 1978.

# AN INTELLIGENT SUPPORT SYSTEM FOR ENERGY RESOURCES IN THE UNITED STATES

S. Rosenberg

Information Methodology Research Project
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

## ABSTRACT

This paper describes a frames based system for reasoning in a petroleum resources domain. By extending the notion of frames to include rule frames, which can then be interpreted and applied, expertise of various kinds can be directly encoded into the frame representation. Frame based rules are useful in encoding constraints, performing actions, noticing complex situations, and deducing solutions. By varying the interpretation of a rule frame, the same competence knowledge can be used in performing each of these tasks. Rules are able to use the frame based representation in finding other rules, avoiding most pattern-directed invokation. Making rules part of the frame based semantic structure may provide a natural way to encode plans and metaknowledge.

The Information Methodology Research Project has, as one of its focuses, the goal of developing intelligent information systems for dealing with energy resources in the United States. A first step in this process is the development of a small test bed system within a petroleum domain to provide capabilities currently either unavailable or performed by human analysts. In this paper I shall describe some of our results in using a frame representation methodology for capturing essential features of our petroleum domain. After describing our concept of a "friendly" representation I shall focus on the use of frames as rules, and the several ways this has proved to be a helpful and effective extension of the frame concept.

Our goals are ultimately quite practical; namely the transfer of A.I. "technology" into a real world domain. This imposes certain constraints on the design. Over 200 databases dealing with energy resources are already maintained by DOE. Our representation scheme must be able to use this existing knowledge. Similarly, there are limits on the types of information which can be collected. (For example, in the controversy over the recent U.S. oil shortage, not enough relevant information existed to determine the cause.) Of the data available, there are problems with validation, with information gaps, with variable definitions of terms, etc.

Several of our colleagues (Krishnan & Cahn 1979) are developing precise formal models for the flow of energy resources. Any representation must capture the features of such models so that existing databases on energy resources can be mapped into it. Real data in these databases is often "messy." Crucial information is sometimes missing, incomplete, or invalid. To be useful, the representation scheme should provide help with these problems. Methods such as default procedures, cross validation checks, caveats, and constraint monitoring are necessary to augment the raw data. We use FRL (Frame Representation Language) as the basis for our representation. In FRL, a frame can make use of inheritence, default values, procedural attachments, etc. This augmented notion of what a data object is allows us to create the types of "friendly" representation we need.

A "friendly" representation takes the burden of performing routine, if sometimes complicated, functions, from the reasoning component or user. It is able to massage or augment the data to provide more complete information. Some of these functions are quite simple, such as providing aggregated information, or default values. Some are complex, such as adjusting the representation by changing deduced conse-

Prepared for the U.S. Department of Energy, Technical Information Center, under contract W-7405-ENG-48.

quences when erroneous facts are corrected. Some we don't know yet how to do, such as handling fuzzy information. The net effect of such a friendly system is to allow a user or reasoning component to focus on doing higher level tasks, while leaving lower level information processing to the representation system. In effect, we propose that in many domains, semantic representations must function dynamically, drawing on interlaced procedural and world knowledge to provide a solid basis for higher level reasoning. An essential feature of such representations is the existence of semantic knowledge in a useful procedural form. In FRL, the augmentation of an attribute/value relation with procedural attachments provides this feature. Thus, if-added attachments can be used to encode and execute constraint relations between frames. For example, in the portion of a frame for Iran shown below, the if-added attachment on the carryover slot automatically updates aggregation information on world supply. The default predicate on the production slot provides a typical value when this data is reported late. The ability of human experts to fill in information gaps, assess credibility, etc. forms a central aspect of their expertise. Consequently, developing such a flexible representation forms a basis for developing expertise in our domain.

```
IRAN
.
.
production    $default  (use-last-month-value)
carryover     $if-added (add-to-world-supply)
.
.
```

The basic semantic system is constructed using FRL (Roberts and Goldstein, 1977). FRL is a Frame Representation Language based on Minsky's (1975) notion of frames. Goldstein and Roberts have developed this into the working frame system which forms the basis for our knowledge representation. FR' is a sophisticated, higher level language designed for the representation of knowledge in a variety of domains (e.g. NUDGE (Goldstein & Roberts 1977)). It provides a hierarchically organized, frames-based semantics with inheritance and procedural attachments among other features. FRL is in turn written in LISP, hence is compatible with normal LISP code.

FRL extends the traditional characterization of properties as attribute/value pairs by allowing properties to be described by comments, abstractions, defaults, constraints, indirect pointers from other properties, and attached procedures. A value of a property becomes one of a range of potential descriptors. A frame can be thought of as a named collection of slots which form the semantic definition of a concept. These slots define the properties of the frame (i.e., they form a list of such properties). Each property can have many values. A slot (= property) can be specified further through the use of an arbitrary number of associated user and system defined "facets." One of these facets will be the traditional "value" of attribute/value pairs in property lists. Useful system defined facets are: Value, which contains the value of that slot; Default, which specifies a default value; Require, which specifies procedural constraints on the values for that slot; If-Needed, which specifies procedures that compute a value for the slot; and If-Added and If-Removed, which specify actions to be taken when a value is added or removed. Notice that many of these facets are procedural attachments. Each slot can have associated procedures which can perform calculations when required. Thus a frame in FRL is more than a simple datastructure.

FRL allows concepts (represented as frames) to be arranged in an inheritance hierarchy using the AKO (A Kind Of) slot. The value of this slot is a generic frame of which the current frame is a specialized instance. Thus the frame system forms a tree structure. Generic information is stored higher up in the hierarchy and shared by frames lower down; specialized frames specify new distinguishing knowledge. The generic knowledge, including computational procedures, is inherited automatically. The inheritance can be restricted, if desired.

A small testbed model provides a useful

domain for extending the scope of a "friendly" system through the development of frames based reasoning. Information is organized around the major semantic categories of site, area, and company. These represent the primary physical loci where we wish to track petroleum (Rosenberg, 1979). A site represents any actual physical location at which oil is handled or consumed, such as a city, port, tank farm, refinery or oil field. (Foreign countries are currently treated as single sites.) Areas, such as states, are considered to consist of a set of sites physically located within their boundaries. Companies represent ownership of either a collection of sites in arbitrary locations, or other companies.

The testbed serves as a domain for developing a frame based rule interpreter for reasoning about energy scenarios. Suppose, for example, an overseas supplier of crude oil, such as Iran, decreases supplies to U.S. sites. Figure 1 below, shows a portion of the petroleum flow network. In it, Iran and Dallas supply the site of Newark. If Iran reduces production, are there alternate supplies of petroleum? In dealing with this and other questions, we will need to do several things. First, we need to model this reduction in the shipment of oil. This change in production will alter some of the constraints we have set in our database. We must then notice when problems such as reduced production occur. Next, we will need to find alternate sources.

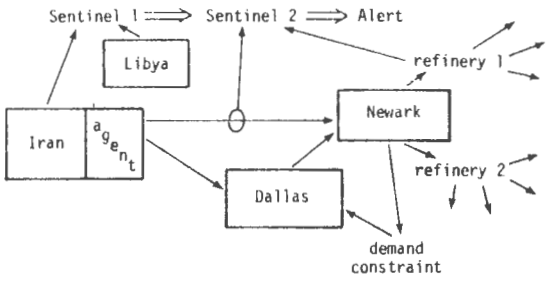



FIGURE 1

To deal with these four problems of encoding constraints, driving our model, noticing the development of important situations, and deducing solutions, we have found it useful to extend the concept of frames to include rules. By creating a class of frames called rules, and varying our interpretation of these rules, we can do all four of these tasks. All knowledge is represented as frames. Rules are expressed as productions (Newell & Simon 1972).

The rule syntax is:

```
(Rule type (vars)
      ((Frame Slot Tests)     ; Condition
        (Body)(Body)))        ; Action
```

More complex conditions can also be expressed, e.g.:

```
(Rule type (vars)             ; AND form
      ((and
          (Frame Slot Test)
          (Frame Slot Test))
        (Body) (Body)))
```

These productions are in turn translated into rule frames with condition and action slots. The only indication that such declarative knowledge is a rule consists in the value of the generic pointer (e.g. = AKO rule). Thus rules are semantically defined but represented as declarative knowledge in the frame tree. All features of the hierarchical frame representation are available, such as the use of inheritance, the ability to use semantic relations in determining an appropriate rule, and so on. Rule frames contain competence knowledge. To use rules, a rule frame is interpreted as a procedure, with the slot values controlling the interpretation. Thus a condition slot causes a condition to be tested; the action slot specifies the action to be performed and so on.

The type slot on a rule frame holds information about the way the rule can be used. By changing this value, we can vary the interpretation of a rule. Variable interpretation allows us to use the competence knowledge expressed in a rule frame in different ways.

Thus Rules come in various flavors. Some always erase themselves after success. Others do not. Some trigger on the removal of information, others on the addition of new knowledge.

Some function as expectations, others fail if the information does not already exist in the database. And so on. These flavors are all useful for different purposes. By using variable interpretation of rule frames, all these variations are controlled by the application of the type knowledge in a rule frame. By modifying this value, the same rule can be used in different variations, depending on our goals.

Many of the relations between semantic entities in our model can be encoded as constraints. For example, a refinery fire will alter the amount of oil a consuming site needs. This in turn affects its relation to its suppliers. Changes in information can cause propagation of these constraints to occur (although in a much simpler form than Doyle (1978) proposes.) Simple constraints can be encoded directly as procedural attachments to frames. A change in the information content of one frame triggers a simple predicate which then modifies the information available at another frame. (For example, the if-added procedure in the Iran frame.) Some constraints either are generic (i.e. apply to a large class of items) and/or require some deductive capabilities. These are encoded as (antecedent) rule frames, with triggers in one frame, and the ability to modify other frames. Figure 2 presents the constraint of Figure 1 in more detail. Here, a generic constraint exists whose purpose is to see that the petroleum needed by a site is equal to the amounts its suppliers intend to provide. This constraint places a trigger in the generic site frame. Hence it monitors all sites. If a particular site, such as Newark, changes its monthly needs, this trigger is inherited, and fires. The constraint then tries to adjust supply among the suppliers to Newark to correspond to demand. These values are used in turn in "shipping" oil properly. Drastic breakdowns in supply/demand relations are treated as alerts, rather than constraint violations.

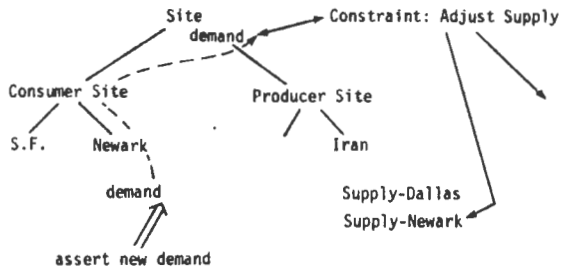



FIGURE 2

Actions such as "shipping" oil, are performed by using rules as agents. An agent consists of a (set of) rule(s) together with an environment, usually a single frame (although sometimes a cluster of frames). In Figure 1, an agent is shown "attached" to Iran, which ships the oil. Like the constraint just discussed, this agent could have been attached to a generic frame, such as the site frame, and taken responsibility for shipping all oil. By monitoring production and carryover in the Iran Frame the agent determines when to ship oil. At the right time the oil is allocated among the sites supplied, and relevant information is modified on the various frames involved, to indicate this. (i.e. agents function as state change operators.) This action can in turn trigger a new agent. Using rules as agents provides a method for driving our model to simulate the changing state of our domain.

Given a database of changing information, we want to provide some capability to monitor important developments, and alert us when necessary. (For instance, drastic changes in supply are beyond the scope of constraints.) Many subtle problems can arise in providing such alerts. For example, small reductions in supply by various producers, together with changes in demand at several sites can result in a severe shortage at one particular site. However the change at any one other site is not significant in itself. Such dynamic noticing is done by treating rules as Sentinels (Rosenberg, 1979) which leave active expectations in the data base.

Sentinels are created by varying the interpretation of rules to encode expectations. Sentinels draw on the information available about the frame organization to place data driven triggers in appropriate semantic locations. Thus, only input with a high liklihood of fulfilling the conditions triggers a sentinel. Frame-based sentinels function as instance driven demons. Consider a demon attached to the instance slot of a particular frame. Whenever a new instance which inherits from that frame occurs, the demon matches its pattern against that new frame. However, it does not consider any other new frames which do not inherit from the frame it is attached to. Such a demon or antecedent rule, if attached to the top-most frame in the heritage tree, is equivalent to a traditional demon (Charniak, 1972). Otherwise the demon will match against only a selected subset of new input. By choosing the appropriate frame to attach such demons to we insure that they match against only likely candidates with precise semantic relations to the expectation frame. Sentinels are a type of rule, and can be created and manipulated by other rules or Sentinels. This provides a flexible mechanism for monitoring complex conditions and providing alerts.

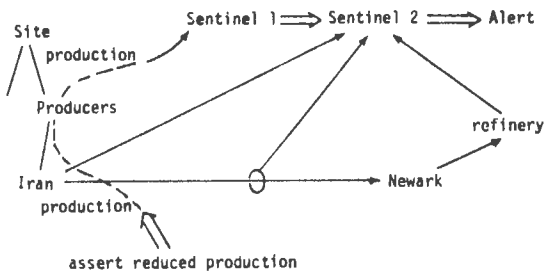



FIGURE 3

For example, suppose we wish to be warned whenever a consuming site such as Newark, will experience a severe shortfall in supply. In addition, we would like to have as much advance warning as possible. Figure 3 above shows in more detail the Sentinels from Figure 1 which do this. By taking advantage of the semantic structure a frame hierarchy provides, we can create a sentinel which places a trigger in the generic producer site frame. This trigger will be inherited by all production site instances. Reductions beyond some local criterion expressed on the individual sites will trigger the sentinel. Thus the assertion of reduced production into the Iran frame causes the conditions of this Sentinel to succeed. A reduction in production by one producer however, (or even several) does not necessarily mean a shortage of oil at a site. What sites are dependent on this producer? How large a proportion of their requirements are met by this producer? Are their other normal suppliers capable of making up the slack? Can suppliers who do not normally ship to this site do so? Are there surpluses elsewhere which can be rerouted? Before a shortage warning can be issued, questions such as these must be considered. The first sentinel will examine the sites supplied by Iran, and try and determine if any are excessively affected. If, for instance, Newark was solely dependent on Iran for oil, a reduction in Iranian production can reliably be used as sufficient evidence for a warning. If there are several suppliers to Newark, the best choice may be to monitor Newark's supply more closely. In this case, Sentinel1 creates another sentinel, Sentinel2, to monitor both shipments to Newark, and demand at Newark, directly. Iranian production is also monitored. If production returns to normal in Iran, this Sentinel will erase itself. The ability of sentinels to manipulate their own conditions, create new sentinels, and call on deductive procedures provides a flexible, powerful mechanism for encoding noticing. In some cases the triggering conditions form a precise scenario. In this case, Sentinels can be thought of as frame (or script) instantiation. Its slots serve to define the expectations the sentinel searches for. In other cases, the situation is less well structured, and Sentinels simply provide a way to use rules in encoding complex sets of expectations, some of which

are determined based on the success of prior ones. If Sentinel2 does notice a drastic supply imbalance at Newark, it will give an alert.

Once an alert has been given, one of the uses for rules in our testbed is in answering questions such as those posed earlier about possible alternative supplies. What alternative supply sources are available for Newark? We are exploring the uses of frame based rules for goal directed reasoning. Such consequent reasoning is transformed into antecedent reasoning by treating the assertion of a condition as an implicit goal. While appropriate rules can be found by some variant of pattern matching (e.g., planner, or production systems) we take advantage of the organized semantic structure to have alerts assert information into a location where it can directly trigger the appropriate rule(s).

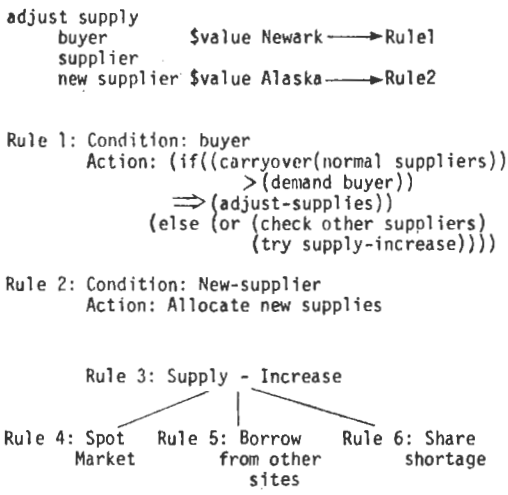



FIGURE 4

Figure 4 shows part of an adjust-supply frame. Sentinel2, on noticing an oil shortage in Newark, can assert Newark into the buyer slot of this frame. Other slots on this frame can serve either to encode more complex conditions or as buffers. The addition of this information triggers a Rule, R1. By contrast, the assertion of a new supplier would trigger a different Rule, R2. R1 first collects all normal suppliers to Newark, and if these have sufficient stocks, adjusts supply using these stocks. Otherwise, it can try two alternatives; checking other suppliers which do not normally ship to Newark, and trying alternative methods for increasing supply. We show in more detail only this second alternative. (Try supply-increase) will cause all rules which inherit from the supply-increase frame in figure 4 to be evaluated. Thus, although we do not know which specific rule might be relevant, we use the frame hierarchy representation to allow rules to call on other classes of rules known to be helpful in achieving their goals. In effect, the use of frames such as adjust-supply, and a rule hierarchy, allows us to create small contexts of relevant rules and pertinent information. The rules collected through [R3: Supply increase] are [R4: buy on spot market] [R5: borrow from carryover stocks at other consuming sites] and [R6: Reroute shipments to share the shortage equally]. Any or all of these rules might succeed. However it is obvious that even so, these provide only a first order solution. For instance, buying on the spot market can drive up the price of oil. At some point this becomes less desirable than reducing demand by restricting gas station hours. Similarly, we may always want to "share" a gas shortage across all sites in the U.S. Considerations and interactions like these are difficult to capture directly in simple rules.

A first order solution is to augment rule frames with caveats that must be satisfied in order to use the rule. (Goldstein & Grimson, 1977) For instance, the caveat for buying on the spot market might require paying a price lower than a maximum set by DOE. However such a solution does not handle more complex interactions among sets of rules. Fortunately, there is an obvious place for such planning or meta-knowledge. Since Rules are organized in a semantic hierarchy, more generic frames, such as the supply-increase frame, provide an appropriate place for planning information. If the Rule hierarchy is deep, several layers of plan-

ning information might be accessable. We are just beginning to explore the feasibility of this method of using planning knowledge.

To conclude, we have found that extending the notion of frames to include rules has proven useful in capturing much of the common sense reasoning that occurs in our domain.

## References


Charniak, E. "Toward a Model of Childen's Story Comprehension," AI TR-266, MIT, December 1972.

Doyle, Jon, "Truth Maintenance Systems for Problem Solving," AI TR-419, January 1978.

Duda, R.O., Hart, P.E., Barrett, P., Gaschnig, J.G., Konolige, K., Reboh, R., and Slocum, J., "Development of the Prospector Consultation System for Mineral Exploration," SRI, October 1978.

Goldstein, I.P. and Roberts, R.B. "NUDGE: A Knowledge-based Scheduling Program," AI Memo 405, MIT, February 1977.

Goldstein, I., & Grimson, E. Annotated production systems. A model for skill acquisition. Proceedings of the Seventh International Joint Conference on Artificial Intelligence, August 1977.

Krishnan, V.V., and Cahn, D.F. "An Aggregated Vectorial Model of Petroleum Flow in the United States," LBL-8874, University of California, Lawrence Berkeley Laboratory, March 1979.

Minsky, M. "A Framework for Representing Knowledge," in P.H. Winston (Ed.) The Psychology of Computer Vision, McGraw-Hill, N.Y., 1975.

Newell, A. and Simon, H.A. "Human Problem Solving," Prentice-Hall, N.J., 1972.

Roberts, R.B. and Goldstein, I.P. "The FRL Manual," AI Memo 409, MIT, June 1977.

Rosenberg, S. "A Knowledge Based System for Providing Intelligent Access to a Petroleum User Database," LBL-8720, University of California, Lawrence Berkeley Laboratory, January 1979.

Rosenberg, S. Reasoning in Incomplete Domains. Proceedings of the Sixth International Joint Conference on Artificial Intelligence, 1979.

# Qualitative Reasoning about Time Series.

Dr James L. Stansfield
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge MA USA 02139

## Abstract

Many large systems, such as the economy, present an analyst with a large amount of numeric information. I discuss the hypothesis that an important component of an analyst's expertise in explaining the behavior of such systems is the ability to describe features and structure in data that is represented graphically. Symbollic descriptions of graphs form a basis for qualitative reasoning about numeric data and interface with the use of knowledge-based models of complex systems. I present a method for describing graphic information at multiple scales of detail. Descriptions are represented in a frame based representation language (FRL). Some extensions needed for a program that can find meaningful relationships between syntactic forms in graphs are also described.

## Introduction

It is currently impossible to construct a complete model of a system like the economy. A reasonably complete model with all the relevant data would still be very hard for a human expert to understand - almost as hard as to understand the real economy. So how does an expert reason in such a situation? What does an explanation of a large amount of numeric data look like and how can we construct one? How is data summarized and described? I am investigating these issues in the context of explaining graphic data about a sector of the corn supply system. In this paper I propose that qualitative descriptions of graphic data are a central component of the process. I describe a method for producing descriptions at multiple levels of detail and explain how these will fit into the design of an expert system.

---

Acknowledgements

*1. This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Office of Naval Research under contract N00014-77-c-0389.*

*2. I am grateful to Patrick Winston, Kent Stevens and other members of the MIT AI Laboratory for most helpful conversations about this work.*

Manipulating descriptions of graphs provides a different emphasis from other work on qualitative reasoning about systems. Rieger [1976] has applied a vocabulary of causality primitives to describe some simple mechanisms such as a steam boiler. The descriptions were used to simulate the behavior of the mechanism through a series of time ticks. A program by De Kleer [1978] reasoned about a description of a simple electronic circuit to determine the change in voltages and currents for a given perturbation. Both systems used complete descriptions of the mechanism and both attempt to predict what will happen next. The systems being processed are well-understood. In contrast, the corn supply system is large and incompletely understood. Rather than projecting a completely described state forward into a deterministic future an analyst navigates through a large amount of graphic data to arrive at a set of partial qualitative hypotheses about what happened. These suggest further examination of the data for corroborative evidence, or refutory facts.

## Qualitative reasoning about graphs

Information about the grain market from sources such as government reports and commercial newsletters is of two broad types; tables of figures, and commentary. Tables give values over time of variables such as stocks, prices, and shipments. Meta-information can be associated with the tables giving typical values, describing what constitutes anomalous behavior, showing which figures are available, and relating the tables to each other. Commentary performs two functions. It points out some important features of the figures and it attempts to relate the by means of a causal argument.

Commentary relies heavily on a vocabulary of qualitative concepts for describing the behavior of quantitative variables. This includes terms like "increasing", "decreasing", "accelerating", "large", "fast", "sharp", and "moderate" which describe the structure of data and approximately quantify it. The number and variety of qualitative terms and the way in

which they mix with other concepts is evidence that qualitative reasoning about numeric variables and their changing form is a central part of human reasoning. Trading may be frantic or slack, there may be a glut of supplies, rainfall can be heavy and a drought can be severe. Although this large set of descriptive terms can be condensed into a few neutral ranges such as, small, moderate, and large, most terms apply only to particular classes of concept and have special overtones of meaning. "Frantic trading" implies a rush for time. "Severe drought" refers to the harm caused by the lack of rain.

Graphic representations are an important tool for analyzing and describing complex systems. An important reason is the visually striking way they display certain features of the data. As a visual representation, graphs allow direct access to absolute size, proportion, shape, peaks and troughs, gradient, and time relationships as well as to non-localized properties such as smoothness, sharpness, waviness, randomness and regularity. The visual system highlights important structure in an otherwise uniform array of numbers. These qualitative concepts are experienced through other than visual channels. Kinesthetic feedback allows us to drive a car smoothly and we recognize sounds with a great variety of structure. The ability to describe the form of a changing variable and to relate the description to others and to a stored body of knowledge must be a significant component in our natural reasoning. How do we manage the interface between numeric variables and logic? Designing a program to produce descriptions and explanations of the behavior of the corn market from graphic data is a way to investigate the general problem in a particular domain.

**The domain**

The domain concerns the movement of corn from farmers through the transportation system to domestic users and exporters. It is considered from a standpoint of broad patterns rather than exact detail. This is how an expert must reason about the domain as a whole. A finer grain of detail requires a smaller area of attention. Graphs of stocks, flows and prices are provided for a selection of strategic and representative locations and figures are given for exports from major regions and for new export sales for the US as a whole. Samples of these variables are given for weekly intervals over a period of just over a year. The time span includes a variety of significant events and the sampling interval provides enough data to characterize each event.

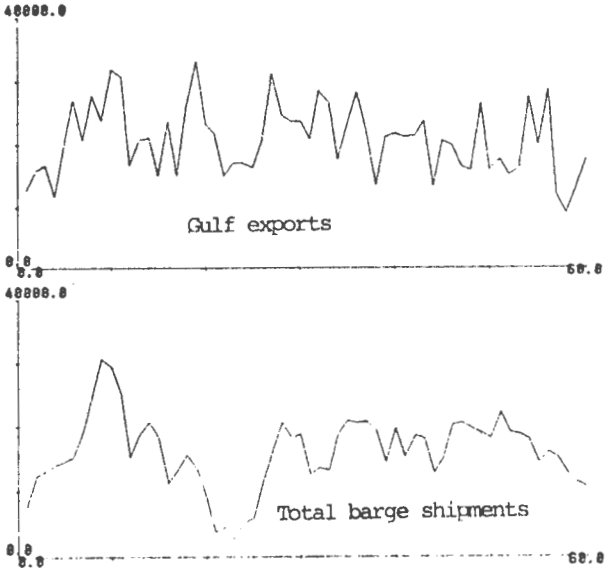



Figure 1.

As an example, figure 1 shows graphs of total barge shipments to Gulf coast ports, and exports from these ports. We would expect in general that shipments would follow exports quite closely. However, other factors affect the relationship. These effects can be hypothesized and supported by data in other graphs. It is clear from the figure that the grain market showed considerable structure during this time period. This is quite typical. Here are some events that occured.

A period of high exports caused large flows of grain along the rivers.

Severe freezing of the rivers caused a transport blockage. This backed up stocks inland, and tightened supplies at Gulf Coast ports, raising prices at the coast and depressing them inland. Alternative transport routes were strained, and the uncertainty discouraged new export orders for a while.

Seasonal opening of the Lakes siphoned off some of the supply so that less corn moved to the Gulf from Northern prooduction centers.

Onset of harvest allowed a high flow of grain without raising the price.

The domain has many valuable properties. It uses real data about a real situation of definite interest to analysts. At the same time it is well circumscribed. This is because of the abstraction away from details and because the domain is open in many places. It follows that some large-scale features of the data, and many detailed fluctuations, are unexplainable. Even

a human expert faces this. An appropriate success criterion for an expert program is to notice and explain some typical situations and some which are clearly anomalous. There will be a middle ground of mundane situations or situations for which the program should either admit lack of data or knowledge, or recognize that the data is too ambigous. A further property of the domain is easy of extensibility. Some choices are to cover a longer period and include seasonal information, to consider several crops and the interactions caused by competition for storage and transport, and to increase the number of places in the model.

### Graph Description

In the case of systems whose behavior is to a large extent represented by graphic data, I hypothesize that symbollic descriptions of the features present in the data, their rough proportions, and their positional and subpart relationships, are a prerequisite for recognition, comparison and meaningful reasoning about the behavior of the systems. While statistical techniques, and correlations can provide information, they are not sufficiently logical to handle the interactions and exceptions that occur in a complex domain. The purpose of the graph description program is to take part of a graph and return a symbollic description of the features present for use by the domain expert. These descriptions are constructed from FRL frames [Roberts, 1977] which represent the features and assertions about them. Primitive syntactic features of graphs include segments of various kinds and special points such as peaks, lows, and points of maximum or minimum gradient. Frames for these can be modified with properties such as increasing, decreasing, concave, convex. They may also have numeric information giving the ordinate and abscissa and rate of change. These numbers are then quantized into five ranges according to the typical values the graph takes. Quantization intervals can be provided externally as a property of a graph or type of graph. For instance, the price of corn might be considered high when it is over $3.00. Alternatively, they can be calculated from the data by examining the distribution of values and gradients. Quantization is context dependent. Typical intervals for the time period between 1970 and 1980 may be different from intervals obtained for a different time period. Knowledge about seasonal information will be incorporated eventually. Similarly, a given price may be considered absolutely high but low for the time of year. For such contextual quantization, the assertion that a variable has a quantization x must be annotated with information about the context.

Primitive descriptive elements are combined to generate a small vocabulary of larger structural types. An increasing segment is a concatenation of increasing atomic segments. Basic types are the TREND, the HILL, and the VALLEY. These can be modified with properties such as sharpness, width of shoulder, and asymmetry. A related vocabulary was developed by Hollerbach in connection with the analysis of Grecian urns [1975]. The following frames describe a trend in Chicago corn prices which contained several hills and valleys all lying within a narrow band of increasing price.

```
TREND-1
  AKO:  TREND
  VARIABLE:   PRICE-2
  VARIABILITY: 80 cents
  BEGINNING:   May 1973
  ENDING:      Jan 1975
  GRADIENT:    5 cents/month
  PARTS:  HILL-3, HILL-4, HILL-5
          VALLEY-10, VALLEY-11
  PARTOF:  GRAPH-6

PRICE-2
  AKO:  PRICE, VARIABLE
  COMMODITY:  CORN
  LOCATION:  CHICAGO
  GRAPHS:  GRAPH-6

HILL-3
  AKO:  HILL
  VARIABLE:  PRICE-2
  QUALITIES: SHARP-10
  PARTOF:  TREND-1
  PARTS:   SEGMENT-7
           SEGMENT-8
  BEGINNING:  Mar 1973
  ENDING:     Oct 1973
  PEAK-AT:    Aug 1973
  MAXIMUM-VALUE: $3.00
  HEIGHT:        80 cents
  FOLLOWED-BY:   HILL-4
```

There is no unique way to parse a graph into these larger structures since multiple descriptions are reasonable. Figure 2 shows a case where an export graph can be described as three consecutive hills. The same approach applied to the graph of barge shipments to the Gulf gives two hills. The hills in the two descriptions do not correspond. An alternative description of the shipments as a hill followed by a valley followed by a hill does result in an appropriate match. An analyst program would explain the first and third sections of this situation as

barge shipments following export highs. It would look for possible causes of the anomalous middle section and find that a fall in river capacity restricted transport to the Gulf preventing shipments from following exports as otherwise expected.

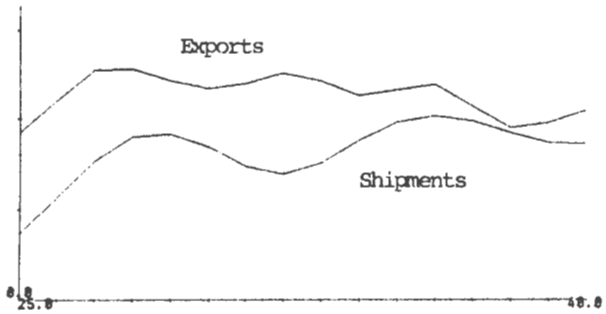



Figure 2.

Noise interferes with the description process. An exports trend is likely to be broken during some weeks so it cannot be found simply by noticing consecutive changes in one direction. Noise can also seriously mislead a reasoner which attends to every detail. A troublesome situation is shown in figure 3. Exports increase and shipments to the Gulf follow roughly, but in some weeks the two move in opposition. It is a mistake to try to explain these cases since they are really too transitory at this resolution and without more detailed knowledge. For these reasons excess noise must be smoothed from the data. Smoothing is done by convolving the graphs with a Gaussian mask. Figure 4 shows the previous example with effects smaller than one week smoothed away. The ups and downs have disappeared and the graphs are sufficiently clean to trigger rules that will produce a simple explanation.

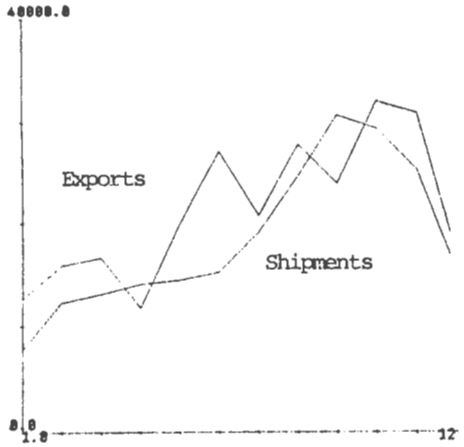



Figure 3.

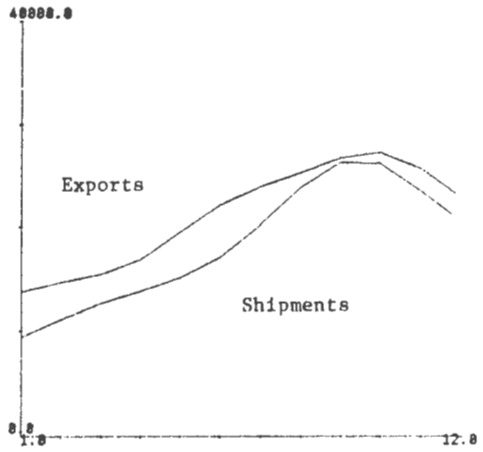



Figure 4.

An expert attends to large scale features before small scale ones. The broad description is a helpful framework for examining details. I use the smoothing mechanism to handle multiple levels of detail by employing smoothing masks of various scales. The process was suggested by analogy with edge-detection in vision processing where an image is convolved with Gaussian filters of different scales [Marr & Hildreth, 1979]. Gaussian filters are the optimal compromise between a spatially localized filter and a frequency localized filter. In my case, spatial corresponds to the time axis of a graph and frequency to the period of change of its value. A single frequency filter, used to pick out effects of say a one week period, would sum effects from the entire time period considered. In other words, it would have zero spatial localization. Commodity supply analysts use simple moving averages or sometimes moving averages with ad hoc weighting. These are not optimal. Though not perfect, the analogy with edge-detection suffices. Edge are fairly sharp phenomena whereas underlying causes in the corn system may be gradual. Nevertheless, there are causes at every different scale of resolution.

As the smoothing mask grows to cover more of the time-axis, the higher frequency features of the graph smooth away gradually. Figure 5 shows three stages in this process. A large scale hill with several perturbations will gradually lose its perturbations. They do not disappear suddenly, nor at the same time. Individual features have their own fade out scales. It is impossible to make an a priori selection of scales that results in a set of cleanly smoothed graphs each with features only of that scale or larger. How then should we proceed?

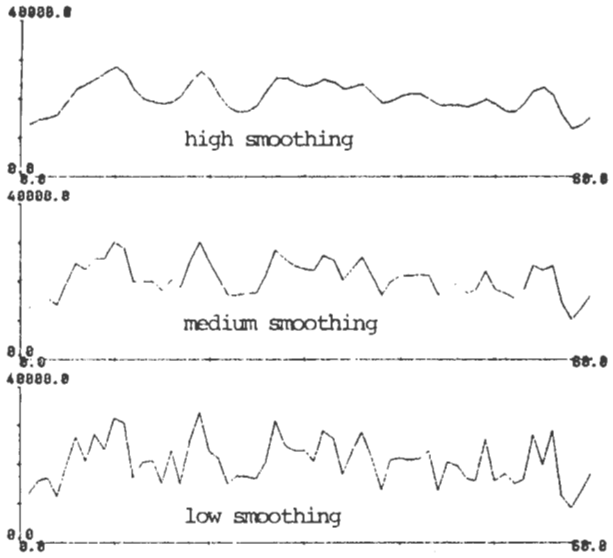



Figure 5.

My approach is to consider each feature independently and determine how it fades. For this purpose, features are considered to be segments of the graphs with only positive or only negative gradient, and segments which are concave upwards or are concave downwards. These segments are delimited by points at which either the gradient crosses zero or the second derivative crosses zero. They are turning points or points of inflexion. Figure 6 shows a three-dimensional representation of this. The surface is composed of a set of graphs placed side by side. Each graph is the second derivative of the original smoothed with a filter of scale sigma. Sigma decreases as we move into the page. This representation clearly shows how the features fade away as sigma becomes larger. Figure 7 shows is a contour map showing where the second derivative is zero. One axis is the time axis and the other axis is sigma, the scale of smoothing. At any particular scale, the segment between two adjacent zeros represents a segment of the original graph which is concave or convex vertically. As the scale increases, some of these features fade away completely. Each feature has an approximately triangular contour. I call the tip of the triangle a *vestigial point* since the last vestiges of the feature disappear at that scale. In fact, the feature loses its significance before then. To examine this one must look at the rate at which the value changes sign as it crosses the zero.

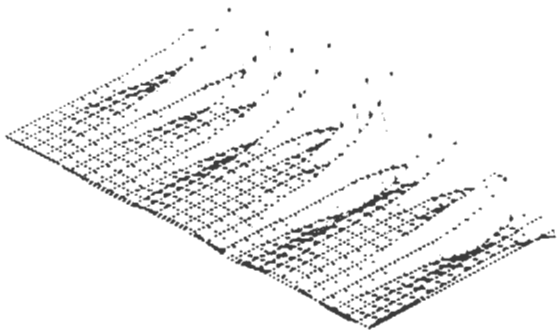


Figure 6.

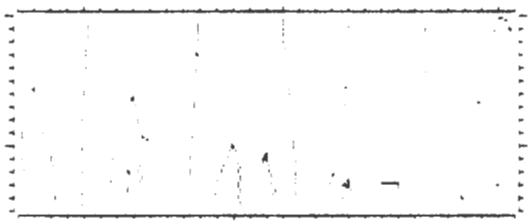


Figure 7.

This technique leads to descriptions of features which cut across all scales. By representing the shape of the triangle with a few parameters we describe just how prominent the feature is at different scales in a single description. The representation is feature oriented rather than scale oriented. It can be used to find the most global features needed for an explanation at low level of detail. Then, more detailed perturbations can easily be accessed.

For presentation purposes, figure 6 shows more levels of scale than are really need to construct a multi-scale description. It is possible to do even less processing. Once features are located at a fine scale, their zero crossing contours can be followed around the triangular outline so that at large scales, smoothed values need only be calculated close to the contour. Since these are the ones which require most computation, the saving is large. Similarly, the interval between scales is really logarithmic. At large scales, intervals need not be as close.

A further method also reduces computation. Since convolution is associative and the convolution of two gaussians is a gaussian with larger scale than both we have the following result where $\ast$ is the convolution operator.

GRAPH $\ast$ LARGE-GAUSSIAN
= (GRAPH $\ast$ (SMALL-GAUSSIAN $\ast$ SMALL-GAUSSIAN)
= (GRAPH $\ast$ SMALL-GAUSSIAN) $\ast$ SMALL-GAUSSIAN

Convolving with the small gaussian first returns a graph whose values are used many times each in the second convolution. This results in increased efficiency by saving repeated calculations.

### Graph organisation

Besides a frame description of the structure of a graph there must be assertions which relate graphs to their meaning. Typically, some graph might represent the price of corn at Chicago between time-a and time-b at weekly intervals and smoothed with a scale of two weeks. This information is needed to access graphs and in deciding what to do with their descriptions. There are also relations between graphs. Since St Louis barge shipments are a part of total barge shipments, we may construct a new graph showing the proportion or the two with respect to time. Or we may subtract them to calculate the amount of grain shipped from points other than St Louis. Most analysis situations have a number of operations that may represent the information in a more suitable form. Graphs produced in this way must have assertions relating them to their sources.

### Representation

An analysis module is to be written to take these graph descriptions and interpret them with qualitative explanations. Descriptions will trigger rules expressed as frame structures and implemented in a system of *sentinels*. The remainder of this paper discusses the representation scheme and the sentinel system. Since the goal is to develop an expert program and not to investigate representation problems per se, the approach is to use what is available rather than to investigate the many representation issues recognized. Most of the descriptions, assertions and rules are easily represented using FRL. Some issues, however, required extensions.

An FRL frame is effectively an item with a property list of properties and values but also has *facets*, *inheritance*, and *procedural attachment*. Each property of a frame has a slot associated with it. The values of the property fill only one part (or facet) of the slot. Other facets contain different types of information about the slot. Inheritance works by using the AKO slot. If FRAME1 has an AKO property whose value is FRAME2, FRAME1 will inherit all the property value pairs of FRAME2. Procedures can be stored in special facets of a slot. An important type triggers whenever a new value is added to a slot. Since procedures are inherited, those that describe generic concepts can be applied to all instances whenever the required information is asserted.

An important convention for using FRL frames, suggested by Roberts, is followed in this work. Suppose we wished to assert that farmers sold corn heavily on July 15th. On one hand, we need a convenient way to retrieve the assertion starting from a frame that represents the farmers. It is reasonable to give FARMERS a SELL slot whose value is some instance of corn. But we also need to represent the time of the SELLING and the fact that was heavy, so we need a separate frame for it. The solution is to have both the SELL slot and the SELLING frame and to annotate the farmers sell corn assertion with the SELLING frame.

```
FARMERS-1
  SELL: CORN-2 (SELL-3)

SELL-3
  SUBJECT: FARMERS-1
  OBJECT: CORN-2
  BE: HEAVY
```

Figure 8 shows how this can be represented with a diagram. The method allows us to have a frame for the farmers and a frame for the sale with equal status.

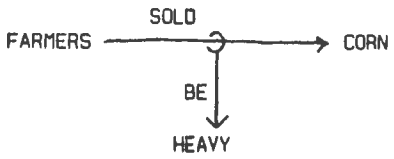



**Figure 8**

An annotation represents the assertion made by the corresponding frame- slot-value triple. Access to them is critical for representing many types of statement and for recording logical dependencies between statements. Figure 9 shows how annotations can represent that one event caused another. Notice in this example, that is is the fact that the selling was heavy that was caused by the harvest progress. Annotations allow us to focus on an assertion and say something about it.

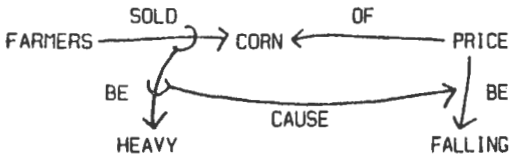



Figure 9

It is important to distinguish between those assertions in a frame which describe the concept the frame itself represents and those which say something about that concept. "The price of corn at St Louis increased yesterday" asserts a fact about a concept which is itself described by assertions. Figure 10 shows this pictorially with internal assertions separated from external ones. I implement this with partitions [Hendrix 1975], attaching a special partition to a frame to describe its internal structure. A partition should be a frame object just like any other frame. Another use of partitions is to represent an entire statement such as the whole of Figure 9. This statement may then suggest or justify some other statement. A hypothesis, situation, belief structure, or argument may then be represented using a partition of such statements. At this point there are three levels of partition, each used for a different conceptual purpose.

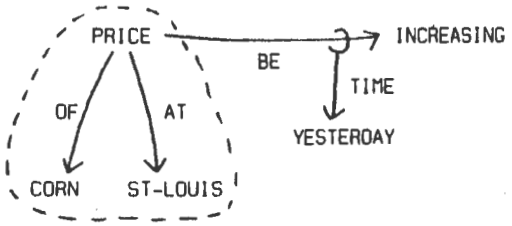



Figure 10

**Sentinels**

A sentinel [Rosenberg 1978, Stansfield 1978] is a form of production rule which triggers when a set of assertions in the frame data base matches the condition of the rule. The sentinel system described here overcomes difficulties in writing these rules directly with the if-added mechanism of FRL. Several issuse arise. One concerns the use of variables and the other concerns control. There is also an interaction between the inheritance hierarchy and productions which had to be catered for by the system. I next consider these problems.

Suppose we are using if-added procedures and want to notice when assertions are found that match two template assertions. Templates are frame-slot-value triples and might contain variables. A value is added that causes the if-added procedure for the first template to trigger. It binds a variable and creates a new if-added procedure to watch for values matching the second template. The variable bindings must be consistent with the first assertion. A value that triggers the second assertion may not be asserted yet, so the new if-added may not execute immediately. In any case, it is unlikely that it will run in the environment it was created in. This means that we cannot use a fluid variable in the code of the second if-added to refer to the value that triggered the first. Most Lisps use fluid variables. My sentinel system uses lexically bound variables.

If-added procedures in FRL force control to follow a depth first route and to build inappropriate run time stacks if they are used as much as required for a rule-based system. Suppose an if-added triggers and makes an assertion. Immediately the assertion is made, if-addeds that match it also trigger. While new assertions are created, the triggering will continue and will build an unnecessary control stack. Triggering should not require the return of control to the process that created the trigger fact. Because of this, all triggerings in my sentinel system are placed on a queue which is serviced at top level. This effectively decouples the search through the production rule space from the Lisp control structure. Control can then be added at will by the use of explicit control statements, for example through a GOAL frame [De Kleer et al. 1977].

An sentinel which triggers whenever "heavy farm selling" is asserted looks like this.

```
(SENTINEL
   ((NIL    (FARMERS INSTANCE &F))
    (&SELL  (&F SELL &C))
    (NIL    (&C AKO CROP))
    (&BE    (&SELL BE &H))
    (NIL    (&H AKO HEAVY))
   ...then execute the body...)
```

This sentinel executes its body whenever a set of assertions are made that match the triples in the condition. Each triple may be associated with an *annotation variable*. If present, this variable is bound to the annotation frame that represents the triple. In the example, farmers sold a crop and the annotation frame for selling was bound to a variable so it could be

examined further. A sentinel body is arbitrary lisp code although there is a lexical environment associated with it. This allows it to define other sentinels which use the variables no matter when they are executed. In this way, it is easy for rules to define rules.

There are both generic and individual sentinel triggers. The first trigger of the example sentinel is generic because it looks for instances of the generic frame, FARMERS. These may occur anywhere down the inheritance tree from the FARMERS frame. The second trigger is individual. It watches over the SELL slot of a particular farmer discovered by the generic trigger. Individual triggers are fairly simple. A complication with generic triggers arises when we allow new inheritance links to be asserted during running of the system. A new link may complete an inheritance chain allowing a trigger at the top of the chain to fire on a whole set of new instances. Since some of these instances may already have an alternate inheritance path up to the top, the set has to be pruned and only the new instances allowed to fire the trigger.

## Conclusion

Qualitative reasoning is an important mode of reasoning in large domains which cannot be grasped in their entire detail. Information about such domains can often be presented graphically and this assists an expert in reasoning about the data. The process of describing graphs symbollicly is a prerequisite for interfacing graphs and reasoning. I have presented a domain which is ideal for investigating graphic reasoning and have described a method for producing frame descriptions of graphs at multiple levels of detail. These descriptions will eventually form part of a larger system which will apply production rules to derive explanations of graphs from the domain. A sentinel system implements these rules in an extension of FRL.

## References


De Kleer, J., [1978] "*Causal reasoning and rationalization in electroninc*s." MIT AI Lab Memo-499.

De Kleer, J., Doyle, J., Steele, G. L. Jr., & Sussman, G. J., [1977] "*Explicit control of reasoning*." MIT AI Lab Memo-427.

Hendrix, G.G., [1975] "*Expanding the utility of semantic nets through partitioning*." Fourth International Joint Conference on Artificial Intelligence.

Hollerbach, J. M., [1975] "*Hierarchical shape description of objects by selection and modifiaction of prototypes*." MIT AI Lab TR-346.

Marr, D., & Hildreth, E., [1979] "*Theory of edge detection*." MIT AI Lab Memo-518.

Rieger, C., & Grinberg, H., [1976] "*The causal representation and simulation of physical mechanisms*." University of Maryland TR-495.

Roberts, R. B., & Goldstein, I. P., [1977] "*The FRL Manual*." MIT AI Lab Memo-409.

Rosenberg, S. R., [1978] "*Sleuth: an intelligent noticer*." Proceedings of the second Canadian National Conference of the CSCSI.

Stansfield, J. L., [1978] "*A test-bed for developing support systems for information analysis*." Proceedings of the second Canadian National Conference of the CSCSI.

# THEOREM PROVING BY REDUCING CONNECTION GRAPHS

Donald Kuehner
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B9

## ABSTRACT

A connection graph is a representation of an unsatisfiable set of clauses in which all potential resolvents are indicated by links between complementary literals. Kowalski has shown that connection graphs have many advantages in the organization of resolution theorem proving problems. This paper shows that it is possible to predict the complexity of the graph which would be produced when any one of the potential resolvents is activated. The search for a proof is simplified by activating those potential resolvents which reduce the size of the graph. When no further reduction is possible, a minimal expansion may be attempted, or some alternative search procedure may be employed. Using graph reduction and minimal expansion for complete searches for proofs produces exceptionally efficient searches with a test collection of problems.

## 1. INTRODUCTION

In the following, theorems are represented by statements of first order predicate calculus. These statements, in negated form, are written as sets of clauses. Robinson's resolution rule [1] is used to construct refutations for unsatisfiable sets of clauses. Kowalski [2] has developed the connection graph as a means of displaying a resolution theorem proving problem. A connection graph is a representation of a set of clauses in which all potential resolvents are indicated by links between complementary literals as in Figure 1.

In the search for a refutation, a link is activated by replacing it with the resolvent which it indicates. A new graph is formed by deleting the activated link from the old graph, and then linking the literals of the resolvent to the literals of the rest of the graph. In the resolvent, the literals descend from literals in the parent clauses. In a similar way, links to the resolvent descend from links to the parent clauses. In Figure 1, the link labelled X in the top graph is activated to form the middle graph.

Clearly, the clauses of the middle graph, with one resolution forbidden, are unsatisfiable if and only if the clauses of the top graph are unsatisfiable. But in the middle graph there is one literals which does not have a link to it. Robinson's purity principle [1] implies that any

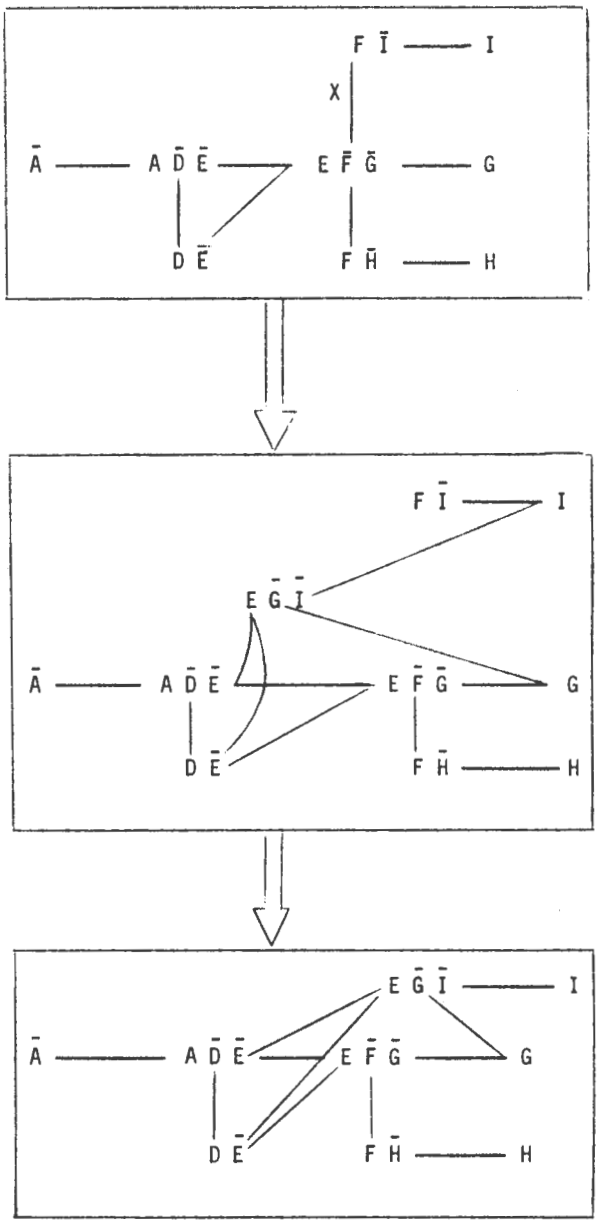



Figure 1.

clause containing an unlinked literal can be deleted from a set of clauses without affecting its unsatisfiability. Deleting this clause and its linkage results in the bottom graph of Figure 1. The deletion of pure clauses will be considered to be part of the activation of a link. Thus the bottom graph is assumed to be deduced directly from the top graph.

The purpose of this paper is to present a method for assigning a precedence for the order of activation of the links in a connection graph. The precedence is assigned so that the complexity of the graph will be reduced or, at worst, increased as little as possible. The graph reduction is bound to simplify the search for a proof. Continuing the search by minimal expansion, although it has given encouraging results, is difficult to justify.

In the top graph of Figure 2, the links of Figure 1 have been assigned precedence numbers. The calculations involved are outlined in Section 3, and are illustrated in Figures 4, 5 and 6. The second graph of Figure 2 is the result of activating the four outer links of precedence +4 and deleting one subsumed clause. Each succeeding graph is obtained by activating the link of highest precedence number. Note that in the fourth graph there is a new kind of link, a merge link between literals of the same clause.

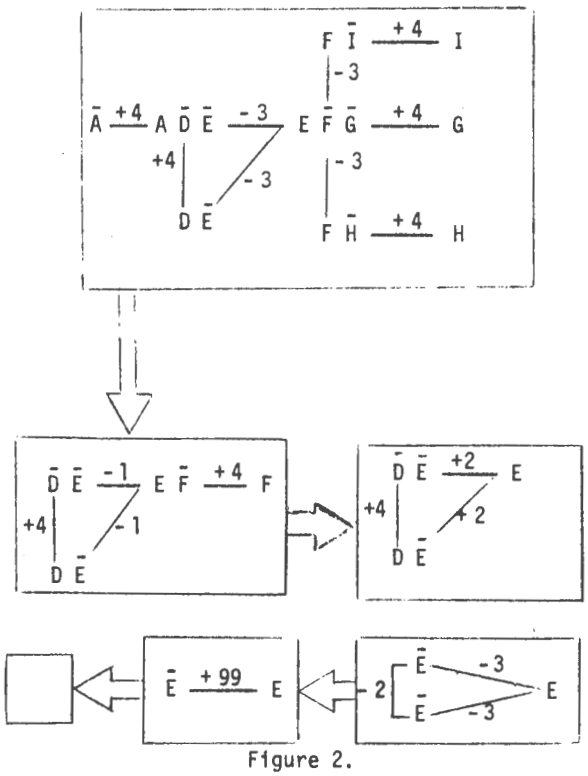


Figure 2.

For simple problems, these precedence rules seem to find short proofs with a small search. In this example, the search size is 17, and the proof size is 14. Using SL resolution [3] or SNL resolution [4], selecting the right-most literal, the search size is 21 and the proof size is 17. These sizes count the number of clauses involved, including the input clauses.

## 2. CONNECTION GRAPHS

A connection graph has literals as nodes. Any pair of nodes may be connected by one of four different kinds of link. The first kind is a clausal link. It connects adjacent literals of a clause. These links are not usually displayed, but are indicated by a close spacing between adjacent literals. The second kind of link is a resolvent link, and indicates a potential resolution. It joins two literals in different clauses which have the same predicate letter, one being negated and the other not, and whose arguments are simultaneously unifiable. The unifying substitution may be thought of as labelling the link.

The third kind of link is a factor link. This is like the resolvent link except that the two literals are in the same clause and have the same sign. Such a link indicates the potential of factoring. The fourth kind of link is a loop. This is like the third kind of link in that it links literals of the same clause, but the literals are of opposite sign. This kind of link indicates the potential of resolution between two copies of the same clause. The unifying substitution must assume that there are distinct variables in the two clauses.

The activation of a resolution link generates a new connection graph. This new graph is constructed from the old one in four steps. First the activated link is deleted. Next, if any clause contains a literal which is not linked to any other literal, then that clause is deleted together with all of the links to its literals. Third, the two clauses, whose literals' link was activated, are resolved together. Finally, the literals of the resolvent are linked in all allowable ways by links to the other literals of the graph. This includes the possible insertion of factor links between literals of the resolvent. To avoid redundancy, these factor links should only be between literals descending from different parents.

It should be noted that except for factors, these new links descend from resolution links or loops in the old graph. If the literal $L'$ in the resolvent descends from the literal $L$ in one of the parent clauses, then $L'$ has a resolution link to a literal $M$ only if there was a link from $L$ to $M$, and then only if $L'$ and $M$ are unifiable.

The activation of a factor link is much the same as the activation of a resolution link, except that there is only one parent, and a

factor rather than a resolvent is generated. Clausal links and loops are not intended to be activated.

## 3. THE GRAPH REDUCTION SEARCH PROCEDURE

It is proposed to activate links in a connection graph in such a way that the graph becomes less complex or increases in complexity by as little as possible. The complexity of a connection graph can be indicated by a triple containing the number of clauses, the number of literals and the number of links. The top graph of Figure 3 has complexity [9, 16, 9].

After activating the left-most link, the resulting graph has complexity [8, 14, 8]. The reduction in complexity can be represented (1,2,1). That is, one clause, two literals and one link have been deleted. The total reduction is represented by the sum of these individual reductions, so the total reduction is 4.

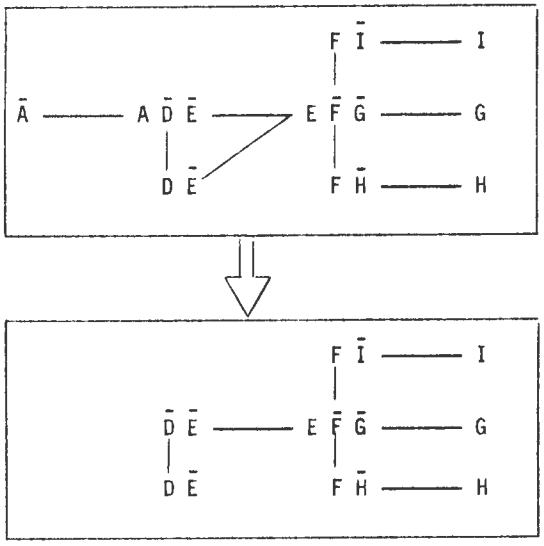



Figure 3.

The generalisation of the preceding example is shown in Figure 4. In the illustrated initial subgraph, the non-unit clause is assumed to have m+1 literals with a total of n+1 links to them. Thus, the whole subgraph has two clauses, m+2 literals and n+1 links, giving it a complexity of [2, m+2, n+1]. Whenever a link can be identified with the initial subgraph, that link is given a reduction estimate of four. This is only an estimate because the unifying substitution may make some other links fail, and factor links may appear when two non-unit clauses are resolved.

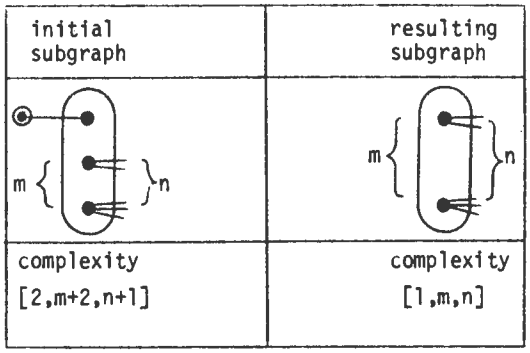



Figure 4.

In order to mechanize the search procedure, there should be a non-pictorial representation of the initial subgraph. In Figure 4, a literal with only one link, in a unit clause, is joined to another literal with only one link in a non-unit clause. This description can be abbreviated to the quadruple [one, unit, one, non-unit] or luln. Of course, lnlu also describes the same configuration. In a connection graph, whenever a link can be described as luln or lnlu , then the link is labelled with the reduction estimate of +4 .

As with all search procedures, the highest priority should be given to a link between two unit clauses. Activating such a link produces the null clause, and the search is completed. Such a situation has link descriptions lulu , lusu , sulu and susu , where the s stands for several links. Since all clauses, literals and links are irrelevant when the null clause is produced, the graph reduction is arbitrarily assigned the number +99 .

In less trivial configurations, one can make a reduction estimate only by counting the number of other literals in the non-unit clause, and the number of links to them. As illustrated in Figure 5, the resulting subgraph contains not only the resolvent, but the non-unit parent. Here the purity principle only applies to the unit parent. Although the unit literal and its link have been deleted, m literals and n links are duplicated, so the reduction estimate is 2-(m+n) .

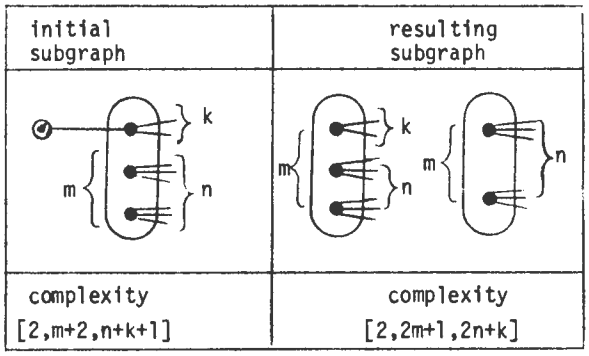



Figure 5.

In this case there is a 1-link unit connected to a several-link literal in a non-unit, so the link description is lusn or snlu . Any link with one of these descriptions has a reduction estimate of 2-(m+n) .

For this initial subgraph, a special reduction estimate could be used when the unifying substitution is empty. Then the resolvent would subsume the non-unit parent. However, to simplify the rules, such special cases are not listed.

Factor links are assigned a precedence in much the same way as resolvent links, except that a link description does not seem appropriate. Their activation is delayed beyond that of most resolution systems.

Finally, loops are arbitrarily assigned a reduction estimate of -99 , although the increase in complexity could be calculated for each case. This effective ban on the activation of loop links is not as restrictive as it may seem. Actually, the activation of these links is only delayed until some other clause resolves with the self-resolving clause. Then the link which descends from the loop is a normal resolution link between the resolvent and the self-resolving clause. This strategy prevents a general axiom from resolving with itself until it is needed to prove a theorem.

The summary of all rules for calculating reduction estimates are displayed in Figure 6. It should be noted that links described by rule 1 should always be activated immediately to terminate the search. Links described by rule 2 always reduce the graph. Those described by rule 3 never reduce the graph, and usually expand it. Links described by rule 4 are never activated.

Theorem proving by graph reduction labels each link of a connection graph with its reduction estimate. Then the link with the highest reduction estimate is activated. If two or more links have the same highest reduction estimate, then some sort of tie-breaking rule is needed. Although it is somewhat foreign to the graph reduction methodology, the most reliable tie-breaking rule seems to be a preference for a link one of whose literals' clauses has the conclusion of the theorem as an ancestor. This is essentially the set of support strategy. Next in preference is a link to a condition of the theorem. If a tie remains after using these tie-breaking rules, then some arbitrary choice must be made.

| rule | link | reduction |
|---|---|---|
| 1 | lulu<br>lusu<br>sulu<br>susu | +99 |
| 2a | luln<br>lnlu<br>lnln | +4 |
| 2b | suln<br>lnsu | +2 |
| 3a | lusn<br>snlu<br>lnsn<br>snln | 2-(m+n) |
| 3b | susn<br>snsu | -(m+n) |
| 3c | factor | -(m+n+k+1) |
| 3d | snsn | -(m1+n1+m2+n2) |
| 4 | loop | -99 |

Figure 6.

## 4. EXAMPLES

There are several advantages to the use of connection graphs. They allow an easy combination of top-down and bottom-up search, or even a middle-outward search. They do not allow the repetition of deductions, and they always allow the unification of complementary units. These advantages are ably pointed out by Kowalski [2].

Any subgraph is easy to reduce if its links have positive reduction estimates. The difficult portion of the graph is characterised by links with numerically large negative reduction estimates. A partly easy, partly difficult problem is the geometry theorem

$$AB \,||\, CD \;\&\; AB{=}CD \rightarrow AC{=}BD$$

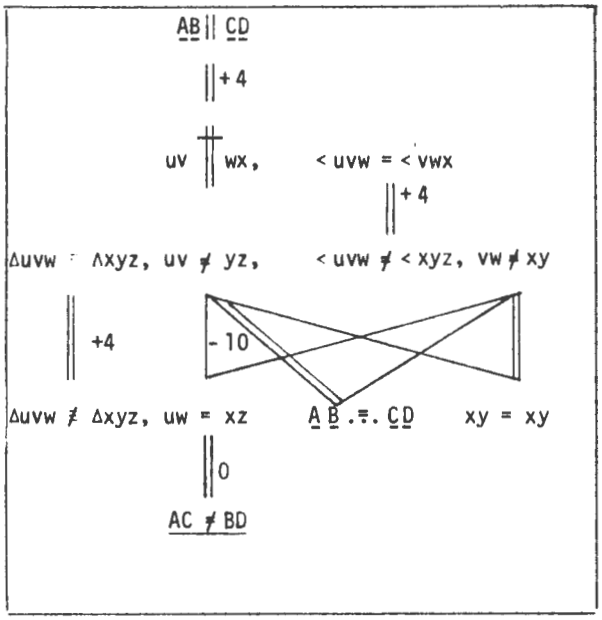



Figure 7.

This can be proved using the following axioms:

uv=yz & <uvw=<xyz & vw=xy → Δuvw≡Δxyz.

uv || wx → <uvw=<vwx.

Δuvw≡Δxyz → uw=xz.

xy=xy.

Figure 7 above illustrates the initial connection graph for this problem. The negation of the theorem produces three clauses. The conditions of the theorem, and later their descendents, are underlined with dashes in the connection graph. The conclusion of the theorem and its descendents have solid underlines. Commas are used to separate the literals of a clause. All predicates are represented as infix relational symbols, with a stroke through the symbol signifying negation. The links are labelled with their reduction estimates. The double links are those which should be activated to form a proof.

The top portion of the graph of Figure 7 can be reduced easily. The graph resulting from the activation of the top two +4 links is the top graph of Figure 8. In all of the graphs of this figure it should be noted that the links needed for the proof have consistently higher reduction estimates. This is an easy search since each activation produces a positive reduction. Thus the problem becomes easier as the search continues. The search is 100% efficient in that only the proof links are activated.

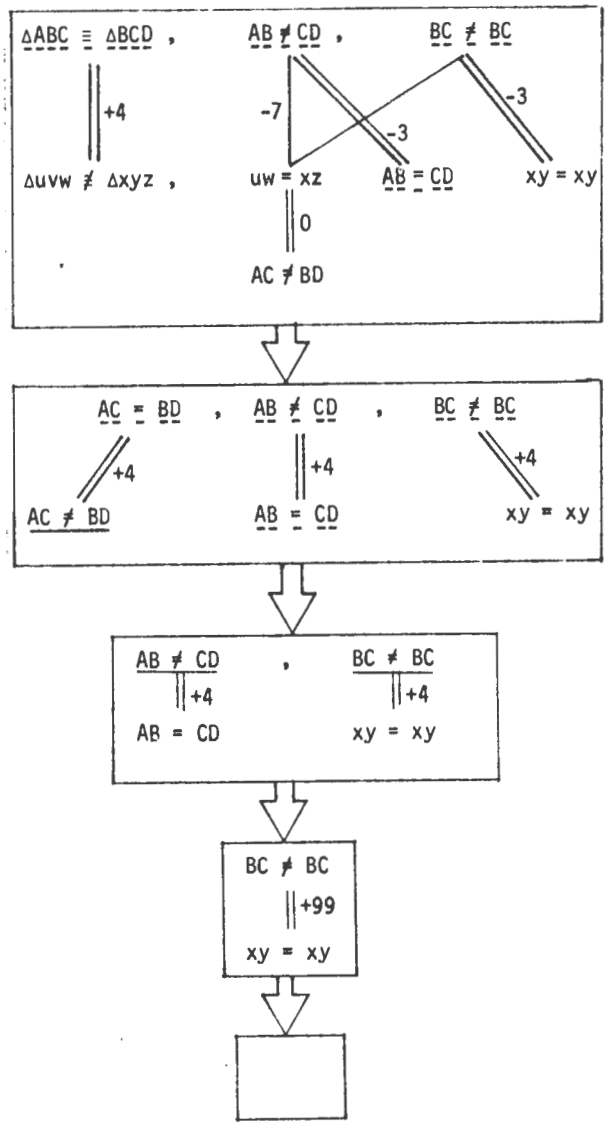



Figure 8.

## 5. CONCLUSIONS

Although it is difficult to form an objective judgement of the usefulness of a deductive system, a comparison of its performance relative to other systems can indicate that it is sometimes useful. Wilson and Minker [5] have solved 152 problems using six resolution inference systems. Of the first thirty, the ten simplest have been solved by connection graph reduction. The testing of the remaining problems will have to wait for the completion of the computer implementation of the procedure. The current results appear in the following table.

The inference system abbreviations stand for unrestricted resolution, linear resolution, set of support, P1 , SL, linear plus set of support, and connection graphs. The first six use diagonal search, while the last uses graph reduction. The problem names and the statistics for the first six columns are those of Wilson and Minker. For each problem, the top line of statistics is the proof size, the second line is the search size and the third line is their ratio, the efficiency of the search. A question mark for proof size indicates that no proof was found. In calculating the proof and search sizes, all resolvents and input clauses were counted. The line labelled "average" is the average of the efficiencies for each inference system. The average of 90% for connection graph reduction seems significantly better than the next best average of 65%.

| | INFERENCE SYSTEM | | | | | | |
|---|---|---|---|---|---|---|---|
| Problem Name | UNR | LIN | SOS | P1 | SL | L+S | CG |
| ANCES | 18 | ? | 19 | 19 | 18 | 18 | 13 |
| | 47 | 2668 | 45 | 251 | 84 | 337 | 13 |
| | .38 | .00 | .42 | .07 | .21 | .05 | 1.00 |
| DM | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| | 9 | 9 | 9 | 9 | 12 | 9 | 8 |
| | .89 | .89 | .89 | .89 | .67 | .89 | 1.00 |
| EW1 | 14 | 13 | 13 | 13 | 13 | 13 | 11 |
| | 15 | 41 | 13 | 15 | 22 | 23 | 11 |
| | .93 | .32 | 1.00 | .81 | .59 | .57 | 1.00 |
| EW2 | 12 | 11 | 12 | 11 | 11 | 12 | 11 |
| | 12 | 118 | 14 | 11 | 22 | 23 | 11 |
| | 1.00 | .09 | .86 | 1.00 | .50 | .52 | 1.00 |
| EW3 | 21 | ? | 21 | 19 | 18 | ? | 13 |
| | 360 | 852 | 360 | 40 | 58 | 800 | 16 |
| | .06 | .00 | .06 | .48 | .31 | .00 | .81 |
| MQW | 8 | 9 | 8 | 10 | 11 | 9 | 8 |
| | 16 | 68 | 11 | 24 | 65 | 55 | 12 |
| | .50 | .13 | .73 | .42 | .17 | .16 | .67 |
| NUM1 | 12 | 12 | 12 | 12 | 12 | 12 | 11 |
| | 19 | 34 | 17 | 16 | 32 | 30 | 17 |
| | .63 | .35 | .71 | .75 | .38 | .40 | .65 |
| QW | 9 | ? | 9 | 9 | ? | ? | 10 |
| | 19 | 1298 | 19 | 30 | 1348 | 1720 | 11 |
| | 47 | .00 | .47 | .30 | .00 | .00 | .91 |
| ROB1 | 9 | 9 | 11 | 9 | 11 | 11 | 9 |
| | 10 | 13 | 41 | 10 | 11 | 41 | 9 |
| | .90 | .69 | .27 | .90 | 1.00 | .27 | 1.00 |
| LS5 | 9 | 9 | 10 | 10 | 9 | 9 | 7 |
| | 13 | 25 | 11 | 12 | 13 | 17 | 7 |
| | .69 | .36 | .91 | .83 | .69 | .53 | 1.00 |
| AVERAGE | .65 | .28 | .63 | .65 | .45 | .34 | .90 |

## REFERENCES


1. Robinson, J.A., A machine-oriented logic based on the resolution principle. J. ACM 12, (Jan. 1965), 23-41.

2. Kowalski, R.A., Proof procedure using connection graphs, J. ACM 22, 4 (Oct. 1975), 572-595.

3. Kowalski, R., and Kuehner, D., Linear resolution with selection function, Artificial Intelligence 2, (1971) 227-260.

4. Kuehner, D., Some special purpose resolution systems, In Machine Intelligence 7, B. Meltzer and D. Michie, Eds., Edinburgh U. Press, Edinburgh, Scotland, 1972, pp. 127-128.

5. Wilson, G.A., and Minker, J., Resolution, refinements and search strategies - a comparative study, IEEE Transactions on Computers C25, 8, (Aug. 1976).

# TOWARDS AN ITERATIVE APPROACH TO PROGRAM SYNTHESIS

Michael A. Bauer

Department of Computer Science
University of Western Ontario
London, Ontario N6A 5B9

## Abstract

An approach to program synthesis based upon modifying an existing procedure is described. The particular synthesis problem studied involves synthesizing a procedure from example computations. The paper focuses on the kinds of errors introduced during synthesis, the way errors can be detected by comparing a procedure to examples and describes several rules for modifying a procedure in order to correct certain kinds of errors.

## 1. Introduction

Program synthesis, in a general sense, involves the formation of a program from descriptions of an algorithm or its properties. A common paradigm within this approach entails a user presenting one or more descriptions to a synthesizer which, in turn, constructs a procedure. Then, should the user present additional descriptions, the new descriptions as well as previous ones, are used to form a new procedure. The initial procedure (or most recent) is discarded.

In this paper an approach to program synthesis is described which attempts to take advantage of the most recent procedure - essentially attempting to alter it when new descriptions are presented. This same paradigm was utilized by Sussman [4] in developing a program which improved its performance by expanding procedures already existing to solve more complex problems. Such an approach may be more difficult, but may suggest approaches to more powerful synthesis systems or may provide insight into more fundamental problems of how systems can cope with errors, how errors can be analyzed and utilized.

The particular synthesis problem investigated here involved the formation of a procedure from examples of its computation (see [1,2]). Examples of computations are basically traces of executions on specific values. In the approach adopted here, the user presents a number of examples to an initial synthesizer which, in turn, produces an initial program and, possibly, some additional information. This information may simply be the original examples or may be a summary of assumptions made during the synthesis process. Determining what kind of information to keep will depend on how successful (or unsuccessful) the synthesis process is when given a program and additional examples. In the work described in this paper, no additional information was kept.

The user may then, or at some later point, present additional examples of the same procedure. Now, the synthesizer, given new examples, attempts to construct a new program. The first step of this process is to compare the existing version of the program with the examples. The purpose of the comparison process is to detect inadequacies within the program, i.e., discrepancies between the program and the examples. These discrepancies constitute "errors" in the program. Should no "errors" be discovered, then the examples, in essence, provide no new information and the program is left unaltered.

However, assuming that some discrepancies do exist, the next step involves determining the incorrect portions of the program. These portions must be removed and, using the examples, corrections made and a new program produced. The removal and insertion of code must be done (1) so that the resulting procedure can be compared and modified, if necessary, with subsequent examples and (2) in such a way that the code corresponding to previously presented examples remains correct. Once again, it is conceivable that certain "comments" about the modification could be produced. In the work reported here, this was not investigated.

## 2. An overview of the initial synthesis process

Programs are represented as rooted, labelled digraphs in which the successors of each node are ordered (in figures, counter-clockwise). Examples, basically sequences of instructions executed, are represented as trees. A program has an associated list of formal parameters and an example has an associated list of inputs (actual parameters). An example may include specific uses of the inputs, say as replacements for variables or included in temporary assignments to variables. Variables from example to example need not be the same. Figure 1 contains an example of a procedure and Figure 2 presents two examples which could be used to describe that procedure. A more complete description of procedures and examples can be found in Bauer [1,2].

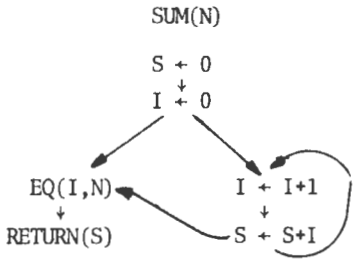



Fig. 1: A procedure to sum the first N integers.

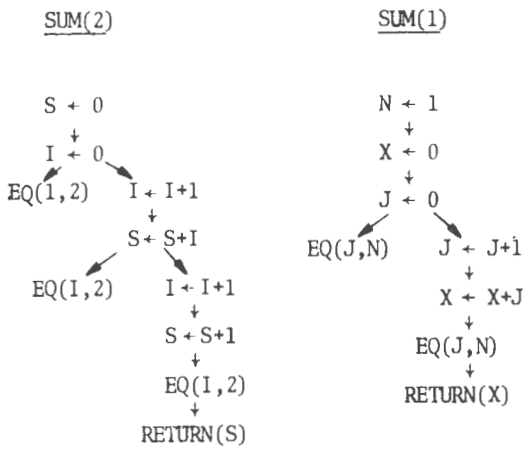



Figure 2. Two examples.

Given a number of examples, as trees, the initial synthesis process must (1) determine a graph whose nodes are labelled by instructions and (2) determine a parameter list. The formation of a graph is accomplished by a grouping process which attempts to group nodes from the examples. Each group represents a node of the synthesized procedure and is valid if it satisfies a number of constraints. First, each instruction within a group must be an instance of a single general instruction (similar to the least generalization of a Predicate Calculus formula, see Plotkin [3]). Second, connectivity between groups is determined by the connectivity between individual nodes of the examples. If nodes $v_1$ and $v_2$ are placed in the same group, then each pair of their corresponding successors (i.e., the first successor of each, the second of each, etc.) must also be in a single group. In addition, there are two remaining types of constraints - one on possible variable renamings and the other on replacement and use of actual parameters.

The result of the grouping process is a procedure body, i.e., a rooted digraph in which each node is labelled by an instruction. Given the procedure body and the information computed during the grouping process, the next step is to form a parameter list. This process also relies on a number of constraints based upon (1) the actual arguments within the examples, (2) any renaming of variables and (3) any deleted instructions (essentially added assignment statements which assign an input to a variable). Should this phase of the synthesis process fail, the grouping process is resumed and the search for a new procedure body continued. If a parameter list is formed, then the construction is completed and the resulting procedure is guaranteed to be correct with respect to the given examples (Bauer [2]).

An overview of the initial synthesis algorithm follows:

1. Input the examples.
2. Find a procedure body.
   (a) Form an acceptable grouping of nodes not yet tried. The grouping must satisfy connectivity and instructional constraints. If no acceptable grouping can be found - halt with Failure.
   (b) Construct a procedure body P.
   (c) If P is consistent with the information in the examples, then continue with step 3; otherwise reject P and go to step 2a.
3. Find an acceptable parameter list, $(X_1,\ldots,X_n)$ - one consistent with the variables of the examples. If none can be constructed, then reject P and go to step 2.
4. Find parameters to replace any constants in P, if possible.
5. Halt with procedure $P(X_1,\ldots,X_n)$.

## 3. Errors: Sources and Detection

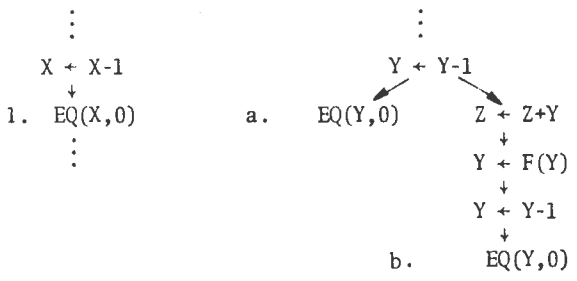
Given examples of a synthesized procedure, the first task must be to compare the examples and the existing version of the program. This comparison should identify portions of the procedure which conflict with one or more of the examples. Once this is done, the task of correction can proceed.

In the existing environment, "errors" introduced into a program originated with its synthesis or modification. Moreover, "errors" could only be introduced (barring implementation problems, bad data, etc.) at points in the synthesis process where one of a number of alternatives was selected, i.e., choice points. Whenever a choice is made, a number of reasonable alternatives may have been available. If the number of alternatives could always be constrained to one, then, of course, one no longer has a choice, but an algorithm for computing the appropriate result. In synthesis, one typically has partial information and may, therefore, have a number of choices. This provides a general rule: "Each class of choices gives rise to a class of potential errors".

In the synthesis algorithm described at the end of the previous section, one can identify three classes of choices:

1. Group membership - Although there are constraints dealing with the connectivity of nodes and similarity of instructions, it may be possible that a node could be put into two or more groups.

Consider the following portions from two examples:




Assume that nodes 1 and a can be grouped together (this implies that X and Y are renamings of the same variable). Node b might or might not be grouped with a - both are plausible.

2. Selecting Parameters - Given a number of variables within a procedure body and even though there are constraints on the variables in the constructed procedure, it may be possible to put some variable in one or more parameter positions.

Suppose that a procedure was to have two formal parameters and that the argument lists of the two examples being used to synthesize it were (3,3) and (4,4). After synthesizing a procedure body, assume that X and Y were potential parameters. Based upon the information available, the parameter list (X,Y) is as likely as (Y,X) .

3. Replacing Constants - A constant appearing within an instruction of the procedure might be replaced by two or more parameters. Obviously if it is replaced, one must choose among the potential candidates. Suppose a node in the above synthesized procedure body was Z ← Z+3. In this case, the 3 could actually be a constant, or it could be X or Y . Without more information, choosing one of X or Y could very likely be a wrong choice. Note that if one left 3 as a constant, then in some situations that too would be an error.

Having identified these classes of choices, one can examine the consequences of the choices on the resulting procedure. This will provide a general classification of the possible errors within a synthesized procedure.

Suppose that an error in group membership has occurred. This means that (1) the connectivity of nodes within the procedure is incorrect and/or (2) two variables identified as the renaming of another are actually renamings of distinct variables. The first of these is a structural error and the second is a renaming error. If a variable involved in the renaming error is a parameter which has not been assigned to, then one has a parameter error.

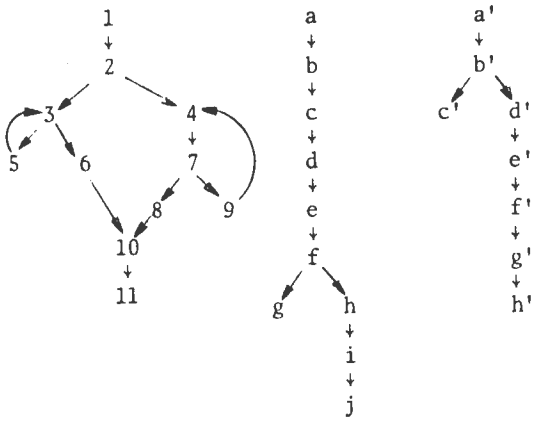
In the case of an inappropriate choice of parameters, one may have, as above, both parameter errors and renaming errors.

Finally, in the case where one has replaced a constant by an incorrect variable or if the constant should not have been replaced, then one has another parameter error.

Two important problems with which one must deal are (1) a single choice may yield a number of possible errors and (2) one class of errors is not necessarily associated with a single class of choices.

The comparison process begins by finding a common starting node in the procedure and each of the examples. This is not necessarily the first node in the procedure or in the examples. Rather, it is based upon the first node in each having a predicate or function (see [2] for a more complete definition). Once this set of start nodes has been formed, nodes of the procedure are grouped with nodes from the examples. Since the successors of a node are ordered, the comparison process simply follows the ordering. Each set of grouped nodes is called a match.

Suppose that in the following procedure and examples that the set of start nodes was {2,c,b'}.




Then the next matches would be {3,d,c'} and {4,d'}. Beyond this, the nodes of each example would be matched with nodes of the procedure independently, since each represents a different computation path.

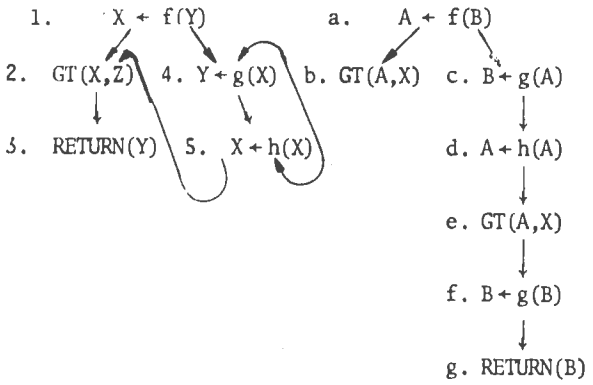
The resulting matches would be:

{2,c,b'}, {3,d,c'},{ 4,d'}, {5,e}, {7,e'}, {3,f}, {8,f'}, {5,g},{ 10,g'},{6,h}, {11,h'}, {10,i}, {11,j}.

The detection of errors is based upon an examination of nodes in a match. Nodes within a match must satisfy local and global constraints. The violation of a constraint signifies an error.

### 3.1 Structural Errors

Consider the following portions of a procedure and an example:




Assume that {1,a} is a match. Any match must satisfy two local constraints - they must be structurally similar and corresponding variables and constants must be consistent. This is determined by a computation similar to that of computing the least generalization of a predicate calculus formula (see [1,2] for a precise description). Intuitively, this means that the instructions must involve the same functions, predicates and constants and that the variables involved must be used in similar ways. Given {1,a} the instructions are identical except for variable renamings and there is a unique renaming: $X \leftrightarrow A$, $Y \leftrightarrow B$ . Once established, the pairing of a variable in an example to one in a procedure must be preserved throughout the matching process. This forms one of the global constraints (see Section 3.2).

The next matches are {2,b} and { 4,c}. In {2,b} the pairing $X \leftrightarrow A$ is still intact and the pairing $Z \leftrightarrow X$ is formed. From {4,c}, the previous pairings are preserved.

The next match is {5,d}. Note that the set {3}, which follows from {2,b} is ignored, since it only contains nodes of the procedure. After {5,d}, the next match is {2,e}. Hence node 2 of the procedure occurs in an additional match. This will happen if the procedure contains loops.

The set {3,f} forms the next match. The instructions are no longer similar. The match {3,f} has revealed a structural error.

This situation might arise when a node within an example (in this case one of the form GT(V,W) ) should not have been grouped with a similar node, but it was because of lack of examples, in this case a subsequent computation involving the function g .

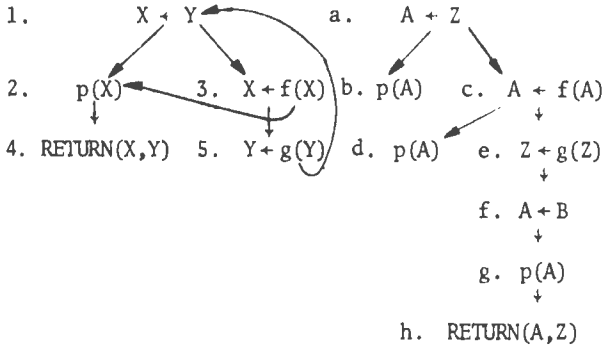
Other structural errors can be detected in matching nodes even if the instructions are similar. Consider {X ← f(Y,Z), A ← f(B,B)} . In forming the pairings, one has $X \leftrightarrow A$, $Y \leftrightarrow B$ and $Z \leftrightarrow B$ . Since Y and Z are interpreted as distinct variables, Y and Z cannot both be paired to B - hence no match is possible.

Similarly, consider {X ← f(3), Y ← f(6)}. Here the 3 is a constant in the procedure and 6 is a constant in the example (i.e., does not appear in the argument list). Once again, no match is possible and an error is detected.

### 3.2 Renaming and Parameter Errors

Assuming nodes in a match are locally consistent, i.e., do not reveal a structural error, the variable pairings are examined with respect to previous variable pairings.

Given a variable within the procedure, it may be paired with at most one variable in any example and with at most one input value in any example. For example, consider the following:

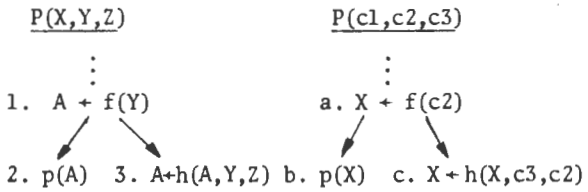



Assume that {1,a} is a match. The variable pairings are $X \leftrightarrow A$ and $Y \leftrightarrow Z$ . Matches and pairings are formed as follows:

| | |
|---|---|
| {2,b} | $X \leftrightarrow A$ |
| {3,c} | $X \leftrightarrow A$ |
| {2,d} | $X \leftrightarrow A$ |
| {5,e} | $Y \leftrightarrow Z$ |
| {1,f} | $X \leftrightarrow A$ , $A \leftrightarrow B$ |

Since Y had been previously paired with Z , the matched set {1,f} reveals an error - a renaming error.

As an example of a parameter error, consider the following portions of a procedure and example:




Assume that {1,a} is a match. Then {3,c} is a match with pairings $A \leftrightarrow X$, $Y \leftrightarrow c3$ and $Z \leftrightarrow c2$. Assuming that Y has not been assigned to on the path from the root of P to node 3, if $c2 \neq c3$, then Y, which is the second parameter of the procedure, has been paired with an input other than the second - a parameter error.

As in the case of structural errors, it is easy to construct examples in which the synthesized procedure actually contains such errors.

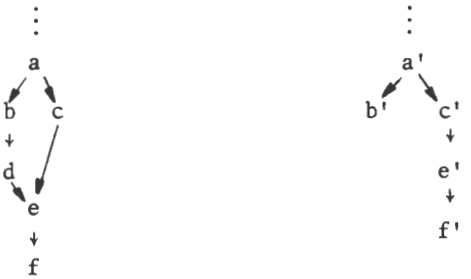
## 4. Eliminating Errors

When an error has been discovered, the comparison process is interrupted and an attempt is made to correct the error. Once corrected, the comparison process is restarted with the modified procedure and the examples. This permits multiple errors to be detected and corrected. As noted, the detection of an error in a particular match does not determine uniquely the choice in the construction which produced that error. Hence the elimination of an error requires (1) isolating of one or more instructions so that changes can be made without affecting correct computations (if possible), and (2) deleting erroneous instructions and inserting correct ones.

In the approach adopted here, the isolation of nodes once an error is discovered is relatively independent of the particular error. The correction, of course, will depend upon the kind of error.

### 4.1 Isolating Instructions

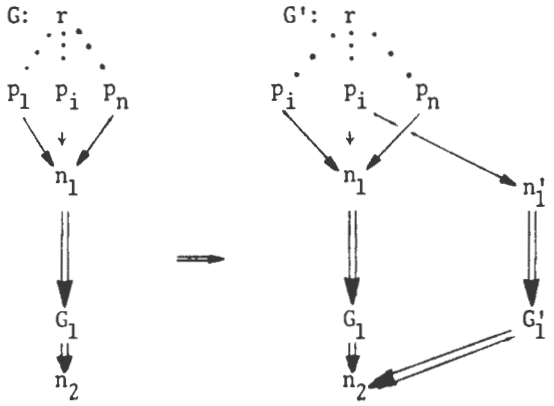
Consider the following portions of a procedure and example:




Assume that the matches are {a,a'}, {b,b'}, {c,c'}, {e,e'}, {f,f'} and that the set {f,f'} has revealed an error. Obviously, one would not wish to simply remove f (and its successors) from the procedure since they may belong to correct computations from node b. To avoid making changes which introduce errors, one must isolate the computation path in the procedure which was matched to nodes of the example.

This isolation is accomplished by three graph transformations which can be applied to subgraphs of an existing graph.

1. Transformation SPLIT(G,$p_i$,$n_1$,$n_2$) ⟹ G':

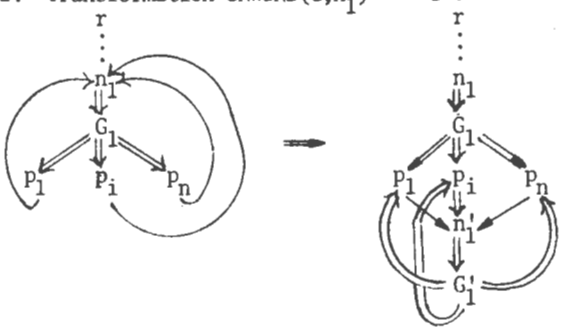



where:

1. r is the root of G, $p_1,\ldots,p_n$ are predecessors of $n_1$
2. there are at least two distinct paths from r to $n_1$ in which $n_1$ occurs once in each and $p_i$ is a predecessor of $n_1$ in one of these paths.
3. $G_1$ is the smallest subgraph of G such that if n is reachable from $n_1$, $n_2$ is reachable from n, (n,m) is an edge of the path from n to $n_2$ and $m \neq n_2$, then m and the edge (n,m) are in $G_1$.
4. $n_1'$ is a copy of $n_1$
5. $G_1'$ is a copy of $G_1$, where if $m \in G_1$ then $m' \in G_1'$, m' a copy of m and for any $m \in G_1'$, $n \notin G_1'$ such that (m,n) is an edge of G, then (m,n) is an edge of G'.

Intuitively, SPLIT duplicates a portion of a procedure between two instructions, $n_1$ and $n_2$ through one particular predecessor of $n_1$. In use, $n_2$ will be the node of the procedure which has occurred in an error, $n_1$ and $p_i$ would be nodes of the procedure which have been matched prior to the error.

2. Transformation UNWIND(G,$n_1$) ⟹ G':




where

1. r is the root of G, $p_1,\ldots,p_n$ are the predecessors of $n_1$ which can only be reached on paths from r which include an occurrence of $n_1$ prior to its predecessor

2. $G_1$ is the subgraph of nodes and edges reachable from $n_1$, excluding $p_1,\ldots,p_n$

3. $G_1'$ is a copy of $G_1$, $n_1'$ is a copy of $n_1$.

UNWIND provides a way to isolate portions of a graph by unwinding loops. One can show that both transformations preserve computational equivalence of G and G'. The combined effect of using SPLIT and UNWIND on nodes along a particular path is that an erroneous node can be isolated and changed in the transformed procedure without affecting other computations.

3. Transformation COPY(G,$n_1$) ⟹ G':


where

1. r is the root of G

2. $G_1$ is the subgraph of nodes reachable from $n_1$

3. $G_1'$ is a copy of all nodes and edges of $G_1$.

This final transformation is used to create a copy of the subgraph reachable from a particular node. Once again, this can be used to duplicate portions of a graph in order to isolate changes.

Given a node which is involved in an error, there is a path from the root to that node corresponding to the path in an example. One can, by using these transformations, form a new procedure, equivalent to the original in which there is a unique path from the root to the predecessor of the erroneous node. In this case say that the erroneous node has been isolated.

4.2 On Error Corrections

The previous transformations can be used to isolate a portion of the procedure which can then be modified. Corrections take the form of code deletion or changes within instructions. Insertion of new code is accomplished by an algorithm similar to the synthesis process. Hence, a modified procedure can itself be altered since identical classes of errors arise.

A formulation of all error corrections has not yet been completed. However, the following examples will illustrate the notions involved in the correction process.

4.2.1 Structural Errors

Assume that a match has revealed a structural error in a procedure P. Let n be the node of P in that set and let $n_1$ be the predecessor of n along the matched path. Assume that n is the k-th successor of $n_1$. This provides the basis for the following:

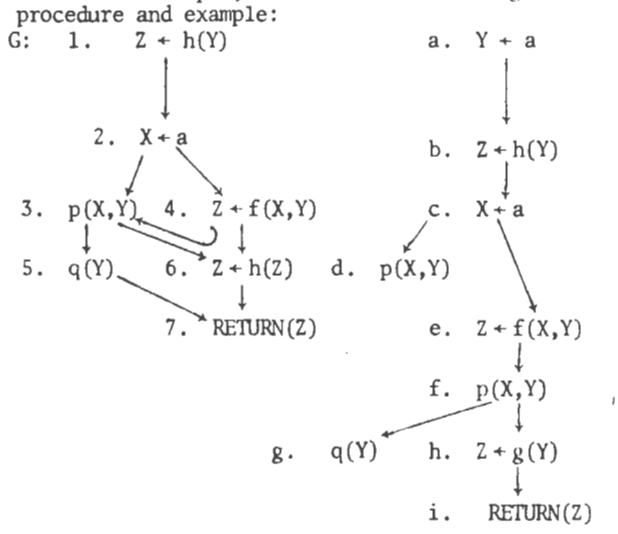
The Correction Rule for Structural Errors:

1. Transform P into P' such that n is isolated in P'.

2. If $n_1$ has m successors, delete edges from $n_1$ to successors k through m and any nodes and edges no longer reachable from the root of P'.

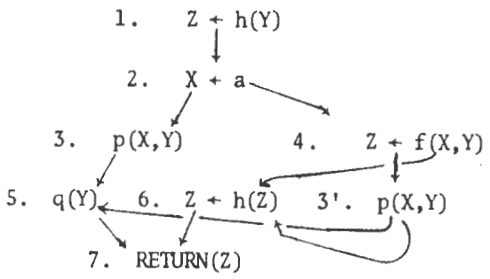
Note that only code is deleted. New code is inserted by an algorithm similar to the initial synthesis algorithm described earlier.

As an example, consider the following procedure and example:



where the following matches have occurred: {1,b}, {2,c}, {3,d}, {4,e}, {3,f}, {5,g}, {6,h}. Since the set {6,h} reveals a structural error (they involve different functions), the above correction rule is invoked.

The first step is to isolate node 6 based on the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 6$ . To do this, the procedure is transformed using SPLIT(G,4,3,6) resulting in:




Now the set {3,d} has become {3',d} and there is a unique path from the root to node 3! Then the edge from 3' to 6 is deleted. In the final phase, nodes h and i of the example would be incorporated into the procedure producing the correct change.

4.2.2 Renaming Errors

When a renaming error is discovered, the following information is available: (1) the node of the procedure, say, n , in which the error occurred, (2) the incorrect pairing, say $X \leftrightarrow B$ , (3) the pairing which had previously existed, say $X \leftrightarrow A$ and (d) perhaps a pairing $Y \leftrightarrow B$ . The correction rule is:

Correction rule for Renaming Errors:

1. Form P' = COPY(P,n) .
2. In the copies subgraph, replace all occurrences of X by W , where W is Y if a pairing $Y \leftrightarrow B$ existed, otherwise W is a new variable not occurring in P.

The copy operation arises from the fact that if a renaming error occurred then the nodes used in the group corresponding to that node when the procedure was constructed would have had to have nearly identical computation paths from them. In fact, they would be identical except for variable renamings - hence the necessity of copying.

4.2.3 Parameter Errors

A parameter error, though similar to a renaming error, involves a different kind of correction. The information available is (1) the node n of the procedure in which the error was detected and (2) the incorrect pairing $X \leftrightarrow t$ , where t is either a variable A or an input argument c . The correction rule is:

Correction rule for Parameter Errors:

1. If a unique parameter Y of P can be paired with the input c or A , then replace X in n by Y .

Unlike the previous correction rules, this one is conditional. This is the case since an arbitrary replacement may replace one erroneous parameter by another. Hence, it is possible that, although the error is detected, it cannot be corrected without additional input. It seems that in this particular instance the pairings of parameters to inputs from previously used examples would be useful. Nevertheless, the uniqueness of Y can often be determined by using the information within the previous pairings and from the usage of inputs within all the examples. In practice, these errors seem to occur infrequently.

## 5. Summary and Conclusion

Based upon experience with the rules and the synthesis process, structural and renaming errors appear to be the most common. Moreover, the described correction rules seem to cover most of the "common" cases. This is certainly the case for structural errors. In the case of both renaming and parameter errors, the rules are incomplete.

Parameter errors are interesting in that the correction rules seem to be of a conditional form (if one wishes to avoid replacing one error by another). In the case of structural and renaming errors, if the error is detected then enough information exists to make the correction. It appears that to guarantee this for parameter errors, some additional information must be kept, e.g., previous inputs. Beyond this, there seems to be little need to keep all previous examples.

Multiple errors are handled by correcting one error at a time, when discovered, and restarting the comparison process. This can lead to a proliferation of duplicate instructions. A more realistic, but apparently more difficult approach, would involve collecting information about a number of errors and making a minimal number of changes.

As alluded to, the final phase of the synthesis process involves a process similar to the initial synthesis algorithm. By constraining this algorithm, a modified procedure can be altered by examples and yet be guaranteed to work on the input of previous examples - this forms the basis for the iterative paradigm. Without these constraints, one may face new classes of errors arising from the introduction of errors into already incorrect code.

## References

1. Bauer, M., "Programming by Examples", Artificial Intelligence, Vol. 12, No. 1, 1979.

2. Bauer, M., "A Basis for the Acquisition of Procedures", Ph.D. Thesis, University of Toronto, Dept. of Computer Science, 1978.

3. Plotkin, G., "A Note on Inductive Generalization", Machine Intelligence 5, editor D. Michie, University of Edinburgh Press, 1969.

4. Sussman, G., "A Computational Model of Skill Acquisition", AI TR-297, MIT Artificial Intelligence Lab., 1973.

# HANDLING EXCEPTIONAL CONDITIONS IN PSN

Yves Lesperance

Department of Computer Science
University of Toronto
Toronto, Canada
M5S 1A7

## ABSTRACT

This paper describes a scheme for handling both exceptional objects and classes and exceptional conditions that arise in the execution of programs, within a knowledge representation formalism. The scheme consists of two mechanisms: the excuse, which allows the justification of specified constraint violations in instances of a class through membership in a second class within designated contexts, and the mapping, which permits the specification of similarity relationships between the definitions of two objects, so that arbitrary elements of these definitions may be copied or inherited (a flexible IS-A). Exceptions in programs are handled through an extension of the excuse mechanism.

## 1.0 INTRODUCTION

In order to perform intelligently , a system must possess a model of its world and be able to use it to deal with the often unexpected situations that arise. The knowledge in this model (knowledge base) is organised in terms of a system of categories. The cathegories may be explicit, as in frame systems [Minsky 74], or more implicit as in logical formalisms. Exceptions in representation systems arise as a result of (1) the sometimes unpredictable nature of the world, which produces atypical situations, and (2) the inadequacies of current representation formalisms in dealing with "natural" concepts (as used by people). These exceptions manifest themselves through the violation of some constraint during the lifetime of the knowledge base.

A simple classification of exceptional conditions will help in finding ways to deal with them. Generic exceptions can first be distinguished fron individual exceptions, as the former pertains to constraints violated in the definition of a category rather than in particular individual objects. Individual exceptions can be further subdivided into static exceptions, which arise while the systems is attempting to instantiate or recognize an object (basic operations at the top-level), and dynamic exceptions, which are encountered during the execution of a user defined program.

This paper sumarizes an exception handling system develloped for the PSN representation formalism [Levesque 79], which is explained in details in [Lesperance 80]. The seminal ideas for the system came from [Minsky 74], where two ways of recovering from failure in a frame system are suggested. First, it may try to create an excuse for the exceptional condition with an appropriate reason. In this approach, the failure is seen as arising from the fact that the defective object is really an instance of two frames which interact, thus the object does not satisfy perfectly the ideal defined in one of the frames. The knowledge necessary to make the repair should be attached to a higher thematic context frame. The second approach involves using the local advice embedded in a similarity network to replace the defective frame by a more appropriate one.

The two approaches reflect the distinction between individual and generic exceptions. In the first case, we do not wish to create new categories for every single exception, thus an excuse mechanism has been devised to allow the handling of both static and dynamic exceptions and the maintenance of the consistency of the knowledge base. The excuse mechanism has been influenced extensively by exception handling mechanisms develloped for programming languages, [Levin 77] in particular. These mechanisms allow the mainline of the program to be expressed without cluttering it with the code required to handle exceptional conditions. Moreover, the handling code for the condition is attached to the caller or user of the program module which raised the exception, allowing for a context dependent recovery from the exception. This facility permits the use of a procedure even if the conditions for which it was designed are not satisfied, as long as the exceptions that will be raised can be handled by its caller or user. For generic exceptions, the problem lies in the insertion of the category into the existing hierarchies, especialy when the inheritance of only part of the definition of the category is desired. This has been done through a mapping mechanism inspired from [Moore 73], which makes explicit the inheritance process of definition elements and gives control to the user over it when this is needed.

The developpement of this system is seen as a step in the direction of improved flexibility for

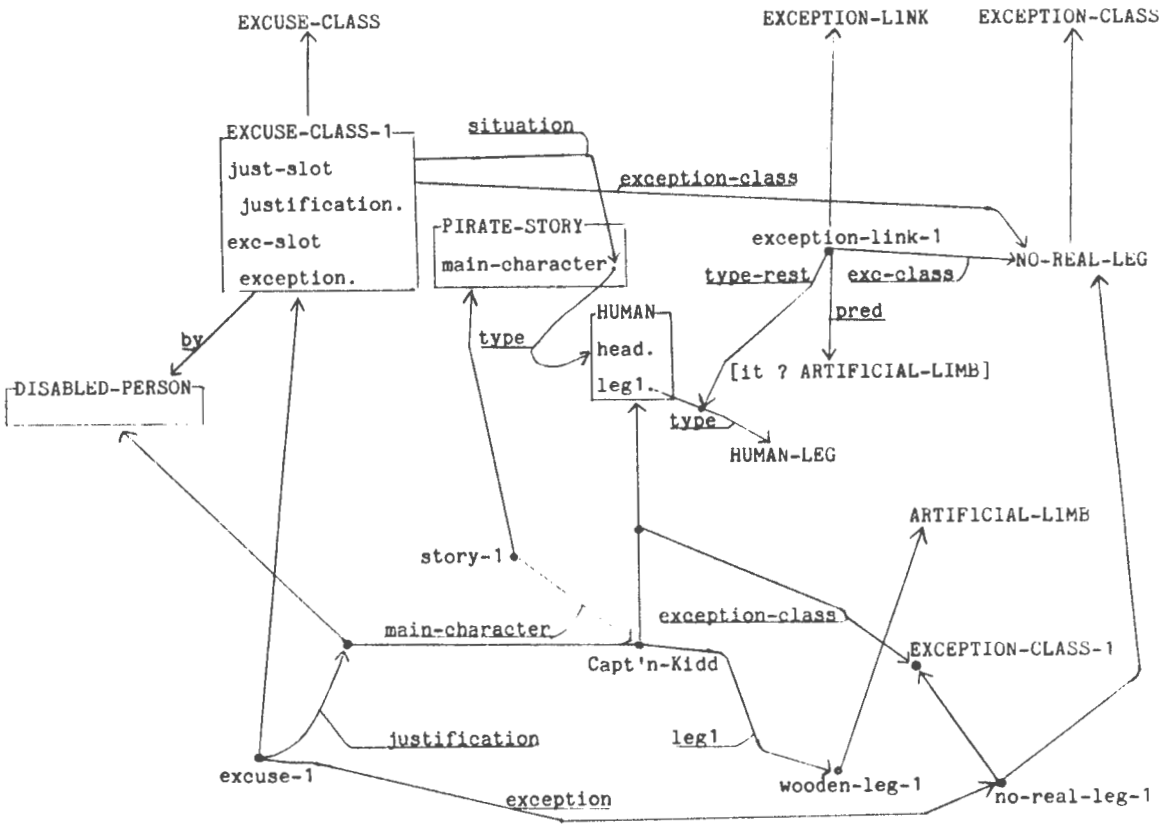



Figure 1 - Example of excuse for static exception.

representation formalisms, both for practical purposes and modelling adequacy. The system can be readily adapted to most other semantic network or frame based formalisms. The approach taken emphasizes the knowledge base definition aspect, but generality has been preserved. Before the system can be explained, an overview of its host formalism must be given.

## 2.0 OVERVIEW OF PSN

The PSN formalism grew out of a desire to develop a facility for defining semantic network knowledge bases with well defined semantics. The formalism is basicaly procedural, as the semantics of classes, which represent generic objects, are defined in terms of four attached programs, which prescribe the behavior of the class under the operations of instantiation, removal of an instance, testing for membership and fetching of all instances. Classes are represented graphicaly by their external name in capitals, for example "HUMAN" or "EXCEPTION-CLASS" in figure 1. Whenever an individual object is made an instance of a class, the appropriate attached program is executed, this allowing the desired inferences (antecedent theorems) to be added to the knowledge base. Similar action is taken in the case of the three other operations. Simple token objects are represented in the graphic notation by their external name in lower case, for example "Capt'n-Kidd" in figure 1. The INSTANCE assertion is represented by an unlabled single line arrow. Incidental relationships between objects (the links in traditional semantic networks) are represented by a class of objects called relations, whose semantics are also defined by four programs. The instances of relations are assertions of the relationship between two specific objects.

This basic procedural PSN is augmented with declarative facilities which help in the organization of the knowledge base. The defining properties of a class are grouped together to form the structure of the class, which consists of a set of slots which can have a type, restrictions, default, etc.. The structure of a class is represented by a box under the name of the class, for example "HUMAN" in figure 1, and slots by

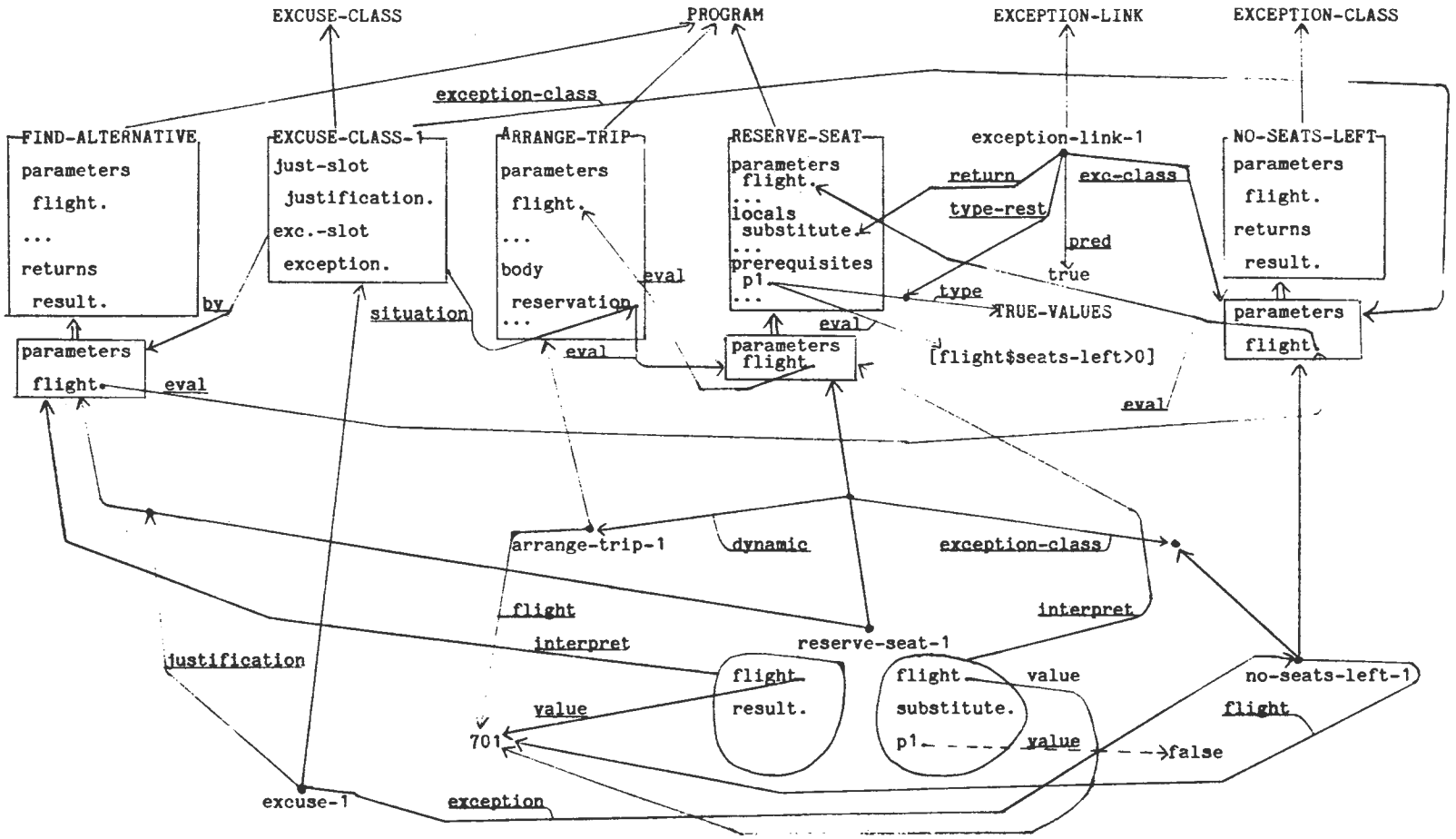



Figure 2 - Example of excuse for a dynamic exception.

their name with a node written in the box, for example "leg1". These slots can then be filled with values when an instance of the class has been created. This is represented by a link with the name of the slot as for the "leg1" of "Capt'n-Kidd" is "wooden-leg-1" in figure 1. The closure of these structural property value relationships forms the PART-OF hierarchy. The classes can also be organized in an IS-A or specialization hierarchy (represented by unlabled double line arrows, see figure 2). This facilitates the definition of the subclasses as the structure of the superclass is inherited by them. The slots can be refined but are required to satisfy the IS-A constraints, which guarantee that the subclasses are effectively specializations.
Slot values, in particular the four programs defining the semantics of classes, can also be inherited if necessary.

The instance hierarchy is not restricted to two levels and classes can be instances of metaclasses. This is used extensively in the definition of the formalism itself and many aspects of its behavior arise as a result of the definition of the metaclasses: CLASS, RELATION, OBJECT, PROGRAM,etc.. A metaclass can constrain the structure of its instances through its metastructure [Kramer 80], as the slots of the instance must be instances of the metaslots in the metastructure. Programs are represented as classes in the formalism, and thus benefit from all the declarative facilities. In figure 2, the program "ARRANGE-TRIP" calls another program "RESERVE-SEAT". Metaslots have been used to partition the slots into different categories: parameters, locals, etc.. To specify the desired parameter bindings and evaluations, a form is used (the box with no heading under "RESERVE-SEAT"). The programs are executed by creating processes which are instances of the programs, "arrange-trip-1" and "reserve-seat-1" in the example. The formalism also provides a context mechanism [Schneider 78, Schneider 80]. An object which is visible in a context is called a view. Context are used to implement inheritance, structures being essentialy special forms of contexts. A slot is inherited because it is visible (a view) in the structure of subclasses.

The only differences with some previous versions of PSN are the use of valuers to implement manifestations (ex: John as a taxpayer) as in [Schneider 78], which are needed for the proper treatment of dynamic exceptions, and the ability to refer to most systems assertions (INSTANCE, type, etc.). This feature can be simulated without any extension to PSN by replacing the single link assertion reference by a triple link reference to the relation and its arguments.

## 3.0 EXCUSES

### 3.1 STATIC EXCEPTIONS

The excuse mechanism takes care of objects which are instances of a class while violating some of the constraints associated to its slots. The exceptions which are raised by these violations must be handled by the class of the object which has the defective object as one of its parts (slot value), thus one level up on the PART-OF hierarchy. This provides a basic form of context sensitivity to the mechanism. The handler attached to the "situation" is restricted to being a class of which the defective object must also be an instance, thus retaining Minsky's idea of frame interaction in a context.

Let's explore the mechanism in more detail by considering an example of static exception handling represented graphicaly in figure 1. Here, we have an object "Capt'n-Kidd", which would be a legal instance of the class "HUMAN", except for the fact that the value of its slot "leg-1", "wooden-leg-1", violates the type constraint of the "leg-1" slot definition in the class "HUMAN". The violation is precisely that "wooden-leg-1" is not an instance of "HUMAN-LEG". To characterize this type of constraint violation, an exception-class called "NO-REAL-LEG" is created. Then this class is associated to the type of the slot "leg-1" using an exception-link. When the system, attempting to fill the value of "leg-1" for "Capt'n-Kidd" will detect the type violation, it will find the exception-link and then, if the predicate of the link is satisfied, it will create an instance of the exception class "NO-REAL-LEG". The exception "no-real-leg-1" is attached to the INSTANCE link between "Capt'n-Kidd" and "HUMAN", which thus becomes an EXCEPTIONAL-INSTANCE link. This is done by making the exception an instance of an exception-class created especialy for the link. Many exceptions could be raised on the instance in the same way.

The rest of the mechanism concerns the handling of the exception where the system tries to build an excuse for the exception. For that, it climbs up one level in the PART-OF hierarchy and looks at the corresponding class to find an excuse-class. In the example, this corresponds to following the "main-character" assertion to "story-1", then looking at its class "PIRATE-STORY" and then finding "EXCUSE-CLASS-1". This excuse-class must have been attached to the slot whose value is the exceptional instance. For the excuse-class to be usable, it must be associated to the exception-class of which the exception is an instance. If this is the case, then the system tries to make the exceptional object an instance of the class which is the value of its "by" slot, which is "DISABLED-PERSON" in this case. Any desired checking for evidence for this type of excuse can be done at this stage. If the instantiation has been succesful, then an excuse is created, which associates the justification to the exception. In the example, this is "excuse-1". The excuse marks the

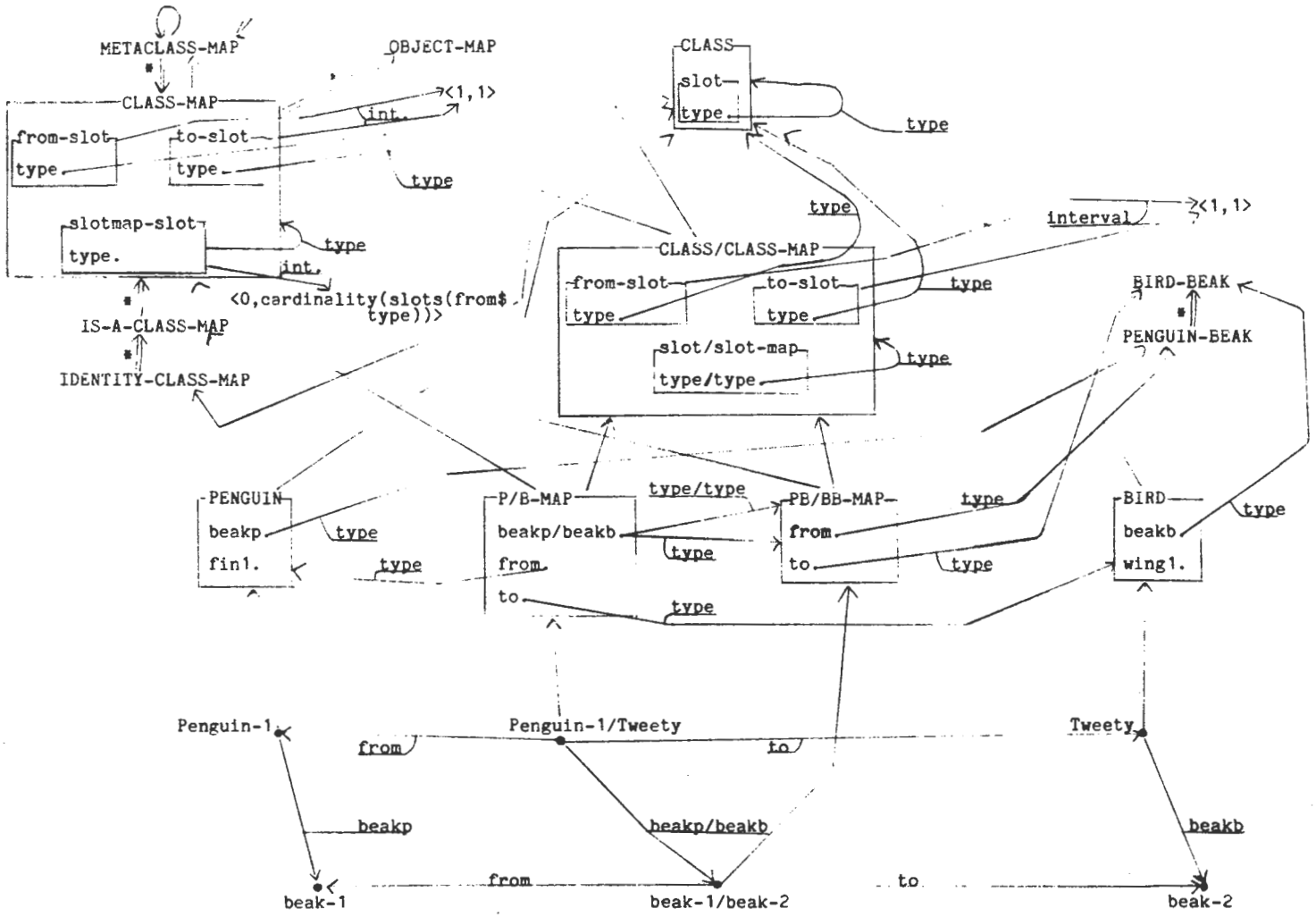



Figure 3 - Example of mapping.

succesful handling of the exception. If all the exceptions attached to an exceptional-instance link via its exception-class have been excused, then the link becomes an EXCUSED-INSTANCE link.

Exception-classes in this system have a two-fold function: they are abstract descriptions of the violations that arise and they allow an economical interface between the excuse-classes, which handle the violations, and the violations themselves, assuming that some violations will be treated in the same way. The use of the PART-OF hierarchy as a kind of context mechanism for exceptions is new to PSN, but resembles that of NETL [Fahlman 79]. The excuse mechanism also works nicely for cases of non-existant slot values. In this case, the special object "nothing" is given as a value. This can be treated as a type violation and be handled in the normal way.

## 3.2 DYNAMIC EXCEPTIONS

The excuse mechanism can be used to handle dynamic exceptions with a few extensions. It is natural to see exception-classes as the interface between the program context raising the exception and the one which will be selected to handle it. As these two belong to different levels of abstraction, it is necessary to provide parameter passing facilities with exceptions. These are defined as slots in the exception-class. The raising of an exception is similar to a procedure call, with the difference that the actual procedure to be invoked has to be selected by the system using the information provided by the excuse-classes. The scheme chosen requires the exception handling program to return control to the raiser of the exception after it has completed, as in [Levin 77]. This requires the definition of a returns slot in the exception-class.

In the example represented graphicaly in figure 2, a type violation has occured in the process "reserve-seat-1", which was invoked by "arrange-trip-1". The violation is on the prerequisite slot "p1", which checks whether some seats are available on the flight. As the value returned was "false", an instance of the exception-class "NO-SEATS-LEFT" is created ("no-seats-left-1") and attached to the INSTANCE link of the process. In the case of dynamic exception handling, the exception-link does not point directly to the exception-class, but to a form which is a subclass of it, allowing the parameter bindings to be indicated by "eval" assertions. A more important difference is the presence of a return slot value indicating which slot of the raiser should recieve the result of the evaluation of the exception handler.

After the creation of the exception, the system looks for an excuse-class (having the appropriate exception-class) attached to the slot that was being evaluated in the caller of the process that raised the exception. The dynamic hierarchy is used instead of PART-OF as it fills a similar role in dynamic objects like programs to that of part-of in static objects. Thus the "dynamic" assertion is followed from "reserve-seat-1" to "arrange-trip-1", where the "EXCUSE-CLASS-1" is located, from the "reservation" slot that was being evaluated. Then, the form which is the value of the "by" slot and a subclass of the "FIND-ALTERNATIVE" program is instantiated (executed), as the exception handler. Here again, a form is used to allow for the binding of parameters. The instance of the "by" class "FIND-ALTERNATIVE", is a manifestation of the same object "reserve-seat-1" that raised the exception. The explicit representation of the valuers (the ovals containing the value assignements to the slots) makes the separation of the two manifestations clear. The exception handling process thus appears as a tailoring of the process "reserve-seat-1" to fit the particular situation at hand. Once the instantiation has completed, an excuse is created ("excuse-1") for the succesfuly handled exception. Then, the "result" of the handler, that is the value of its slot which is an instance of the "returns" metaslot, can be passed back to the exception and to the process which raised it. This amounts in this case to set the local slot "substitute" to this value. Then, the process resumes after the point of interuption. A process can trigger an exception voluntarily by returning the special value "fail" in the same way as "nothing" in the static case.

## 3.3 INTERACTIONS WITH THE HIERARCHIES AND SEMANTICS

The immediate father in the PART-OF (dynamic) hierarchy is not always the best class to provide an excuse for an exception, but the scheme requires the exception to be reformulated in terms of the father class before it can be passed up higher, so as to preserve the abstraction structure. This is done in the static case by considering the unexcused exceptional object as violating the type of the father. In the dynamic case, the handler ("by" class) can also raise a new exception of its own, as it is treated as a part of the caller's context.

Even if it does not appears so by the examples given, it is intended that exception-links and excuse-classes be inherited with the slot they are attached to down the is-a hierarchy. They can also be refined and have to satisfy the is-a constraints (that their parts be identical or is-a, including the exception-class and the "by" class). This can be enforced by the formalism if these objects are defined as classes with slots representing the links, as in [Kramer 80]. However this solution is not totaly satisfactory. A default exception-class called "GENERAL-EXCEPTION-CLASS" is provided by the formalism to every slot defined, through the inheritance mechanism.

The excuse mechanism can be considered to be simply a syntactic extension of the original PSN formalism. The attachement of an exception-link and exception-class to a slot can be seen as the creation of a class which only differs from the original class by the required presence of the violation which would raise the exception. The attaohement of an excuse-class to a slot effects a modification of its type, generalizing it to include some of these "violation" classes.

## 4.0 MAPPINGS

Our goal in designing the mapping mechanism was to define a very general construct which would (1) provide a facility for describing similarities that exist between objects and (2) allow the definition of classes in terms of other classes, including the copying of parts of their structure on a piecemeal basis to enhance expressive efficiency. The motivation for this came mainly from the lack of flexibility of the current IS-A construct, which is heavily felt whe dealing with natural concepts. In fact, IS-A should appear as a particular specialization of the general mapping construct and as such, it cannot be used in its definition.

An example of application of this more general mapping construct would be defining the class "PENGUIN" in term of the class "BIRD" by specifying a mapping from "PENGUIN" to "BIRD" which includes, as a submapping, saying that the "beak" slot of "PENGUIN" has a type which is a particular specialization of that of the "beak" of "BIRD". This is represented graphicaly in figure 3, where "Pb-MAP" is such a mapping (more details later). In this definition process, the user creates a mapping and expects the mapping instantiation program to create all objects and views not already existing and have them form the class being defined in terms of the other, as a side-effect of the mapping instantiation. Two aspects of the definition of mappings can thus be identified: their structure, which is concerned with the description of the relationship between the two objects, and their side-effects, which include object creation and manipulation of the structure hierarchy (contexts) to effect inheritance. The rest of the presentation concerns mainly the structural aspect as the other still needs to be worked out in details.

The main influences on the mapping mechanism have been the mappings of MERLIN [Moore 73], where the recursive aspect of their definition is taken, the "cables" of KLONE [Brachman 79], for the idea of structured inheritance, and the similarity networks of [Winston 75].

The main idea on which the mechanism is based is that any mapping of an object must also involve the mapping of its type(s), as it is an essential part of its definition. This requirement causes the structure of mappings to mirrors closely that of the INSTANCE hierarchy. If we return to our example in figure 3, the mapping "P/B-MAP" between the clases "PENGUIN" and "BIRD" is also a class and an instance of "CLASS-MAP". It contains a slot-mapping slot, "beakp/beakb", from the "beakp" slot of "PENGUIN" to the "beakb" of "BIRD". The type of this slot, "PB/BB-MAP", is another mapping class from the type of "beakp", "PENGUIN-BEAK", to the type of "beakb", "BIRD-BEAK". "PB/BB-MAP" would itself be expanded in the same way to map the slots of both classes. Now at the token level, there is an instance of "P/B-MAP", mapping "penguin-1" to "Tweety". It has as slot value a mapping between both "beak" slot values, which is an instance of "PB/BB-MAP". Thus, the mapping at the class level allows us to map the instances of the class. The structure of the mappings is exactly parallel to that of the classes mapped.

However, to satisfy completely our requirement, the types of the classes "PENGUIN" and "BIRD" must also be mapped. This is accomplished by "CLASS/CLASS-MAP", which maps the class "CLASS" into itself. Note that both "P/B-MAP" and "PB/BB-MAP" are also instances of this metaclass. The type of "CLASS" itself, "METACLASS", would also need to be mapped, but eventualy this will stop as "METACLASS" is only an instance of itself.

The classes that define mappings ("CLASS-MAP", "METACLASS-MAP", etc.) also allow us to create a taxonomy of mappings and differentiate between identity mappings, IS-A mappings and general similarity mappings. This is done by gradualy adding more constraints on the structure of mappings (e.g. the "interval" of "CLASS-MAP"), mainly on the metaslot controling slot mappings ("slot-map-slot"). This produces a pseudo-IS-A hierarchy of mappings. In the example, the "PB/BB-MAP" is an instance of "IS-A-CLASS-MAP" and its argument classes would satisfy the IS-A constraints. "CLASS/CLASS-MAP" is an instance of "IDENTITY-CLASS-MAP" as it maps a class to itself.

The mapping construct allows the representation of similarities of similarities, as mappings are simply objects like everithing else. It is also a powerful tool to study relationships involving the parts of objects as well as the objects themselves. An interesting question raised by the characterization of IS-A as a class of mappings is whether its set-inclusion aspect (instances of subclasses are instances of superclasses) is simply a side-effect of the IS-A constraints or a supplementary relationship. A mapping class can also be devised which exibits the constraints of the INSTANCE relationship. However, this abstract comparison of existing structures should not be confused with the INSTANCE assertion itself, which is the result of an external recognition process starting from sensory features and whose existence is assumed by the mapping mechanism.

## 5.0 COMPARISON TO OTHER SCHEMES

The only other representation formalism to give significant attention to the static and generic exception problems is NETL [Fahlman 79]. Its solution is much simpler than ours, being based on the insertion of "CANCEL" links in the virtual copy hierarchy to cancel inheritance when needed. This may be considered analogous to a mapping mechanism based on differences. There is no need for excuses as NETL neither does include a separate instance hierarchy nor programs. The mechanism is defined at a lower level of abstraction than ours (the user is concerned with the inheritance process) and is affected by the emphasis on retrieval. It does not offer the descriptive facilities of our solution and does not enforce any consistency or justification requirement.

The excuse mechanism for dynamic exception handling has many points in common with those of [Kramer 80] and [Mylopoulos 79]. However, it differs essentialy with that of [Kramer 80] on the question of where control should be returned after the completion of the exception handler. We require the resumption of the process which raised the exception, rather than return control to its caller. This makes it easier to ensure that the model is not left in an inconsistent state, is more efficient and promotes a more natural view of abstractions.

A more logical approach to exceptions has recently been proposed. Exceptions are seen as entities for which some default inference rule does not hold [Reiter 78](e.g. birds fly unless we can prove otherwise, for penguins the rule does not hold). Systems based on this principle maintain justifications for their assertions and reevaluate them as new facts are learned, which may contradict existing defaults deductions [Doyle 79]. If a satisfactory (non-monotonic) logic can be found to characterize these systems, it could improve greatly our understanding of the nature of exceptions and how to deal with them.

## 6.0 CONCLUSION

Some work remains to be done to achieve the full potential of the excuse mechanism. It should be possible to extend it so as to accomodate "structural" exceptions that arise on objects shared among many program contexts, which need to be propagated along the user hierarchy instead of the dynamic hierarchy [Levin 77]. This would involve a better integration of static and dynamic exception handling. The side-effects aspect of the mapping mechanism also need to be worked out in details.

It is certainly necessary to experiment with both mechanisms on a larger scale, to see whether they are really useful and suggest improvements. This would show in particular whether the whole-to-part style of object definition (where the object is created before its parts), which is necessary to take full advantage of the excuse mechanism, is practical.

## REFERENCES


Brachman, R.J. (1979). "On the Epistemological Status of Semantic Networks",in Associative Networks: Representation and use of knowledge by computers, Findler, N.V. (Ed.), Academic Press, New York.

Doyle, J. (1979). "A Glimpse of Truth Maintenance", in Artificial Intelligence: An MIT Perspective, Winston, P.H. and Brown, R.H. (Eds.), MIT Press, Cambridge, Mass..

Fahlman, S.E. (1979). NETL: A System for Representing and Using Real-world Knowledge, MIT Press, Cambridge, Mass..

Kramer, B.M. (1980). "Representing Programs in PSN". Proc. 3rd Nat. CSCSI Conf., Victoria.

Lesperance, Y. (1980). Handling Exceptions in PSN. M.Sc. thesis, Dept. of Computer Science, Univ. of Toronto, to appear.

Levesque, H.J. and Mylopoulos, J. (1979). "A Procedural Semantics for Semantic Networks", in Associative Networks: Representation and use of knowledge by computers, Findler,N.V. (Ed.), Academic Press, New York.

Levin, R. (1977). Program structures for Exceptional Condition Handling. Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburg.

Mylopoulos, J., Bernstein, P. and Wong, H. (1979). A Language Facility for Designing Database-Intensive Applications. CSRG-TR-105, Dept. of Computer Science, Univ. of Toronto, to appear in TODS.

Minsky, M. (1974). A Framework for Representing Knowledge. A.I. Memo No. 306, MIT A.I. Lab., Cambridge, Mass..

Moore, J. and Newell, A. (1973). "How Can Merlin Understand ?", in Knowledge and Cognition, Gregg, L. (Ed.), Lawrence Erlbaum, Potomac, Md..

Reiter, R. (1978). "On Reasoning by Default", Proc. TINLAP-2, Urbana, Ill..

Schneider, P.F. (1978). Organization of Knowledge in a Procedural Semantic Network Formalism. Tech. Report No. 115, Dept. of Computer Science, Univ. of Toronto.

Schneider, P.F. (1980). "Contexts in PSN". Proc. 3rd Nat. CSCSI Conf., Victoria.

Winston, P.H. (1975). "Learning Structural Descriptions from Examples", in The Psychology of Computer Vision, Winston, P.H. (Ed.), McGraw Hill, New York.

# Contexts in PSN

Peter F. Schneider

Department of Computer Science
University of Toronto
Toronto, Ontario
M5S 1A7

Abstract

Contexts play an important part in PSN (the Procedural Semantic Network formalism). This paper discusses some of the problems that have been encountered with contexts in PSN and gives an informal presentation of the current definition of contexts in PSN. Particular attention is paid in this presentation to the notion of inheritance along the context hierarchy and its implications.

## 1. Introduction

PSN (the Procedural Semantic Network formalism) is a formalism for the representation of knowledge. The basis of PSN (as described in [Levesque 1977] and [Levesque and Mylopoulos 1979]) is the notion of a class with four attached procedures. These procedures are responsible for adding instances to, deleting instances from, recognizing instances of, and enumerating the instances of a class.

PSN also contains several traditional semantic network concepts in addition to this procedural basis. These concepts serve in part to impose order on the heterarchical nature of the procedures. In fact, throughout the remainder of this paper the procedural base of PSN may be largely ignored since the paper talks about these organizational principles, in particular contexts.

The main organizational semantic network principles in PSN include the INSTANCE-OF relationship, the IS-A relationship, and properties of objects (also known as the PART-OF relationship). The INSTANCE-OF relationship relates an object to the classes it is an instance of. The IS-A relationship relates a sub-class to a super-class and thus is concerned with the specialization and generalization of concepts. Properties may be associated with any class and define the kinds of information that can be incorporated into the instances of the class and thus are concerned with the aggregation of information. All of these organizational principles are well described in the papers mentioned above although an extension to properties has been added to PSN and is discussed in [Kramer 1980].

One non-standard aspect of PSN is that everything in PSN is an object and a member of a class. This means that classes themselves are members of other classes, notably the meta-class "CLASS", and thus may have properties as defined by "CLASS" (such as cardinality). So in PSN a typical object such as "John" is an instance of a class, namely "PERSON", which is in turn an instance of "CLASS". "CLASS" itself must be an instance of a class but with a little inspection it should be clear that "CLASS" itself is the appropriate class thus eliminating the need for ever more higher meta-classes.

## 2. Contexts

There is a fourth organizational principle in PSN which has been less investigated and much less understood than the above two. This principle is the ability to group objects into contexts, much like the contexts of CONNIVER [Sussman and McDermott 1972] or the partitions of Hendrix's partitioned nets [Hendrix 1975, 1979]. Recently a formal investigation of the properties of contexts in PSN has turned up some surprising aspects of contexts as they had been developed as well as clarifying many points concerning contexts and their interaction with the other organizational principles of PSN. This resulted in a formal definition of contexts [Schneider 1979]. This paper presents contexts in PSN in a much less formal manner than [Schneider 1979] and concentrates on inheritance between contexts.

Contexts in PSN are most often used to represent alternate views of the knowledge base. For example, one context could be used to represent the real world. This context, perhaps called "reality", would have all the knowledge pertaining to the real world. Another context could then be used to represent the well known alternate world where fairies exist. This context, perhaps called "fairy world", would have the same information as "reality" in most areas but would not correspond to "reality" in other areas and would have some additional information in yet others.

## 3. Contexts and Views

When contexts are considered it no longer suffices to think of objects in isolation. Instead, it is necessary to look at an object as seen in a context, such as "John" in "reality" or "FAIRY" in "fairy world". It is also possible to consider an object in two or more different contexts such as "John" in "reality" versus "John" in "fairy world". In these different views "John" may have different properties but the views are still views of the same object.

So far this is fairly straightforward. However, in PSN contexts are objects (because everything in PSN is an object). But objects and thus also contexts cannot be considered as objects in isolation but only as objects as seen in a context. For example, "fairy world" as seen in "fairy world" is different from "fairy world" as seen in "reality". This interaction requires a change in the definition of contexts.

The redefinition of contexts in PSN proceeds as follows: First a <u>view</u> in PSN is defined to be an object as seen in a context. Then a <u>context</u> is defined to be a special type of view in which the view is an instance of the object "CONTEXT". A little inspection of this definition shows

that there is no way of creating any non-infinite views (or contexts) since any view must contain another view. This is corrected by creating a single base context, the universal context. The universal context is thus the only view (and context) not consisting of an object as seen in a context and must form part of every view (and context) in PSN.

This makes it possible to create contexts and views of varying complexity. For example, the object "John" as seen in the universal context is a perfectly ordinary view, essentially corresponding to "John" in PSN without contexts. The context "reality" in the universal context is an ordinary context and "Bill" as seen in this context is a view corresponding to a view in other approaches to contexts. This situation (plus others mentioned below) is illustrated in Figure 1.*

In PSN an object may be part of several views (as is "John" in Figure 1) and if an object forms a view in a context then that object can be referenced in the context. Each view of an object may have different values for its properties, may be an instance of different classes, and thus also may have different properties.

---

*In each figure the largest box represents the universal context and the names each represent a view with those names that are attached to boxes representing contexts. The containment relationship in the figure represents the 'as seen in' relationship between objects and contexts in PSN. Also unlabelled single arrows represent the instance relationship, unlabelled double arrows represent the IS-A relationship, and labelled single arrows represent property values with the labels being the property names. Property definitions are not shown in the figures.

The views of "John" in "reality" in the universal context and in "fairy world" in the universal context in Figure 1 illustrate these possibilities. Also an object need not form views in every context. (For example, "Bill" in "reality" in the universal context is a view in Figure 1 but "Bill" in the universal context is not.) If an object does not form a view in a context then it cannot be referenced in that context and definitely cannot exist in that context. This is, of course, also allowed in other context mechanisms.

However, there is no prohibition against having more than one view of an object being a context. This situation is illustrated in Figure 1 where there are two contexts using the object "fairy world". The context "fairy world" in "reality" in the universal context may be quite different from the context "fairy world" in the universal context. For example, instances of "FAIRY" in "fairy world" in the universal context may not be able to perform magic whereas instances of

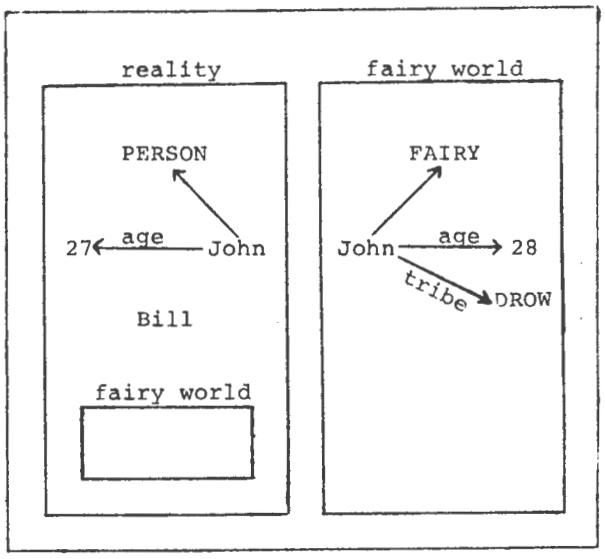



Figure 1

"FAIRY" in "fairy world" in "reality" in the universal context may. This shows one additional ability of this approach to contexts over the other, more traditional approaches which may be of use in deductive mechanisms or natural language interfaces that consider the different views of an object as being related.

## 4. Inheritance in Contexts

If the contexts "fairy world" in "reality" in the universal context and "reality" in the universal context were part of a knowledge base they would contain a lot of information not related to fairies and thus the two contexts would have many objects and much information in common. This indicates that the former context should inherit this shared knowledge from the latter. However, not all the objects in the latter would be in the former and there may be other differences between the contexts and thus this inheritance should not be strict.

To facilitate talking about the inheritance between contexts in PSN a context hierarchy exists in PSN. This hierarchy is defined by the following rule: If a view of an object in a context forms another context then its parent in the context hierarchy is the context in which it is seen. Thus the parents of "reality" in the universal context and of "fairy world" in the universal context are both the universal context. Further, the parent of "fairy world" in "fairy world" in "reality" in the universal context is "fairy world" in "reality" in the universal context and its parent is "reality" in the universal context. This gives rise to a tree hierarchy where inheritance between contexts goes down the hierarchy (similar to inheritance between classes down the IS-A hierarchy).

This inheritance differs from inheritance down the IS-A hierarchy in several ways. First, the things inherited are different. Definitions of properties and values of properties are inherited down the IS-A hierarchy whereas everything in PSN including objects, properties of objects, and definitions of properties for objects are inherited down the context hierarchy. Second, the inheritance down the IS-A hierarchy is very strict in some aspects: as far as definitions of properties go only additions of new definitions and specializations of existing definitions are allowed from a class to a sub-class. Inheritance down the context hierarchy is not nearly so strict and any change can be made between a context and its children in the context hierarchy.

The inheritance proceeds as follows: if an object A in context c is a view and if B in c is a context then A in B in c will be a view unless explicitly deleted by B in c. The same is true of A being an instance of a class in c. The properties of A in B in c are defined, as before, to be those properties defined in the classes of which A in B in c is an instance and the property values for these properties are inherited in the same fashion as views and instances except that properties are single valued so any new value for a property will override the inherited one.

For example, if "John" in the universal context is a view and "fairy world" in the universal context is a context then by context inheritance "John" in "fairy world" in the universal context is a view. However, this may be overridden so that "John" in "fairy world" in the universal context is not a view.

Further, even if it is a view then "John" could be an instance of a different class in the two contexts, perhaps "PERSON" in one context and "FAIRY" in the other (as in Figure 2). Here it is not necessary for "FAIRY" to be an IS-A descendant of "PERSON" as would be the case with IS-A inheritance. Other properties associated with "John" may change between the two contexts, such as "John"'s "height", and some may in fact disappear or appear, such as "John"'s "tribe", if the sets of properties defined in "PERSON" and "FAIRY" are different. Note that "John"'s "age" is not changed in "fairy world" in the universal context and thus will be inherited from the universal context. Of course this is not much different from other context schemes such as Hendrix's [Hendrix 1979] or Fahlman's [Fahlman 1979].

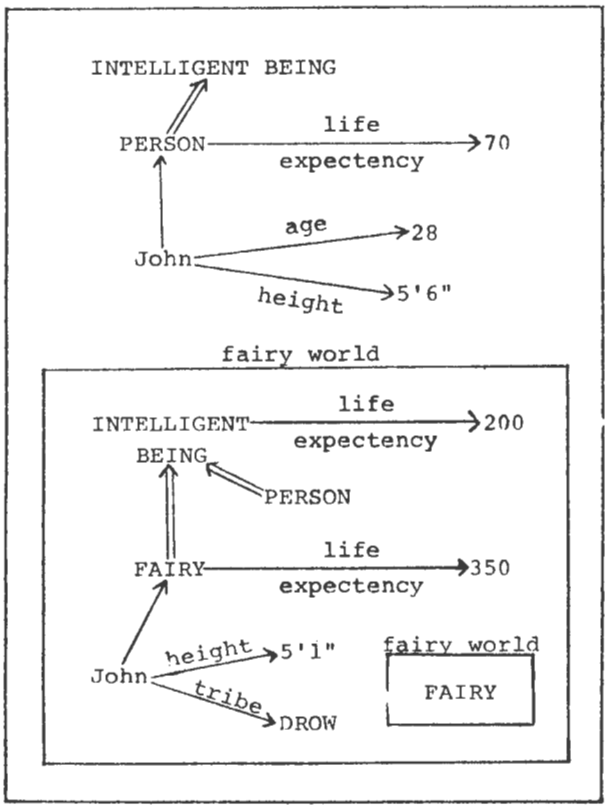



Figure 2

The situation is more complicated for classes because of the interaction with IS-A inheritance. Views of classes and the IS-A relationship are inherited in the same fashion as views of normal objects and the instance relationship. The same is true for properties of classes but for property values there are two possible places to inherit from, the IS-A parents in the same context and the view of the class in the parent context. The solution adopted is to inherit from the parent context if possible, and only if this fails to produce a value inherit from the IS-A parents. This reflects the opinion that different views of an object are more likely to have the same property values than a class and its parents in the IS-A hierarchy. This situation is illustrated in Figure 2 where the "life expectancy" for "PERSON" in "fairy world" in the universal context is inherited from "PERSON" in the universal context and not from "INTELLIGENT BEING" in "fairy world" in the universal context.

Property definitions have the same problem of where to look in inheritance with the extra added problem of the strictness of IS-A inheritance of property definitions. The solution is to inherit as follows: First inherit the property definitions from the IS-A parents. Then if there are changes to the property definitions in the view of the class in the parent context, they will modify, but only to specialize, the IS-A inherited property definitions. Any modifications that would violate the IS-A inheritance rules are not carried out. Finally any modifications in the class itself serve to override the modifications inherited from the parent context as long as they do not violate the IS-A inheritance rules. If

there are property definitions introduced in the view in the parent context that are not in the IS-A parents then these are inherited and can be changed in the class itself. These inheritance rules serve to retain the strict IS-A inheritance of property definitions while allowing non-strict inheritance down the context hierarchy. Again, this is not much different from other context mechanisms, at least as far as the context inheritance goes.

## 5. Inheritance of Contexts

The new aspects of this approach surface if different views of contexts and inheritance of contexts down the context hierarchy are considered. In other context mechanisms, contexts are second class citizens because they, as opposed to regular objects, do not exist as seen in a context. Thus they cannot have different views and cannot be inherited between contexts.

The context mechanism in PSN allows both of these situations. For example, consider the context "fairy world" in the universal context as shown in Figure 2. Objects, such as "John", can form views in this context. However, "fairy world" is an object and so it too can form a view in the context "fairy world" in the universal context, namely the context "fairy world" in "fairy world" in the universal context. This cannot be done in other context mechanisms and the advantage to doing this is that there may well be properties attached to "fairy world" in the universal context such as cautions that unusual things may occur in it and these properties will be inherited by "fairy world" in "fairy world" in the universal context although as with all context inheritance this may be overridden. Further, these two contexts are different views of the same object and this may be of use to deduction mechanisms and natural language interfaces.

Note that context inheritance would also create the context "fairy world" in "fairy world" in "fairy world" in the universal context and then go on ever deeper ad infinitum. This infinite regression of contexts is not really a problem since at some point they will cease to be of interest and so they need not be explicitly stored. If at any time more become of interest then they will be available automatically. However, this infinite regression does preclude calculating all inheritance beforehand (which is not a good idea in any case).

Another advantage of this approach to contexts is illustrated in Figure 3 where one of the views shown represents "FAIRY" in "fairy world" in "John's beliefs" in the universal context. In this figure

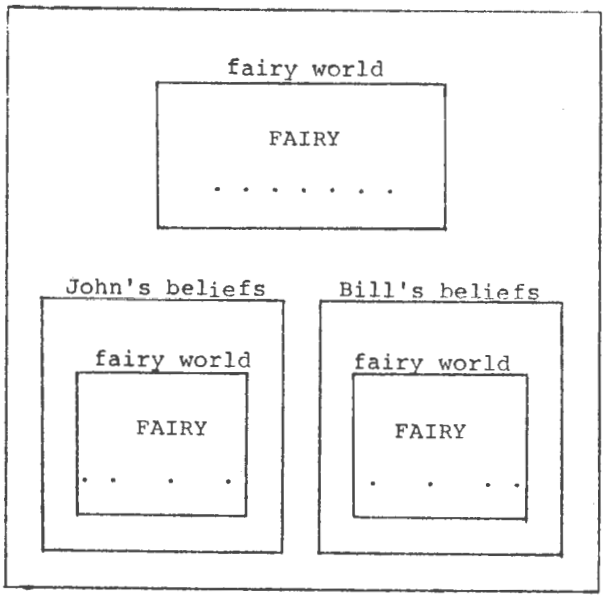



Figure 3

there are three views of the object "fairy world", namely "fairy world" in the universal context, "fairy world" in "John's beliefs" in the universal context, and "fairy world" in "Bill's beliefs" in the universal context and all three of these views are, in fact, contexts. Now, the last two of these inherit from the first because properties of objects including the views in a context are inherited down the context hierarchy (from the universal context to "John's beliefs" in the universal context and "Bill's beliefs" in the universal context). Thus the two lower (in the diagram) views of the object "FAIRY" are inherited from the upper view and unless explicit changes are made will have the same properties as the upper view.

Even if explicit changes are made to the lower views they will inherit the unchanged information from the upper view and will be generally the same as the other lower view (unless massive changes are made). This cannot be done by using a context that is a sub-context of both "John's beliefs" and "Bill's beliefs" since the sub-context method can only represent situations where some knowledge is exactly the same in two contexts.

This inheritance of contexts does cause some complications to context inheritance because it introduces more than one context that can be inherited from. The context "fairy world" in "John's beliefs" in the universal context can inherit views and other information from "John's beliefs" in the universal context (its parent in the context hierarchy) and also from "fairy world" in the universal context (a slightly less involved view of itself). Both of these are reasonable sources for inheritance. The solution is to inherit information from both places. This causes problems if the two sources are contradictory, such as two different values for a property, and in these cases no information would be inherited. The actual resolution of these problems is tedious because of the many cases and will not be discussed here. Note, however, that the simple presence of information versus its absence is not a problem because the information will simply be inherited from where it is present. This more complicated inheritance mechanism still retains the non-strict nature of context inheritance by allowing all context inherited information to be overridden while also retaining the IS-A inheritance restrictions.

## 6. Conclusion

The main contribution of contexts in PSN is the formation of the context hierarchy which provides a fourth organizational principle for structuring knowledge in PSN. Unfortunately, contexts and the context hierarchy have been less understood than they should have been. The discussion in this paper of some of the aspects of the formal definition of PSN contexts given in [Schneider 1979] will help to alleviate this problem.

The most significant aspects of the treatment of contexts proposed here are the additional possibilities inherent in it over more traditional treatments. These come about because contexts are not second class citizens in PSN. Instead, they are formed from objects just like any other view in a PSN knowledge base. Thus they are instances of classes (in fact they are defined as the instances of the class "CONTEXT") and can have properties and participate in relationships just like

other views in PSN. This is achieved in other systems, if at all, by extra machinery, such as associating contexts with super-nodes (as in Hendrix's Partitioned Networks [Hendrix 1979]).

Also the objects which form contexts may form different views each of which may be a context. These different contexts are still views of the same object just as a regular object may have several different views. This allows contexts, along with regular views, to be inherited down the context hierarchy. This inheritance, although it causes some complications in inheritance between contexts, allows large chunks of knowledge to be inherited and shared between contexts with modifications to portions of it still possible while retaining its ability to be referenced as an entity, something no other context formulation allows.

There is still work remaining to be done on contexts in PSN. Their formal definition does not take into account the procedural aspects of PSN and needs to be extended in this direction hopefully leading to a complete formal definition of PSN. Also, contexts should be integrated into the existing implementation of PSN at the University of Toronto (as described in [Kramer 1980]). However, even with these shortcomings contexts have much to add to PSN.

## Bibliography


[Fahlman 1979] Fahlman, Scott E. NETL: A system for representing and using real-world knowledge. Cambridge, Massachusetts: The MIT Press, 1979.

[Hendrix 1975] Hendrix, Gary G. "Expanding the Utility of Semantic Networks through Partitioning". Proceedings Fourth IJCAI, Tbilisi, U. S. S. R., 1975.

[Hendrix 1979] Hendrix, Gary G. "Encoding Knowledge in Partitioned Networks" in Associative Networks: Representation and use of knowledge by computers, ed. Nicholas V. Findler. New York: Associated Press, 1979.

[Kramer 1980] Kramer, Bryan M. "The Representation of Programs in the Procedural Semantic Network Formalism". Technical Report No. 139, Department of Computer Science, University of Toronto, 1980.

[Levesque 1977] Levesque, Hector J. "A Procedural Approach to Semantic Networks". Technical Report No. 105, Department of Computer Science, University of Toronto, 1977.

[Levesque and Mylopoulos 1979] "A Procedural Semantics for Semantic Networks" in Associative Networks: Representation and use of knowledge by computers, ed. Nicholas V. Findler. New York: Associated Press, 1979.

[Schneider 1979] Schneider, Peter F. "A Formal Definition of Contexts in the Procedural Semantic Network Formalism". AI Memo 79-5, Department of Computer Science, University of Toronto, 1979.

[Sussman and McDermott 1972] Sussman, Gerald J. and Drew V. McDermott. "From PLANNER to CONNIVER: A genetic approach". Proceedings FJCC, Volume 41, part 2, 1972.

# Representing Programs in PSN

Bryan M. Kramer

Department of Computer Science
University of Toronto
Toronto, Ontario
M5S 1A7

Abstract

The procedural semantic network (PSN) formalism for representing knowledge has as a basic concept the use of programs to define the semantics of classes of objects. This paper investigates a means of representing programs based on work done by the author ([Kramer 1979], [Kramer 1980]). Included as an important part of representing programs is an extension of PSN which provides a means for categorizing the properties of objects.

## 1. Introduction

The representation of programs as objects of the knowledge base has always been an important part of the procedural semantic network (PSN) formalism ([Levesque 1977], [Levesque and Mylopoulos 1979], [Schneider 1978a], [Schneider 1978b]). This work has been continued in the development of the language TAXIS ([Mylopoulos et al. 1978], [Wong 1980]) which embodies many of the concepts of PSN although its emphasis is on the design of interactive information systems rather than knowledge bases. The behaviour of programs in TAXIS is used in [Kramer 1980] and in this paper as a new basis for representing programs in PSN. The contributions of TAXIS are a new mechanism for associating the statements of a program with the program and a general exception handling mechanism.

The new proposals for the treatment of programs in PSN are discussed in more detail in [Kramer 1980]. This discussion includes some work on the handling of exceptions in the dynamic environment of programs. Exception handling in PSN, however, involves more than the correction of such exceptions. A different kind of exception is that which occurs in the maintenance of an object in the knowledge base which fails to meet some restrictions. For more details on the handling of both kinds of exceptions the reader is referred to [Lesperance 1980].

## 2. Overview of PSN

Programs enter into PSN as entities describing the behaviour of classes and relations. Every object is an instance of a class; the programs of the class specify how an object might be made such an instance, how an instance should be removed from the class, a test for membership in the class, and a mechanism for fetching all instances of the class. Binary relations are represented by classes known as relations. The instances of a relation are assertions that pairs of objects belong to the relation. The four programs for a relation add assertions, remove assertions, test that a pair of objects is a member of the relation, and given an object "x", fetch all objects "y" for which an assertion of the relation between "x" and "y" holds.

In addition to the basic mechanism for describing their behaviour, classes are provided with a means of defining the properties which instances may have. For example, one might define the class "PERSON" whose instances are people, and include in the definition a description of the property "eye colour". Instances of the class may then have values for these properties; thus the object "John" might have the value "blue" for the property "eye colour". This mechanism is similar to the binary relations. For example, an alternative to defining "eye colour" as a property in "PERSON", one could define a relation "EYE COLOUR" whose domain is "PERSON" and range is "COLOUR". The former mechanism is used for properties which the designer of the knowledge base considers to be definitional: properties which characterize the objects. For example, one might consider a person's social insurance number and sex as definitional properties. Once assigned, the values of such properties may not be changed.

The four programs associated with a class or relation are attached as property values of that class or relation. The definitions of these properties are provided in the classes "CLASS" and "RELATION". Thus "CLASS" contains definitions of the properties "to add", "to remove", "to test", and "to fetch" and "RELATION" provides similar definitions for relations. Classes such as "CLASS" and "RELATION" which have classes as instances are known as metaclasses.

The definition of a property in a class is represented by an object associated with the class. Such an object is called a slot and is said to exist in a class. Slots, being objects,

will themselves have property values. These values serve to provide the constraints on the property values of instances of the class. An important example is the "type" property of a slot. A property value of an instance of a class must be an instance of every member of the set of classes which is the type of the corresponding slot. If the type of the slot "eye colour" in the above example were "COLOUR", only colours may be the corresponding property values, and for "John" to have "eye colour" "blue" it is required that "blue" be an instance of "COLOUR". The exact mechanism used for defining the properties of slots is one of the concerns of this paper.

In continuing the example, there may arise a need to discuss more specialized classes of colours. For example, in discussing eye colours one might wish to distinguish a class of brown eye colours such as brown and hazel from a class of blue eye colours. One would then use the classes "BROWN COLOURS" and "BLUE COLOURS". It is however desirable that any instance of these classes remains an instance of the class "COLOUR". This specialization can be represented through the PSN supplied relation IS-A. If IS-A holds between "BROWN COLOURS" and "COLOUR", "BROWN COLOURS" is a subclass or IS-A child of "COLOUR" and "COLOUR" is a superclass of "BROWN COLOURS". Now, if "brown" is an instance of the subclass, it is automatically an instance of the superclass.

It is the responsibility of the four programs associated with the classes to insure that IS-A behaves in the proper manner. If "B" is a subclass of "A", the program which adds an instance to "B" must also make the object an instance of "A". In other words, once the add program of "B" has been run with an object "b" as a parameter, the test program of "A" must recognize this object as an instance of "A", the fetch program should fetch it, and the remove program should remove from "A" (and at the same time from "B"). The representation of programs in PSN provides a relationship between programs which constrains the assignment of programs to subclasses of a class. When this relation holds between the programs "p" and "q" where "q" is the add program for "A", "p" will be a valid add program for "B".

Another aspect of IS-A which is relevant here is the inheritance of structure (the set of slots in a class). When one defines a class "BROWN EYED PERSON" as a subclass of "PERSON", the slot "eye colour" is automatically contained in the new class. The properties of the slot may be modified in this process of inheritance. In the example, one would constrain the type of the "eye colour" to be "BROWN COLOUR" so that all instances of the new class may have only eye colours which are instances of "BROWN COLOUR". In general, when a property of a slot is modified in inheritance, the new value must be an IS-A descendant of the inherited value (as "BROWN COLOUR" is a subclass of "COLOUR").

## 3. Programs

Each PSN program consists of three groups of statements: the prerequisites, the body, and the returns statement. This division of programs into parts is intended to simplify the writing and understanding of programs. In earlier versions of PSN, a fourth group of statements was included to handle cases where failure occurred in the execution of the body. This has been replaced by a more general exception handling mechanism based on that of TAXIS. TAXIS programs too include a fourth group of statements called results or post conditions which have not yet been incorporated into PSN.

The execution of a program begins by the evaluation of the prerequisites. Should any of these not return true the exception handling mechanism is invoked. This involves the creation of an object called an exception which describes the circumstances of the failure. The process which failed is terminated. The calling program may provide a procedure to handle this exception. The value returned by this exception handler is used in place of the value which the failed program might have returned. Thus a program for division might have as a prerequisite a check that the divisor is not zero. If this prerequisite fails, the calling routine could replace the division by a program which simply returns an arbitrary value, for example zero.

Once the prerequisites have succeeded, the statements of the body are executed. These statements perform the major functions of the program, possibly modifying the knowledge base. Once the statements of the body have all been executed, the expression in the final group is executed returning the value which is to be returned by the program.

The statements of the body of the program may cause changes in the contents of the knowledge base. Such changes fall into two categories: the first is the set of changes caused by the interpreter as it executes the program; the second is the set of changes a statement is intended to perform. The first category of changes is generally not permanent: that is, when the execution of the program is complete, the state of the knowledge base will be unchanged except for the changes in the second category. The second category of changes is known as the set of side effects of the program. As an example, one side effect of an add program will generally be a link joining an object to a class.

Not all of the side effects of a statement in the body remain when the program is completed. It is possible, for example, that a statement near the beginning of execution will assert a relation between two objects and that a later statement will undo this assertion. The net side effects of a program are those side effects which remain when execution is complete. In other words, the set of net side effects of a program

is the set of differences between the knowledge base before execution and the knowledge base after execution. A function is a program which has no net side effects under any conditions (for any set of parameters in any state of the knowledge base). Functions are the only programs which may be invoked when the returns statement of a program is being executed. The prerequisites must also invoke only functions. Thus the side effects of any program must be performed by the statements of the body.

The side effects of a program are further constrained by the fact that all of the statements in any group, that is prerequisites or body, are to be executed simultaneously. Thus if more than one prerequisite fails, a number of exceptions will be raised. In the case of the body, this implies that the side effects of one statement may not depend on those of another. For example, one may not include two statements where the first creates an object and the second asserts that this object participates in some relation because one cannot guarantee that the new object exists when the assertion is attempted. This restriction will become significant when the specialization of programs is considered.

## 4. Programs as Classes

When representing programs in PSN one would like to use the tools for organizing knowledge already existing in the formalism. Thus the relation representing specialization of programs becomes IS-A and classes represent programs. By associating the statements of a program with the slots of a class, the existing inheritance mechanisms provide restrictions on the specialized programs. Also, using classes as programs allows the use of instances of these classes to represent activations of a the programs. Such instances of programs are known as processes.

Consider as an example a program which will compute the reciprocal of its argument. It will have a parameter, say "x", which is a number, a prerequisite which checks that "x" is not zero, and a returns statement which divides one by "x". This program can be represented in PSN as follows (the details will be explained as the text develops the representation):

```
(invert INSTANCE-OF PROGRAM
   STRUCTURE
      (x INSTANCE-OF parameters
         PROPERTY-VALUES type NUMBER)
      (not-equal-zero
         INSTANCE-OF prerequisites
         PROPERTY-VALUES
            eval
            (#1-not-equal
               INSTANCE-OF PROGRAM FORM
               IS-A not-equal
               STRUCTURE
                  (left
                     INSTANCE-OF
                     action_parameters
                     PROPERTY-VALUES eval x)
                  (right
                     INSTANCE-OF
                     quote_parameters
                     PROPERTY-VALUES quote 0))
            exception [new ZERO-DIVIDE])
(divide-arg INSTANCE-OF returns
   PROPERTY-VALUES
      eval
      (#1-divide
         INSTANCE-OF FORM PROGRAM
         IS-A divide
         STRUCTURE
            (left-d
               INSTANCE-OF
               quote_parameters
               PROPERTY-VALUES quote 1)
            (right-d
               INSTANCE-OF
               eval_parameter
               PROPERTY-VALUES eval
               x)))).
```

The notation used to illustrate the program represents an object by a list enclosed in parentheses or by an external name which acts as a reference to the object. In the list representation of an object, the first string is the external name which will be used to refer to that object. The remainder of the list consists of keywords followed by references to objects. The INSTANCE-OF keyword is followed by a list of the classes of which the object is an instance; for example, "invert" is an instance of the metaclass "PROGRAM". The IS-A keyword is followed by the classes of which a class is a subclass; in the example, "#1-not-equal" is a subclass of "not-equal". The STRUCTURE keyword is followed by a list of objects contained in the structure of a class. The structure of "invert" contains the slots "x", "not-equal-zero", and "divide-args". Finally, the keyword PROPERTY-VALUES precedes a list of pairs of objects of which the first object is a slot and the second the corresponding property value. In the example the object "divide-args" has as its "eval" property value the object "#1-divide".

The illustrations of objects will in general include only relevant and non-redundant detail. For example "PROGRAM" is a subclass of "CLASS", therefore "invert" is an instance of "CLASS", but only "PROGRAM" is mentioned in the INSTANCE-OF list. In the case of structure, inherited objects are shown only if their property values differ from those they have in an IS-A parent.

The nature of a PSN program is determined by its slots and their properties. One role that any slot in a program plays is that of a location for storing a value. The value stored for a given slot in a given activation is the property value bound to the process representing the activation. In the language of PSN, the slot plays the role of a variable, although it is not truly variable in that a property value of an

object, once bound, may never be changed. This may at first appear to be a severe restriction for programs requiring iteration. It is, however, possible to implement iterative control structures using recursion. Such an implementation of a FOR loop is illustrated in [Kramer 1980].

The parameter "x" of the program "invert" illustrates the use of a PSN variable. The object "#1-divide" is an expression in the program; the use of "x" as the "eval" property value of the slot "right-d" indicates that the value of the property "x" will be used in the division. The variable binding mechanism which results is static: the use of "x" in the expression "#1-divide" will always mean "x" in the program "invert".

Such a reference to a variable in an expression of a program may be to any slot in any program in the knowledge base. The interpreter will have to decide from which instance of the program containing the slot the binding should be taken. This situation is similar to that occurring in recursion in other programming languages. The use of a variable "X" in a recursive ALGOL program refers to the value stored in the most recent activation of the procedure. The PSN mechanism provides the same effect.

What is required is a mechanism for maintaining a record of the order of activation of programs. This is done through assertions of the relation "dynamic" between processes. For example, the execution of "invert" will be represented by a process, say "invert1". At some point in the execution, the program "divide" will be invoked. An instance of "divide", say "divide1" will be created and linked to "invert1" with an assertion of "dynamic". Now, when a slot is referenced as a variable the interpreter searches the chain of "dynamic" links for the first process that is an instance of a program containing that slot. Thus, when the reference to "x" occurs in "divide1", the interpreter will search along the "dynamic" assertions starting at "divide1". Since "invert1" is an instance of "invert" which is the class containing "x", the value of the variable is associated with "invert1".

The second role played by slots in a program is that of statements of the program. A statement consists of a slot and an associated form, a PSN object which represents an expression. Examples of forms in "invert" are "#1-not-equal" and "#1-divide". When the statement is executed, the value resulting from the execution of the form becomes the value of the slot in its role as a variable. For example, if the process "invert1" had value "5" for the property "x", it would receive value "true" for the property "not-equal-zero" after the execution of the form "#1-not-equal" (five does not equal zero).

The expressions associated with the slots in programs are objects which provide a value, either as calls to other programs as in the slot "not-equal-zero", or as references to the values of other slots as in the slot "left" of "#1-not-equal". When used as a call to another program, the form provides a mapping between the parameters of the program to be called and variables in the scope of the calling programs (or expressions in such variables). In programming languages this binding is usually represented positionally. For example, in ALGOL a procedure of two arguments called "FOO" is called by the form "FOO(X,Y)" where "X" and "Y" are variables known in the calling program. The values are bound to the parameters appearing in the same position in the definition of "FOO". PSN, however, performs this binding by including explicit links from the parameters of a copy of the program to variables or other forms. This copy of the program is an instance of the class "FORM" and at the same time an IS-A child of the program. It therefore inherits the structure of the program, and in particular the parameters slots of the program. The bindings of the parameters are represented by "eval" property values of these slots.

Often in a form one will wish to use a constant instead of another expression. In such cases, the value provided for a parameters slot is taken, not from an "eval" property value, but from a "quote" property value. Thus there are two ways of binding the parameters of a program to produce a form. A PSN form is either a slot or a subclass of a program whose parameters are bound by either an "eval" property to another form or by a "quote" property to any object. In the example, the "quote" property is used in the form "#1-not-equal" to bind the parameter "right" to the value "0".

## 5. Metastructure

When the various constructs in a program are represented by slots there must be a mechanism for distinguishing the functions of these objects. The interpreter must be able to decide whether a given slot is, say, a prerequisite or a parameter. Also, it will be necessary to fetch all the prerequisites or all the slots of the body. In addition, a mechanism is required for associating expressions with slots. The metastructure concept is introduced to enable these operations.

The objects in the structure of a class are categorised because they must be instances of classes. These classes are contained in the structures of metaclasses. If "A" is a class and B is the set of metaclasses of which it is an instance, then each object in the structure of "A" must be an instance of some object in the structure of some member of B. The metaclasses therefore organize the structure of the class. For example, the metaclass "PROGRAM" contains the objects "prerequisites", "body", and "returns", therefore the slots of a program may be

distinguished by the classes of which they are instances. In the program "invert" of the previous section the slot "not-equal-zero" is an instance of "prerequisites" and the slot "divide-arg" is an instance of "returns". Objects in a metaclass whose instances are slots are known as metaslots and the subset of the structure containing metaslots is called the metastructure of the class.

Since they have instances, metaslots must be classes. As classes, they may define the properties of their instances. The prime example of a metaslot in this role is the class "slot" which has all slots as instances. This class is a part of the metastructure of the metaclass "CLASS" thereby allowing any class to contain slots. The structure of "slot" consists of definitions for the type, default, and restriction properties of slots. It is interesting to note that the structure of "slot" is made of instances of itself. This works because it is an instance of "CLASS" as well as being a member of the structure of "CLASS".

The class "slot" is important in PSN because only instances of this object define properties of instances of classes. Thus any metaslot must be a subclass of "slot". One reason for specializing "slot" is to provide more properties to a class of slots. For example, the metaclass "PROGRAM" contains a metaslot called "action_slot" which defines a new property called "eval". The value of the "eval" property of a slot is the associated expression necessary in the definition of programs. Since "action_slot" is a subclass of slot, any instance of the metaslot will have all of the properties of regular slots.

In inheritance, all elements of a structure are treated uniformly. Slots and metaslots are inherited in the same way. If "B" is a subclass of "A" and C is the set of classes of which "B" is an instance, "B" will inherit all objects in the structure of "A" which are instances of some object in the structure of some member of C. When objects in a structure are inherited, property values may be modified subject to the rule that the new values must be IS-A descendants of the old. (Where the values are not classes they must remain unchanged.)

A new aspect of inheritance is that the inherited object may be made an instance of more classes so long as it remains an instance of each class of which it was an instance in the IS-A parent. For example, the parameters of certain subclasses of programs may be of one of two types represented by the subclasses "action_parameters" and "quote_parameters" of the metaslot "parameters". When a program is specialized, parameters slots which are simply "parameters" in the IS-A parent may become instances of one or the other subclasses of "parameters". This is illustrated by the slot "left" of the form "#1-not-equal" as shown in the example of section four. In the program "not-equal" from which it is inherited, the slot would be an instance of the metaslot "parameters". In the form "#1-not-equal" it is an instance of "action_parameters". The specialized slot will however also remain an instance of "parameters" and thus satisfy the rules of inheritance.

The class "slot" in particular, and metaslots in general, must be instances of objects in the structures of higher levels of classes. The class "METACLASS" containing the metaclass "metaslot" is introduced for this purpose. "METACLASS" is an instance of itself, as is "metaslot". In this way one avoids the introduction of an endless chain of classes each providing the means for the definition of structure in its instances. "metaslot" is an instance of an element of the structure --- "metaslot" --- of a class --- "METACLASS" --- of which its containing class is an instance. "METACLASS" is the class of all metaclasses and "metaslot" the class of all metaslots. However, an instance of "METACLASS" is a true metaclass only if it is a subclass of "CLASS", and similarily, an instance of "metaslot" must be a subclass of "metaslot" to be a true metaslot.

In many cases it will be desirable to limit the number of instances of a given metaslot in a class by specifying upper and lower bounds on this number. For example, programs may have only one slot whose associated expression which computes the value to be returned. PSN introduces the slot called "interval" to the class "metaslot". ("metaslot" may have slots because it is itself a class and metaclass.) The interval of a metaslot is an ordered pair of numbers which specify the minimum and maximum number of that kind of slot which may appear in a class. This pair is used by the add program of "CLASS" as the last operation: for each metaslot in each parent class it will fetch the instances of the metaslot in the newly created object and check that the number of such instances falls in the interval of the metaslot. The metaslot "slot" has an interval [0,*) where the * represents infinity meaning that any class may have any number of slots.

One can now discuss how the various aspects of programs can be represented as slots. All programs will be instances of the metaclass "PROGRAM" which has a metastructure describing the various parts. Hence there are metaslots called "parameters", "prerequisites", "body", and "returns". Any slots which are not instances of these metaslots may be used as local variables. The metaslots now provide the mechanism for distinguishing the purpose of the slots of a program. For example, if the interpreter needs the prerequisites of a program it can search the set of slots making a list of all those which are instances of the metaslot "prerequisites".

An earlier example showed the definition of the property "eval" which is used for associating expressions with slots. This definition is contained in the metaslot "action_slot" in the metaclass "PROGRAM". The various categories of executable slots acquire this property because

the metaslots "body", "prerequisites", and "returns" are subclasses of "action_slot", inheriting its structure. Two other special properties are defined for executable slots in the same way. These are the "exception" property which is an expression which may be evaluated to create an exception should evaluation of the slot fail, and an "exception_action" property which specifies exception handlers to be used should an exception be created.

In general a program may have an arbitrary number of statements in each class. Hence the intervals for the parameters, prerequisites, and body are [0,*). However, the "returns" slot must be unique, so the interval for "returns" is [1,1] so that a value to be returned is always computed.

## 6. Specialization

A primary reason for representing programs as classes is the consequent ability to specialize programs through the use of IS-A. This allows uniform treatment of the inheritance of property values between IS-A related classes. In general, the property values of a subclass should be IS-A descendants of the corresponding property values of the superclass. Since programs are property values of classes, the use of IS-A is clearly indicated.

The structural constraints imposed by IS-A on related programs are however not sufficient to insure that the four programs of a subclass act correctly with respect to IS-A. It is necessary to consider programs in terms of their side effects and the values returned.

A first consideration is the add program of a class. An object is made an instance of a class through the side effects of the add program of the class. If a subclass of this class has an add program whose net side effects include those of the original add program, any object of the subclass must surely be an instance of the superclass. Thus one constraint on the specialization of programs is that the net side effects of the specialization include the net side effects of the original.

The structural constraint of IS-A inheritance combined with the simultaneous execution of body slots goes a long way towards this goal. Any new slot added in inheritance may not undo the side effects of an inherited slot because one cannot be sure that the new action will be performed after the inherited action in any execution of the program. One cannot remove an object from a class before it has been made an instance of the class.

The net side effects of an inherited program can, however, exclude some of those inherited because there exists in PSN a way of causing forms to be executed sequentially. This is done through the use of the program "BEGIN". This program takes two arguments called "arg1" and "arg2" of types "OBJECT" (that is any object) and "FORM" respectively. Evaluation is simple: if the second parameter has as value the object "NULL_FORM", the program returns the value of the first parameter, otherwise it applies the interpreter to the value of "arg2". The value returned in the latter case is the result of the evaluation of the form. If a chain of forms calling "BEGIN" is formed with the first parameter of each form having as an "eval" property a form and the second parameter having as a "quote" property value the next subclass of "BEGIN", one has a chain of forms which will be executed sequentially. The object

```
(begin-chain
   INSTANCE-OF FORM PROGRAM
   IS-A BEGIN
   STRUCTURE
      (arg1
         INSTANCE-OF action_parameters
         PROPERTY-VALUES eval F1)
      (arg2
         INSTANCE-OF quote_parameters
         PROPERTY-VALUES
            quote
            (begin-chain2
               INSTANCE-OF FORM PROGRAM
               IS-A BEGIN
               STRUCTURE
                  (arg1
                     INSTANCE-OF
                     action_parameters
                     PROPERTY-VALUES eval F2)
                  (arg2
                     INSTANCE-OF
                     quote_parameters
                     PROPERTY-VALUES quote
                     NULL_FORM))))
```

for example, is a form which calls "BEGIN" in which the form "F1" will always be executed before the form "F2". This results because the execution sequence begins by the creation of an instance of "begin-chain" for which parameter values are supplied by evaluating the value of "arg1" thus evaluating "F1", and taking the value of "arg2" as is. "F2" is evaluated only when the interpreter is applied to "begin-chain2" after it is discovered that the value of "arg2" is not "NULL_FORM".

When inheriting such "BEGIN" structures, one may have a problem with side effects. Any form may be a subclass of "NULL_FORM" (because it has no structure). Thus one can modify an inherited "BEGIN" chain by adding more calls to "BEGIN" at the end because any form (eg. a call to "BEGIN") can be a subclass of "NULL_FORM". Thus the structural IS-A constraints will be satisfied. However these additional forms will always be executed after the inherited forms in the same "BEGIN" chain and therefore may undo the side effects intended by the inherited forms. For example, a valid IS-A descendant of "begin-chain" is represented by

```
(subclass-of-begin-chain
   INSTANCE-OF FORM PROGRAM
   IS-A begin-chain
   STRUCTURE
      (arg2
         INSTANCE-OF quote_parameters
         PROPERTY-VALUES
            quote
            (subclass-of-begin-chain2
               INSTANCE-OF FORM PROGRAM
               IS-A begin-chain2
               STRUCTURE
                  (arg2
                     INSTANCE-OF
                     quote_parameters
                     PROPERTY-VALUES
                        quote F3).
```

The IS-A constraints are satisfied in "subclass-of-begin-chain2" because the only change is a modification to a property value of "arg2" in which the new value, "F3", is an IS-A descendant of the old value. The new form "F3" will always be executed after "F1" and "F2" and therefore may with certainty undo some side effects performed by the original forms.

The PSN solution to this problem is to provide an additional constraint on the use of IS-A for specializing programs. IS-A may hold between two programs only if in any possible knowledge base the set of net side effects produced by the specialization must contain the set of net side effects which would be produced by the IS-A parent if run with the same parameters.

A final constraint on the use of IS-A for programs results from consideration of the values returned by the programs. In particular, the results of test programs are interesting. If class "B" is a specialization of class "A", and "Pb" and "Pa" are the respective test programs, one finds that if "Pb" returns true, "Pa" must return true because an instance of "B" is always an instance of "A". However, "Pa" may return true for an instance of "A" which is not an instance of "B". The relation between the values returned is logical implication: the value returned by "Pb" implies the value returned by "Pa". Now nothing in the structure of programs obviously relates the values returned. Therefore, a program "P2" may only be a specialization of "P1" if, when returning Boolean values under identical conditions, the result of "P2" implies that of "P1" and when not returning Boolean values, the results of the two programs under identical conditions are identical.

The two extra conditions on programs cannot be tested in a knowledge base system. However, if an effort is made to minimize the changes to a "returns" slot and to avoid adding statements causing side effects to "BEGIN" chains, it should be possible to write programs for which IS-A may properly be asserted.

## 7. Conclusion

The representation of programs as objects in the PSN formalism has several useful consequences. Most important is the interaction with the IS-A hierarchy. Programs themselves participate in a strictly controlled IS-A relationship which constrains the programs which play the various roles in the definition of a class in a way that insures that classes related by IS-A behave in the appropriate manner. In addition, the inheritance resulting from IS-A and the separation of programs into separately inheritable statements provide a powerful and flexible programming tool. In specializing programs one may add and modify parts of a program and automatically inherit the unchanged parts.

The representation of programs as objects within the formalism allows the manipulation of programs by other programs. One can therefore write programs which create or modify programs, and, as in LISP, one can write an interpreter for the formalism as such a program. The explicit representation of program activations provides similar flexibility. Programs may themselves alter the flow of control to produce back tracking and concurrent processes through manipulation of the assertions of the relation "dynamic".

A final result is the introduction of metastructure. Metastructure provides a means of organizing and constraining the slots in the structures of classes. This has found immediate use in organizing the slots of a program to provide the different functional divisions. It is anticipated that metastructure may find uses in other representation formalisms.

## 8. Acknowledgements

I would like to thank Professor John Mylopoulos for his efforts in the supervision of this work. I would also like to thank Hector Levesque, Alex Borgida, Peter Schneider, and Yves Lesperance for their valuable comments.

Financial support was received from the Natural Sciences and Engineering Research Council of Canada, the Government of Ontario, and the Department of Computer Science at the University of Toronto.

## 9. Bibliography

[Kramer 1979] Kramer, Bryan M. "Representation of Programs" in "Topics in a Procedural Semantic Network Formalism." AI-MEMO 79-3, Department of Computer Science, University of Toronto, June 1979.

[Kramer 1980] Kramer, Bryan M. "The Representation of Programs in the Procedural Semantic Network Formalism." Technical Report No. 139, Department of Computer Science, University of Toronto, 1980.

[Lesperance 1980] Lesperance, Y. "Handling Exceptions in the PSN Formalism," M.Sc. thesis, Department of Computer Science, University of Toronto, to appear.

[Levesque 1977] Levesque, H. "A Procedural Approach to Semantic Networks." Technical Report No. 105, Department of Computer Science, University of Toronto, April 1977.

[Levesque and Mylopoulos 1979] Levesque, H., Mylopoulos, J. "A Procedural Semantics for Semantic Networks" in Associative Networks: Representation and Use of Knowledge by Computers. Findler, N. V.(ed). Associated Press, New York, 1979.

[Mylopoulos et al. 1978] Mylopoulos, J., Bernstein, P. A., Wong, K. T. "A Language Facility for Designing Interactive Database-Intensive Applications." Paper presented at SIGMOD conference, Austin Texas, May 1978. (to appear in TODS). Also appears in CSRG Technical Report No. 105 (= AI-MEMO 79-4), University of Toronto, July 1979.

[Schneider 1978a] Schneider, P. F. "Organization of Knowledge in a Procedural Semantic Network Formalism." Technical Report No. 115, Department of Computer Science, University of Toronto, February 1978.

[Schneider 1978b] Schneider, P. F. "Organization of Knowledge for a Procedural Semantic Network Formalism" in the Proceedings of the Second National Conference of the Canadian Society for the Computational Studies of Intelligence, Toronto, July 1978.

[Wong 1980] Wong, H. Design and Verification of Interactive Information Systems, Ph.D. thesis, Department of Computer Science, University of Toronto, to appear.

# ORGANIZATION OF MODALLY EMBEDDED PROPOSITIONS AND OF DEPENDENT CONCEPTS

Alan R. Covington and Lenhart K. Schubert

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

Abstract

This paper is concerned with content-oriented retrieval of information from a potentially very large semantic net, such as would be needed to support a natural language system with an unrestricted discourse domain. Specifically, the paper focuses on three issues. The first is the organization of propositions embedded within modal operators such as propositional attitudes and story-operators. A subnet structure is described which permits recursive embedding of various "conceptions of the world", and a previously developed topical access mechanism is extended to operate within this structure. The second is the associative accessing of concepts within subnets according to their type. The third is the design of data structures and mechanisms for the inheritance of parts relationships from generic to (more) particular concepts. The proposed methods have been implemented and their effectiveness demonstrated with the aid of a query system.

## 1. Introduction

Our long-range objective is the design of an English conversational system with a theoretically unlimited domain of discourse. Such a system must be able to store a very large knowledge base, and to use the stored knowledge without succumbing to combinatorial catastrophe during routine inferencing in support of language comprehension, consistency checking, and simple question answering.

Consequently we have concentrated our recent efforts on the design of a semantic net organization permitting fast, content-directed insertion and access of concepts and propositions in an arbitrarily large net. The methods we have developed are more than "book-keeping" methods. Because the insertion and retrieval mechanisms are sensitive to conceptual content and utilize an inheritance scheme, they rapidly establish logical connections which would ordinarily require search and nontrivial inference. Thus they should go a long way towards "keeping the lid on" combinatorial explosions.

In previous work on our system, Goebel (1977) implemented a net structure with the expressive power of higher-order modal logic, and added taxonomic structures and algorithms for topic-oriented insertion and retrieval of propositions. Traversal of "topic access skeletons" enabled selective retrieval of exactly those propositions about a given concept (such as "Clyde" or "zebra") which pertain to a more or less specific topic (such as "colouring" or "appearance"). An earlier version of the system, without a topical access structure but with a simple English front end had been built by Cercone (1975). These connected efforts are motivated, summarized and extended conceptually in Schubert et al. (1979). The recent changes and additions to the system are fully described in Covington (1980).

The first of the following sections (Sec. 2) outlines the net formalism and the modal clause form in which propositions are currently represented. Maximal "structure sharing" is used to economize storage and to simplify the recognition of logical relationships (especially equivalence) among different sentences.

The original semantic net system permitted fast topical retrieval of the system's "knowledge about the world", but not of propositions belonging to alternative conceptions of the world, such as those making up some person's "mental world", or those making up a fairy tale. This was not due to any lack of expressive power - modal propositions were readily represented - but rather to a lack of organizing principles for grouping and accessing modally embedded propositions. Sec. 3 describes the first part of the solution to this problem. All the propositions which make up a particular individual's (or particular story's) conception of the world, such as Alice's beliefs, hopes, intentions, etc., are placed in a subnet. Subnets, like the main net, have logical dictionaries and contain "virtual concepts" as access points to the propositions making up their conception of the world. They may be recursively nested, and the insertion routines ensure that clusters of propositions such as those describing Tom's conception of Alice's mental world are organized as coherent subnets.

Sec. 4 then describes the solution of the generalized retrieval problem, based on the use of topic access skeletons within subnets and on extensions of the topical classification and accessing algorithms to process nested modal sentences. A greatly improved "descendant bracket" representation is used within the current implementation of topic hierarchies, reducing time and storage requirements.

Topical accessing solves the problem of finding the propositional knowledge which is immediately relevant to a question (or other task), given the conceptual referents of the question. However, question-answering (and problem-solving) often requires the "inverse" of this type of access, called associative access: concepts must be found on the basis of certain given propositions about them. An efficient method for one important kind of associative accessing is described in Sec. 5. The method uses "concept access skeletons" attached to subnets and implemented in much the same way as topic access skeletons.

Both topical accessing and concept accessing can be regarded as weak but fundamentally important kinds of inference. The former answers questions of the form "What properties of such-and-such a type does object (or predicate) x have?", and the latter questions of the form "What objects within a given conception of the world are of such-and-such a type?" A third and somewhat stronger kind of inference which also permeates all forms of cognition is property inheritance, or more generally, relationship inheritance. This involves the transfer of the properties of a generic concept (such as "bird") and of relationships among entities functionally dependent on that concept (such as the parts of a bird) to a particularization of that concept (such as a generic or a particular robin and its parts). Previously proposed mechanisms for relationship inheritance suffer from logical and technical difficulties. These are overcome in an approach described in Sec. 6, based on indexing the nodes (variables) functionally dependent on a given node (variable) in a function table attached to that node. Relationship inheritance is then obtained by table look-up for concepts in a type hierarchy (or lattice).

* This research was supported in part by Operating Grant A8818 of the Natural Sciences and Engineering Research Council of Canada.

Sec. 7 contains a sketch of a query system which serves to demonstrate the retrieval and inheritance mechanisms based on the network organization. The concluding section suggests extensions of the subnet structure, concept accessing mechanism and function table organization, as well as other directions for further work.

## 2. Net logic and normal form

We use semantic net terminology in referring to our propositional representation without, however, attaching a great deal of significance to that terminology. Essentially, the network syntax provides for the representation of formulas in higher-order modal logic, with constants, functions, existentially and universally quantified variables, and the usual truth-functional connectives. Of the modal operators, only the necessity operator □ is predefined, propositional attitude operators (such as "hopes" and "believes") and other modal operators being treated as higher-order predicates.

Semantic nets are computer-oriented representations of sets of propositional formulas. They are designed to show the access paths from propositions to their participating concepts and from concepts to the propositions about them explicitly. We call these paths "forward links" and "back links" respectively. Examples of propositions expressed in the network representation of Schubert (1976) are shown in Figs. 1-3.

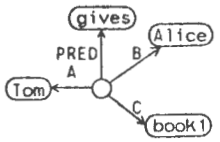



Fig. 1. "Tom gives Alice the book"
[Tom gives Alice book1]

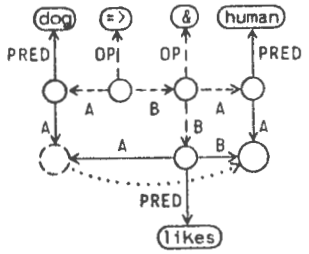



Fig. 2. "Every dog likes some human"
∀x[[x dog]=>[[y human]&[x likes y]]]
The broken circle indicates universal quantification, and the dotted arrow quantifier precedence (scope inclusion).

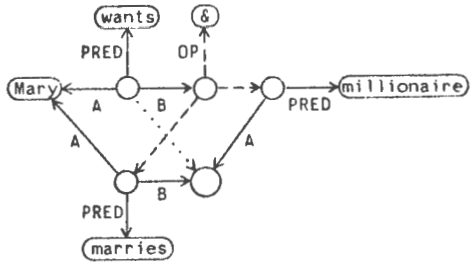



Fig. 3. "Mary wants to marry a millionaire"
[Mary wants ‡x[[Mary marries x]&[x millionaire]]]
The dotted arrow indicates operator-quantifier precedence, and would be missing in the transparent reading, "There is a millionaire whom Mary wants to marry".

No explicit back links are shown; these are easily visualized as the inverses of the forward links. Back links could be implemented as linear lists of pointers, but in our implementation the pointers are attached to tree-structured "access skeletons" (Sec. 4). The graphical representation is derived from the network formalisms of Quillian (1968), Shapiro (1971), and Rumelhart et al. (1972). Its correspondence to standard predicate calculus notation is particularly clear. Some of the features illustrated in Figs. 1-3 are n-ary predication, with n>2 (Fig. 1), quantification and scope inclusion (Figs. 2 & 3), and modal predication (Fig. 3). The figure captions give standard predicate calculus renditions of the sample sentences. However, note that we are using infix notation: a sentential formula is a list in square brackets headed by the first argument of the sentence predicate or operator, followed by the predicate or operator symbol, followed by any additional arguments.

The representation logic also provides a special syntax for functions and for time. The lexical notation for functions is LISP-like. For example,
[(height-in-cm John) = 182].
states that the height in cm of John is 182. The corresponding graphical notation is described in Schubert et al. (1979). The lexical and graphical notation for moments or intervals of time can also be found there.

In order for a system to be able to reason in a human-like fashion, it must be able to represent belief strengths (e.g., see Colby et al., 1969, Schank & Rieger, 1974). We currently attribute degrees of belief to the system and to other entities by means of explicit propositions such as

[[John loves Mary] has-credibility .9],

[Mary believes [John loves Mary] d]
& [d exceeds .5],

where d represents a numerical degree of belief. We could equally well use non-numerical degrees of belief. Hopefully, degrees of belief can be interpreted within a formal framework such as that of Moore (1979).

In the following discussion of propositional normal form and on many subsequent occasions, we will make reference to "type predicates". By this we mean predicates of the kind commonly used in taxonomies of entities, such as "human being", "tree", "nation", "poem", etc. Examples of non-type predicates might be "grey", "happy", "villain", etc. Though somewhat arbitrary, the distinction is nevertheless very useful.

In the early stages of our work on the network system, it was unclear whether internally stored propositions should be unrestricted in form, with arbitrary quantifier and operator embedding, or should be cast in some normal form. We are now firmly committed to normalization, because normalization simplifies topical classification (Sec. 4) and pattern matching (such as that required for property inheritance, Sec. 6) and quickly reveals many logical equivalences which would otherwise have to be inferred.

Uncovering logical equivalences is important for storage economization as well as inference. For example, in a knowledge base containing both

[Eve loves Tom]|[Ann loves Tom]

(where | is the logical "or") and

[Tom hopes [[Eve loves Tom]|[Ann loves Tom]]],

the shared formula need be stored only once; the second proposition would simply point to this formula. This would not be the case if the first formula were represented equivalently as

¬[Ann loves Tom]=>[Eve loves Tom].

Furthermore, this latter representation would make it harder to establish that Tom's hopes were fulfilled. Our revised implementation makes maximum use of subformula sharing, maintaining a formula hash table for determining whether a given subformula is already in the semantic net. (Subformulas which are the same except for the variable nodes they reference are not treated as identical; ways to minimize the number of distinct variable nodes and hence maximize subformula sharing are mentioned below.)

A particularly convenient normal form for the purposes of topical classification is clause form, in which formulas are reduced to sets of purely disjunctive clauses with implicit quantification. In Schubert et al. (1979) a kind of implicative normal form quite similar to clause form was proposed. However, universally and existentially quantified variables were to be shared within type hierarchies to facilitate property inheritance. This scheme has proved to be logically flawed and unsuitable with regard to topical accessing (see Schubert, 1979 and Covington, 1980).

We have therefore chosen a normal form which is closer to clause form than the originally proposed form. In fact, the only difference from clause form (for non-modal propositions) is that universally quantified variables are shared among generalizations about entities of the same type. For example, all clauses containing ¬[x robin] among their disjuncts, where x is some universally quantified variable, use the same universally quantified node for x. Only one modification of this rule is required, for clauses such as

¬[x elephant]|¬[y elephant]|[x likes y]

(i.e., elephants like themselves and each other), containing two or more negated type predications with identical type predicates but distinct variables. In such cases one of the variables is arbitrarily chosen as the variable (node) to be shared with all other generalizations about entities of that type, while the remaining variables (nodes) are kept distinct from the shared variable.

No special rules are required for clauses involving distinct type predicates, such as

¬[x elephant]|¬[y mouse]|[x afraid-of y].

Here the shared "elephant variable" would be used for x and the shared "mouse variable" for y. By thus minimizing the number of distinct universally quantified variables, we conflate numerous type predications, saving storage and facilitating inference.

We have extended the normalization procedure to apply to modal propositions in the following sort of way. Consider the proposition

[Ann hopes ∃x[[x car]&[Tom owns x]]

(i.e., Ann hopes that Tom owns a car). To convert this proposition to "modal clause form", we first replace x by a new constant c embedded in the context [Ann hopes...]. Symbolizing this embedding relation is a problem in the lexical syntax, but not in the network syntax. As in Fig. 3, we simply run a scope inclusion link from the embedding proposition to the embedded constant. Note that omission of the link would yield the nonequivalent proposition that there is a specific car which Ann hopes Tom owns. Next we separate the modal proposition into a pair of modal propositions, each containing "half" of the embedded conjunction. The result could be written as

[Ann hopes [c car]], [Ann hopes [Tom owns c]],

if it were not for the fact that c is embedded within the context [Ann hopes...]. The only correct lexical representation of these propositions is the original compound proposition with an explicit existential quantifier. (A lexical syntax capable of representing modal clause form could be designed, but we haven't done so.)

In general, a sentence whose top-level operator is modal (with one sentential argument) is converted to modal clause form by recursively converting the embedded sentential formula to modal clause form, inserting scope inclusion links from the top-level sentence node to universal and existential nodes whose quantifiers lie within the modal context, and distributing the modal operator over the embedded clauses. Modal clause form prepares the way for the recursive net organization described in Secs. 3 & 4, in which the tabular and taxonomic organizing structures used to access top-level propositions are carried over into modal contexts.

## 3. Modal subnets

The modal predicates we have chosen to consider are those which take an individual as first argument and a proposition as second argument. This format appears to accommodate not only propositional attitudes but also stories. For example, the story of Cinderella might be represented in the form

[c is-a-story-in-which ∃u∃v∃w...[[u girl]&
[v stepmother-of u]&[w pumpkin]&...]],

where the story itself is treated as an individual which is related to its propositional content by the modal relation "is-a-story-in-which". The treatment of stories as individuals also allows other kinds of statements to be made about them, such as statements about their name and origin.

To understand what goes on in people's minds and in stories, an understanding system must be able to access and manipulate modally embedded propositions in much the same way as top-level propositions. We have therefore extended our network system by introducing subnets, which provide access to alternative conceptions of the world and may be recursively nested. Similar kinds of subnet organization were previously proposed by Hendrix (1975, 1979) and Cohen & Perrault (1976), among others. What is distinctive about our subnet organization is the principle according to which propositions are collected into subnets (discussed below) and the topical and associative access structures supported by subnets (discussed in Secs. 4 & 5 respectively).

Automatic creation and maintenance of subnets requires computable criteria for deciding which modal propositions "belong together". Our initial inclination was to associate a distinct subnet with each modal context, such as "John believes...", "John wants...", "Mary hopes...", "In the story of Snow White...", etc. The modal context would then be specified uniquely for each subnet and would not have to be repeated for each proposition within the subnet. However, this advantage is offset by the space inefficiency of having a separate subnet for each mental attitude of each individual, with its own dictionary and concept access skeleton (Sec. 5). Most importantly the beliefs, hopes, wants, dreams, etc., of an individual are so closely bound up with each other that they should be accessible together. For example, the answer to the question why John plans to paint the fence may be that he believes its present colour to be ugly, or that he wants it to be white. It should not be necessary to jump from modal subnet to modal subnet to collect the information required about a particular concept, such as the information about the appearance of the fence in John's mental world.

Thus we have chosen to associate at most one modal subnet with each individual concept in the main net and, recursively, in each subnet. Propositions belonging to a subnet are stored in

full; for example, the modal context "John believes that ..." is repeated for each of John's beliefs (however, the associated degrees of belief are in general different).

Subnets differ from the main net in one important respect: their nodes are "virtual nodes" serving exclusively as knowledge access portals, never as propositional constituents. For example, John's belief that Mary loves him would be stored as a main net proposition accessible from the virtual nodes for John and Mary in the subnet of John's mental world, as well as from the node for John in the main net. Since all propositions are stored in the main net, subnets do not affect structure sharing. For example, since all the nodes participating in the propositions

¬[Mary loves John], [John wants [Mary loves John]]

belong to the main net, the common subproposition can be shared between them.

Like the main net, subnets contain node dictionaries. More importantly, subnets and their virtual nodes hold the same kinds of taxonomic access structures as the main net and its nodes. These are described in the next two sections.

## 4. Topical classification and accessing

When a very large, heterogeneous knowledge base is used to support language understanding or problem solving, it is essential that retrieval be highly selective. For example, a question answering system confronted with the question

?∃x[Clyde afraid-of x]

("Is Clyde afraid of anything?") would be courting computational disaster if it treated all of its knowledge about Clyde and about being afraid as immediately relevant to the question. Facts about Clyde's appearance, food preferences, pastimes, etc., are unlikely to be helpful, as are the majority of known instances of one thing being afraid of another, such as Myrtle's being afraid of spiders. The system would do better to confine its attention, at least initially, to its knowledge (if any) about Clyde's fears, or if that is unhelpful, about Clyde's emotional attitudes and dispositions in general, or if that is still unhelpful, about these sorts of properties as they pertain to members of Clyde's kind (type). This topic-specific information could be combined with general knowledge about being afraid (e.g., that if x is afraid of y, then x will try to avoid y) in trying to infer an answer to the question.

These kinds of examples suggest the need for a topical classification of the knowledge available about each concept. This need is also indicated by questions which are explicitly topical, e.g., "What does Clyde look like?", or "What do you know about Clyde's emotional make-up?". Further motivating examples can be found in Goebel (1977) and Schubert et al. (1979). Reder & Anderson (1979) offer some psychological evidence for a topical (thematic) organization of knowledge about individual concepts, although they appear to have in mind a single-level taxonomy. Rychener (1979) proposes a single-level taxonomy for knowledge associated with concepts in a computer-aided design system.

To be of any use, a topical organization must meet three requirements: the topical taxonomy on which it is based must be coherent in the sense that it brings closely related topics (i.e., those likely to be relevant to the same sorts of questions or problems) into close proximity; the topical category or categories of any sentence must be readily computable ; and topic-specific knowledge about a given concept must be readily accessible.

Schubert et al. (1979) supplied a fairly comprehensive and intuitively coherent taxonomy of knowledge about physical objects (with emphasis on static properties as opposed to behaviour). The taxonomy is hierarchic, apart from some minor violations of hierarchy which are easily eliminated. A fragment of the hierarchy is shown in Fig. 4. The pairs of numbers attached to the topic nodes are explained below.

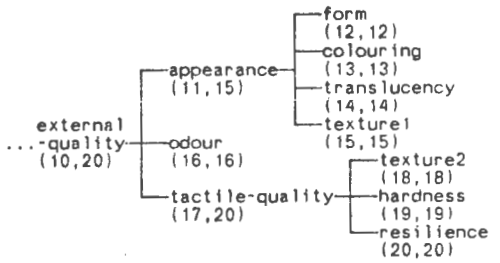



Fig. 4. Topic hierarchy fragment

The requirement that the topical categories of arbitrary sentences be effectively computable is readily met for the sample hierarchy. Some basic classification mechanisms were reported in Goebel (1977), and extensions of these to deal more adequately with a wider range of sentential forms were proposed in Schubert et al. (1979). These have now been implemented, with generalizations to allow for nested modal sentences.

The classification algorithm assigns clauses to topics relative to each of the predicates and existentially quantified variables occurring in them. For example, the clause

¬[x spider]|[Myrtle afraid-of x]

(Myrtle is afraid of spiders) might be classified as an "emotional attitude" proposition relative to Myrtle, and as an "emotional effect" proposition relative to "spider". The classification algorithm relies on "indicator links" attached to predicate nodes, on the position and quantification of the arguments of the clause, and on the signs (negation or none) of the predicates used in the clause. For example, the topical categories of the above clause are based on indicator links from "afraid-of" to "emotional attitude" and "emotional effect", on the position of "Myrtle" in the "afraid-of" predication, and on the negation of "spider" and the universal quantification of its argument. For details of the classification algorithm see Schubert et al. (1979) and Covington (1980).

One difference between the current algorithm and earlier versions is that modal propositions are appropriately classified as beliefs, goals, narrative assertions, imperatives, etc., assuming that the requisite indicators have been supplied. More importantly, the new algorithm recursively classifies the clauses modally embedded within the input clause. Insertion and access within a particular subnet uses the classification at the appropriate level of modal embedding. Modal operators such as necessity, credibility, and causation are ignored at present, i.e., the topic categories of the embedded sentence are transferred to the embedding sentence.

The third requirement for a topical organization, that propositions involving a particular concept and pertaining to a particular topic be readily accessible, may seem hardest to meet. Note that the problem is not to taxonomize the knowledge stored in the semantic net as a whole, but to do this for every concept. The storage requirements of such a scheme could be prodigious.

The topical organization developed by Goebel (1977), Schubert et al. (1979) and Covington (1980) does provide concept-centred topical access, yet

avoids excessive storage costs. The data structures used include a representation of the general topic hierarchy in the main net and tree-structured topic access skeletons attached to all concept nodes (other than universally quantified nodes) within the main net and all subnets. The access skeletons can be thought of as the smallest fragments of the general topic hierarchy needed to taxonomize the propositions actually available about each concept.

An example will clarify the method. Suppose that the topic hierarchy includes the fragment previously shown in Fig. 4, and that the semantic net contains propositions to the effect that a ball is round and resilient. The two propositions, say p and q, would be classified as "form" and "resilience" propositions relative to "ball". If these are the only "external-quality" propositions about "ball", then the part of the access skeleton of "ball" corresponding to the hierarchy fragment of Fig. 4 would have the form shown in Fig. 5.

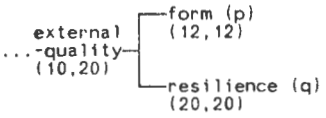



Fig. 5. Path-contracted access skeleton fragment corresponding to Fig. 4.

Note that only paths leading to propositions have been retained in the access skeleton. Furthermore, linear path segments have been contracted, so that every non-terminal node has at least two direct descendants; in particular, the "appearance" and "tactile-quality" nodes have been eliminated by path contraction. It is the pruning and contracting of paths which makes possible concept-centred, topic-oriented accessing of knowledge with modest storage overhead.

A version of path contraction was proposed in Schubert et al. (1979) and was shown to yield access skeletons whose storage cost is only twice that of the unstructured linear back-link lists which they supplant. The current implementation improves on that proposal by retaining the lowest instead of the highest node of a contracted path, thus avoiding the need to recompute the classification of accessed propositions.

A more important innovation in the new version of the topical organization is the use of pairs of numbers to characterize nodes in the general topic hierarchy, as indicated in Fig. 4. These are assigned and used as follows. The first of each pair of numbers, called the identification number (IN) of the corresponding topic, is the number assigned by a pre-order (i.e., depth-first) numbering of the topic hierarchy. The second number, the "highest descendant" (HD), is the maximal IN among descendants of that node. Then all descendants of a node have identification numbers within the "descendant bracket" [IN, HD] determined by the pair of numbers at that node. The topic access skeleton nodes are not explicitly numbered, but contain pointers to the topic hierarchy nodes they represent.

Without descendant brackets, access to propositions subsumed under a particular topic within a particular topic access skeleton required an initial ascent in the general topic hierarchy and involved complications due to path contraction. With descendant brackets, the initial step is merely a look-up of the IN of the desired topic; this is followed by a descent in the access skeleton such that the descendant bracket of the node selected at each step contains the desired IN. For example, the "external-quality" branch in Fig. 5 would be chosen when accessing the "form" proposition p because the IN of "form" is 12 and this lies within the bracket [10,20] of "external-quality". The design of efficient algorithms for expanding access skeletons when inserting new propositions, and for finding all propositions subsumed under a topic access skeleton node deleted by path contraction, has also proved to be quite straightforward; for details see Covington (1980).

As we have emphasized, the "virtual" nodes of subnets serve exclusively as knowledge access portals. More specifically, each virtual node points to the root of a topic access skeleton. The propositions at the terminals of an access skeleton are "about" the concept referenced by the virtual node, at the level of modal embedding appropriate to the level of the subnet. In this way subnets provide concept-centred, topic-oriented access to propositions decribing alternative conceptions of the world. That is half of their function; we now turn to the other half.

## 5. Concept accessing

People can readily recall objects associated with stories they know, given some of their key properties. What animals are there in "Little Red Riding Hood", what vehicles in "Cinderella", what computers in "2001"? The identification of entities with specified properties is often called associative access (or retrieval). It is of direct use in answering questions such as those just given, and may also be important in determining remote referents of noun phrases in story understanding. (For example, the central narrative in Hemingway's The Old Man and the Sea contains widely separated references to "the boy", who is nevertheless easily identified as the old man's young friend Manolin.)

The methods usually suggested for implementing associative accessing without special hardware involve enumerating all known entities with the desired properties and checking whether they belong to the current context, (e.g., Scragg, 1975, Hayes, 1977), or enumerating all entities in the current context and checking whether they have the desired properties (e.g., Brown & Burton, 1975, Hobbs, 1975). The latter type of method seems more natural and potentially more efficient than the former; however, efficiency depends on the search context being small, and this is not always the case. For example, the search for "vehicles" in the Cinderella story is likely to be quite time-consuming if implemented through exhaustive testing of all entities mentioned in the story. PLANNER (Hewitt, 1971) permits pattern-directed retrieval of assertions, but still relies on enumerative search if no explicitly matching assertions exist in the data base.

We have implemented a non-enumerative method of accessing instances of any given type of concept within a story subnet, or any other kind of subnet. The method uses concept access skeletons, first proposed for this purpose in Schubert et al. (1978). However, the proposal there was to attach concept access skeletons to all concepts, which we now believe to be both impractical and unnecessary. Instead, concept access skeletons are associated with subnets. Each is a tree which echoes fragments of a type (generalization, IS-A) hierarchy and whose nodes point to clauses containing unnegated type predications at the appropriate level of embedding. For example, the subnet for the story of Little Red Riding Hood would contain a concept access skeleton with a "wolf" node pointing to the clause

[LRRH is-a-story-in-which [w wolf]],

where w is a "modally embedded constant" as described in Sec. 2.

The construction and use of concept access skeletons is entirely analogous to that of topic access skeletons. A general type hierarchy, analogous to the general topic hierarchy (and in fact structurally united with it in the current implementation), resides in the main net. In effect this hierarchy duplicates the collection of main net

predications of the form [c P] or [[x P] => [x Q]], where P and Q are type predicates, but uses a simplified representation geared toward efficient concept accessing. Like the topic hierarchy nodes, the type hierarchy nodes are labelled with numeric descendant brackets. The concept access skeleton of a subnet contains the paths needed to access the type propositions relevant to that subnet, where these paths have again been contracted to eliminate linear segments. As in the case of topic access skeletons, access to a proposition via a concept access skeleton requires only a look-up of the identification number of the desired type concept, followed by descent in the appropriate concept access skeleton with the aid of the descendant brackets associated (via pointers to the general type hierarchy) with access skeleton nodes. Thus descent towards "animal" propositions within the story net for Little Red Riding Hood, for example, would lead directly to the "wolf" proposition, even though the wolf has not been explicitly described as an animal in the story. This is because the identification number of "wolf" is included in the descendant bracket of "animal".

## 6. Property and relationship inheritance

A system possessing knowledge about entities in a taxonomy of types will often have to combine knowledge about a given entity with knowledge inherited from higher-level entities. For example, in a system knowledgeable about birds in general and owls in particular, the fact that an owl's beak is curved might be an explicit piece of owl-knowledge, but the fact that the owl can use its beak to seize food is more likely to be implicit in its bird-knowledge. To interpret both properties as properties of the same beak, the system must recognize the correspondence between the bird's beak and the owl's beak.

As already mentioned in Sec. 2, a "variable-sharing" method proposed by Hayes (1977) and Schubert et al. (1979) has proved unworkable. A logically and technically satisfactory alternative is to use parts functions. For example, if f is a function which picks out a bird's beak, then bird knowledge and owl knowledge will take the form

¬[x bird]|[(f x) beak-of x], ...

¬[y owl]|[(f y) curved], ...,

where x and y are the primary variables (nodes) associated with "bird" and "owl" respectively (see Sec. 2). Establishing the correspondence between (f x) and (f y), once x and y have been matched, is then just a matter of locating the nodes representing the values of f for argument nodes x and y.

In Schubert (1979) a "function table" scheme was suggested for locating values of functions quickly. This scheme has now been implemented. The "scope inclusion" field of any (non-virtual) node may contain a pointer to a hash table. The table is indexed by function name, and the value tabulated for a given function name is a pointer to the node representing the value of that function applied to the concept to which the table is attached. Limited provision has also been made for inverse access from a functionally dependent node to the node on which it depends; however, the inverse pointer is available for only one functional dependence of the node, and only if the node does not itself have functionally dependent nodes.

With the help of function tables, properties and relationships are easily mapped downward to a concept and its parts from more general concepts, irrespective of the number of intervening concept levels. For example, the fact that Clyde's head joins his neck is easily obtained from the "animal" concept, using functions which yield those parts; this is no more difficult than obtaining a property of his proboscis by referring to elephant knowledge, even though "animal" might lie several levels higher than "elephant".

However, a requirement in both of these examples is that the names of the functions determining Clyde's head, neck, proboscis, etc., be known and usable as keys in Clyde's function table. In the current implementation this information is supplied "by hand", using utility routines for inserting entries in function tables and setting up inverse dependency pointers. For statements of the form ¥x∃y[...] the Skolem function(s) introduced for the existentially quantified variable(s) are automatically inserted in the function table(s) of the appropriate universally quantified variable(s). But again hand-coding is needed to make the Skolem functions introduced by two such statements identical, unless the statements have been supplied as an explicit conjunction sharing the same universally quantified variable. In other words, we have not attempted to deal here with the general problem of automatically inferring that a node is a value of some function already known to the system. (For some proposals with regard to this problem see Schubert, 1979.)

The use of function tables has been integrated with the topical accessing mechanism in such a way that topic-oriented retrieval of knowledge about a (given part of a) given concept will automatically "pull down" relevant information from concepts dominating the given concept in the taxonomy of types. This combination of topical retrieval and property inheritance is illustrated in the next section.

## 7. Querying the semantic net

In addition to the structure building routines, the system implementation includes a limited query language which can be used to demonstrate topical access, concept access, and property inheritance. Some examples of queries follow, with rough English translations and with the responses they might elicit, assuming that the requisite knowledge has been placed in the semantic net.

```
(1)  ?[Clyde animal]
     "Is Clyde an animal?"
      Yes.

(2)  ?s m[Clyde ?p{tp.appearance}]
     "What do you know about Clyde's appearance?"
      [Clyde handsome], [Clyde big], [Clyde grey].

(3)  ?m[Clyde likes ?x],
     "What (or whom) does Clyde like?"
      [Clyde likes John], [Clyde likes Dumbo].

(4)  ?m[?x ?p{tp.gen:elephant}],
     "What elephants do you know?"
      Clyde, Dumbo.

(5)  ?s m[John believes [Clyde
      ?p{tp.appearance}]],
     "What does John believe about Clyde's
      appearance?"
      [John believes [Clyde ugly]],
      [John believes [Clyde pink]].

(6)  ?s m[LRRH is-a-story-in-which
      [?x ?p{tp.gen:animal}]].
     "What animals are there in the story of Little
      Red Riding Hood?"
      [LRRH is-a-story-in-which [w wolf]].
```

Queries are propositional schemas preceded by a question mark and possibly the flags s or m. A propositional schema is either an atomic proposition with constant arguments (as in (1)), or such a proposition with its predicate and/or some of its arguments replaced by query variables (as in (2)-(4)), or another propositional schema embedded within an unquantified modal predication (as in (5) & (6)). Query variables are prefixed with ?; they match any node unless followed by a topic or concept

restriction enclosed in braces. Topic restrictions take the form tp.<topic> and concept restrictions the form tp.gen:<concept>.

The response to (1) is obtained by accessing generalization (i.e., type) predications about Clyde and if necessary, about concepts which generalize Clyde. The response to (2) lists the monadic predications attached to the "appearance" node and its descendants in the topic access skeletons of Clyde and the concepts which generalize Clyde. If the s (subhierarchy) flag were missing from query (2), only the nonspecific appearance propositions directly attached to "appearance" nodes in the appropriate topic access skeletons would be accessed. Thus only the first of the propositions in the response to (2) might be returned, assuming that the others are attached to the appearance subtopics "size" and "colouring". If the m (many) flag were missing, the search would terminate with the first matching proposition found. The response to (3) is found via the topic access skeleton of Clyde or concepts which generalize Clyde. The response to (4) is obtained via the concept access skeleton of the main net. The response to (5) is obtained much as for (2), except that the topic access skeletons and generalization propositions utilized are those belonging to the subnet associated with John. The response to (6) is obtained via the concept access skeleton of the LRRH subnet. (w is a constant embedded in the modal context [LRRH is-a-story-in-which ...].)

All of the previous examples may involve property inheritance of a simple sort. The proposition [Clyde grey] in the response to query (2), for example, may be obtained by ascent to "elephant" (a generalization of Clyde), access of appearance propositions about this concept, and matching of the query against the accessed proposition

¬[x elephant]][x grey].

Property and relationship inheritance for parts, which makes use of function tables, is illustrated by the following queries and responses.

(7) ?m[Cneck connected-to ?x],
"What is Clyde's neck connected to?"
[Cneck connected-to Ctorso],
[Cneck connected-to Chead].

(8) ?s m[Ctrunk ?p{tp.appearance}]
"What do you know about the appearance of Clyde's trunk?"
[Ctrunk long], [Ctrunk tapered].

(9) ?s m[Ctrunk ?p{tp.phys-rel} Cmouth]
"What is the relationship between Clyde's trunk and his mouth?"
[Ctrunk in-front-of Cmouth],
[Ctrunk next-to Cmouth].

Here it is assumed that Cneck, Ctrunk, etc., are constant nodes already existing in the net and referenced in a function table attached to the node for Clyde.

The processing of query (7) begins with a topical access to physical relationships about Cneck and an attempt to match the accessed clauses to the query. Then a check is made for functional dependency of Cneck, yielding its functional dependence on Clyde. The function found is looked up in the function tables of the generalizations of Clyde, and the nodes thus located are examined for further physical relationships matching the query. Thus the response to (7) may be based on knowledge about Clyde and/or knowledge about generalizations of Clyde. In much the same way responses to (8) and (9) may be based on knowledge reached at several levels of generalization with the aid of function tables.

Finally we should mention that topic-oriented and concept-oriented retrieval of time-dependent information, such as the events of a story, presents no special problems. For examples see Covington (1980).

## 8. Concluding remarks

We have described knowledge organizing principles which permit fast topical and associative retrieval of knowledge from an arbitrarily large network containing recursively nested "conceptions of the world". As the examples of the last section showed, the retrieval mechanisms incorporate a rudimentary inference capability, based on property and relationship inheritance and on structural and conceptual matching of queries to clauses. We believe that the efficient implementation of these essential processes in our system provides a firm foundation for further work on mechanized inference and comprehension.

However, most of the inference mechanisms remain to be built, and even some of those built do not fully achieve their purpose. We mention some of the modifications and extensions which seem immediately feasible.

First, we need to decide on a suitable normal form for sentences involving modalities other than stories and propositional attitudes, such as causal constructions and counterfactual conditionals. These permit distribution of the modal operator over a conjunction of consequences (with the aid of "embedded constants") but not over a conjunction of premisses. The classification and access mechanisms should be extended to handle such propositions.

Subnets need to be attached to type concepts as well as individual entities. These would contain conceptions of the world characteristic of entities of those types. For example, knowledge about a particular person's mental world or about a particular fairy tale would be obtained not only from the subnets of those individual entities but also from subnets for humans and fairy tales in general. Clearly, much of our knowledge of what goes on in people's minds (or in fairy tales) is general rather than specific.

It also seems desirable to introduce subnets for certain sets of propositions which are not modally embedded, such as the propositions making up a true story, or a general pattern of events (script?), or the propositions relating to the parts of a system. After all, subnets have no logical significance, serving only to aggregate knowledge into co-accessible, topically and associatively organized clusters. Thus they may be suitable as context structures for contexts other than mental worlds and stories.

The present concept accessing mechanism presupposes a hierarchical taxonomy of types, which may be hard to design for some kinds of objects. For example, artifacts are best categorized by their purpose, but some artifacts are multipurpose (e.g., a knife can be a cutting tool or a weapon). However, it appears to be possible to extend our current methods to deal with mild infractions of hierarchy or with small numbers of alternative hierarchies. Another limitation of concept accessing is that it provides access only on the basis of a single type specification. Thus it does not help to locate objects on the basis of non-type properties such as being an "antagonist" of a story, except to the extent that non-type predications imply type constraints (e.g., antagonists are usually animate beings). Also, concept accessing does not obviate the need for "intersection searches" for objects satisfying multiple constraints such as commonly occur in PLANNER-like problem solving systems, although it may provide an initial set of candidates of a suitable type. Finally, concept accessing may be useless for accessing parts of entities, given the type of part, such as "nose", "rudder", or "department head", since parts tend to form hierarchies "orthogonal" to type hierarchies (Hayes, 1977). Parts accessing may

call for separate "parts access skeletons".

The organization of function tables needs to be modified to allow multi-argument functions, composed functions (e.g., the fingernail of the index finger of the left hand of John), and entities which are the value of more than one function or of the same function applied to more than one argument. The most natural way to allow for multi-argument functions appears to be to detach function tables from concepts, making them indexable by the arguments as well as the names of the functions.

More radical than these extensions will be the introduction of partitioning lattices to represent both taxonomies of types and parts structures, coupled with special inference mechanisms such as those proposed in Schubert (1979). The standard sort of representation of type and parts relationships used at present (exemplified by "All elephants are mammals" and "All mammals have a head") make reasoning about types and parts unnecessarily difficult. Also computationally convenient representations of sets of entities, such as groups of people or an animal's set of legs need to be introduced.

With these extensions most of the substrate for higher-level "combinatory" inference and comprehension processes will be in place.

## References


Brown, J. S., & Burton, R. R. (1975). Multiple representations of knowledge for tutorial reasoning. In D. G. Bobrow & A. Collins, Representation and Understanding, Ac. Press, New York, NY.

Cercone, N. (1975). Representing natural language in extended semantic networks. Ph.D. thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta.

Cohen, P. R., & Perrault, C. R. (1976). Preliminaries for a computer model of conversation. Proc. 1st CSCSI/SCEIO Nat. Conf., Univ. of British Columbia, Vancouver, B. C., Aug. 25-27, pp. 102-111.

Colby, K. M., Tesler, L. & Enea, H. (1969). Experiments with a search algorithm on the data base of a human belief structure. Stanford AI Project memo AI-94, Stanford Univ., Stanford, CA.

Covington, A. R. (1980). Organization and representation of knowledge in a semantic net. M.Sc. thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta.

Goebel, R. (1977). Organizing factual knowledge in a semantic network. M.Sc. thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta.

Hayes, Philip (1977). On semantic nets, frames and associations. Proc. IJCAI 5, MIT, Cambridge, MA, Aug. 22-25, pp. 99-107.

Hendrix, G. G. (1975). Expanding the utility of semantic networks through partitioning. Advance Papers of IJCAI 4, Tbilisi, U.S.S.R., Sept. 3-8, pp. 115-121.

Hendrix, G. G. (1979). Encoding knowledge in partitioned networks. In N. V. Findler (ed.), Associative Networks - The Representation and Use of Knowledge by Computers, Ac. Press, New York, NY, pp. 51-92.

Hewitt, C. (1971). Procedural embedding of knowledge in PLANNER, Advance Papers of the 2nd Int. Joint Conf. on Artificial Intelligence, Sept. 1-3, Imperial College, London, pp. 167-182.

Hobbs, J. R. (1975). A general system for semantic analysis of English and its use in drawing maps from directions. Am. J. Comp. Linguistics, Microfiche 32.

Moore, R. C. (1979). Computational models of beliefs and the semantics of belief-sentences. Tech. Note 187, AI Center, Stanford Research Institute, Menlo Park, CA.

Quillian, M. R. (1968). Semantic memory. In M. Minsky (ed.), Semantic Information Processing, MIT Press, Cambridge, MA, pp. 227-270.

Reder, L. M. & Anderson, J. R. (1979). Use of thematic information to speed search of semantic nets. Proc. IJCAI 6, Tokyo, Aug. 20-23, pp. 708-710.

Rumelhart, D., Lindsay, P. & Norman, D. (1972). A process model for long term memory. In E. Tulving & W. Donaldson (eds.), Organization of Memory, Ac. Press, New York, NY, pp. 198-221.

Rychener, M. D. (1979). A semantic network of production rules in a system for describing computer structures. Proc. IJCAI 6, Tokyo, Aug. 20-23, 738-743.

Schank, R. C. & Rieger, C. (1974). Inference and the computer understanding of natural language. A.I. 5, pp. 373-412.

Schubert, L. K. (1976). Extending the expressive power of semantic networks. A.I. 7, 163-198.

Schubert, L. K. (1979). Problems with parts. Proc. IJCAI 6, Tokyo, Aug. 20-23, pp. 778-784.

Schubert, L. K., Cercone, N. & Goebel, R. (1978). The structure and organization of a semantic net for comprehension and inference. Tech. Rep. TR78-1, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta. A condensed version appeared as Schubert et al. (1979).

Schubert, L. K., Goebel, R. & Cercone, N. (1979). The structure and organization of a semantic net for comprehension and inference. In N. V. Findler (ed.), Associative Networks - The Representation and Use of Knowledge by Computers, Ac. Press, New York, NY, pp. 121-175.

Scragg, G. W. (1975). Frames, planes and nets: a synthesis, Rep. No. 10, Inst. for Semantic and Cognitive Studies, Univ. of Geneva, Centre Universitaire d'Informatique, Geneva, Switzerland.

Shapiro, S. C. (1971). A net structure for semantic information storage, deduction, and retrieval. Advance Papers of IJCAI 2, Imperial College, London, Sept. 1-3, pp. 512-523.

# USE OF AN ATTRIBUTE GRAMMAR IN NETWORK-BASED REPRESENTATION SCHEMES

Clifford R. Hollander
IBM Scientific Center
1530 Page Mill Road
Palo Alto, CA 94304
USA

## ABSTRACT

This paper discusses a technique whereby an attribute grammar can be used to formally define, calculate, and propagate semantic information within the context of a network-based knowledge representation scheme. The goal here is to produce a set of (stored) descriptions, coded as collections of attribute/value pairs, which can serve to characterize the semantics, at least from the viewpoint of certain inference tasks, of the entities and relationships represented by the network. The descriptions are generated in a straightforward mechanical fashion by applying an appropriate attribute grammar to node/link expressions formulated in a given knowledge specification language.

## 1. INTRODUCTION

Network-based representation schemes *1* have proved to be useful tools in organizing, building, and utilizing knowledge bases [3] pertaining to a variety of problem domains and tasks. Ultimately, however, the effectiveness of a representation depends upon the ease with which semantic information can be associated with the various objects and relationships depicted within the network. One way to facilitate this is to look for a formal method by which arbitrary semantic properties of these constituent elements can be mechanically derived from their syntactic characterizations. Ideally such a method will include: (1) A definitional component for statically introducing semantic properties and for expressing the rules governing the assignment of values to properties. (2) A calculational or propagational component for dynamically evaluating semantic rules to yield values for the properties bound to various syntactic constructs. In general, this may entail relating the properties of a subject phrase to those of its syntactic ancestor, descendant, and (even) sibling phrases. (3) An inferential component for utilizing these derived property values to effect certain search and deduction operations.

The approach taken in this paper entails using an attribute grammar [4,5] to formalize the semantics of a particular knowledge specification language, e.g., KRL [6]. As an extended example, the technique is exercised on the language introduced in [7], which defines what is essentially a frame system.

## 2. REPRESENTATION SCHEME

A semantic network (or a frame system) permits the conceptual primitives of a highly structured problem domain to be conveniently represented as an interlinked collection of descriptions. Often, a single formalism can be used both (1) for statically characterizing the objects, activities, and relationships which underlie the domain and (2) for dynamically instantiating pieces of problem specifications and their solutions. This uniformity of representation can greatly facilitate the integration of static and dynamic information where appropriate.

For this discussion, the knowledge structures of interest are composed of descriptive blocks, here called entities. Each entity has a number of identifiable components, each with certain properties, and participates in various relationships, both explicit and implicit, with other entities. An entity is defined by a set of named slots, which elaborate its properties via sets of (nested) descriptive clauses. The ordering of slots within an entity or of clauses within a slot is irrelevent, in that it should not influence the eventual overall effect produced by their processing.

For the purpose of exposition, consider a fragment drawn from the specification language used in [6], as defined by the grammar shown in Fig. 1. In coding these productions, metasymbols beginning with an asterisk denote terminals, a list of <xxx> phrases is denoted by <xxx-1..N>, and an optional phrase is enclosed within square brackets. The principal constructs of the language are (1) names, (2) numeric and string values, (3) relational expressions, (4) entity references, possibly qualified by lists of slot constraints, (5) slot references, and (6) specialization constraints. For simplicity, it is assumed that no two entities have the same name

---

*1* The term 'network-based representation' is intended to to embrace both semantic networks [1] and frame systems [2].

```
<entity>   ::=   <*ent-name> <slot-1..N>
<slot>     ::=   <*slot-name> <clause-1..N>
                | spec <entref>
<clause>   ::=   <*relop> <valspec>
                | is <entref>
<valspec>  ::=   <*value>
                | <slotref>
<entref>   ::=   <*ent-name> [<entcon>]
<entcon>   ::=   with <slotcon-1..N>
<slotcon>  ::=   <*slot-name> <clause>
<slotref>  ::=   slot <*slot-name> of <entref>
```

Fig 1. A Sample Grammar

and that no two slots within an entity have the same name; this is not absolutely essential (see [1]).

Explicit linkage within the network is used to build specialization hierarchies (spec edges) and reference relationships (is and slot edges) among domain elements. Specialization chains represent successive refinements or restrictions of domain classes into nested subclasses. In general, an entity possesses two kinds of descriptive attributes. Each child entity inherits all of the attributes (slots) belonging to its parent (some of which are further constrained by the terms of its specialization), and synthesizes any new ones required for defining the characteristics of its subclass. Each parent entity, therefore, constitutes a factoring of constraint information which must be satisfied by each of its descendants, including entities dynamically created during a specific problem solving session. Each reference declares a slot of the referring entity as being logically bound to (a possibly constrained version of) the referent entity or one of its slots. (Note: In many cases references are bi-directional, in the sense that each entity has a slot which nominates the other.)

Fig. 2. diagrams some of the kinds of structures developed within the system discussed in [7]. Displayed here are portions of six entities, their slots, and their interrelationships within a pair of specialization hierarchies. Specialization relationships are indicated in two ways here: (1) a dashed line connects each child entity with its parent and (2) those slots of the child which are constrained by the terms of the relationship are so indicated. Therefore, the existence of a spec slot as suggested by the syntax is formal rather than actual. Reference relationships, on the other hand, exist as actual values and are depicted by solid arrow.

One hierarchy imposes a taxonomy over a set of device descriptions. The top-level DEVICE entity defines two slots, MODE and ASSOC. Although no constraint is imposed upon the MODE slot here, the ASSOC slot must contain a reference to a DEVICE_FN entity or one of its descendants (see below). The SENSOR entity is a descendant of DEVICE (actually through several levels of specialization). It defines a SIGNAL slot without constraints, a

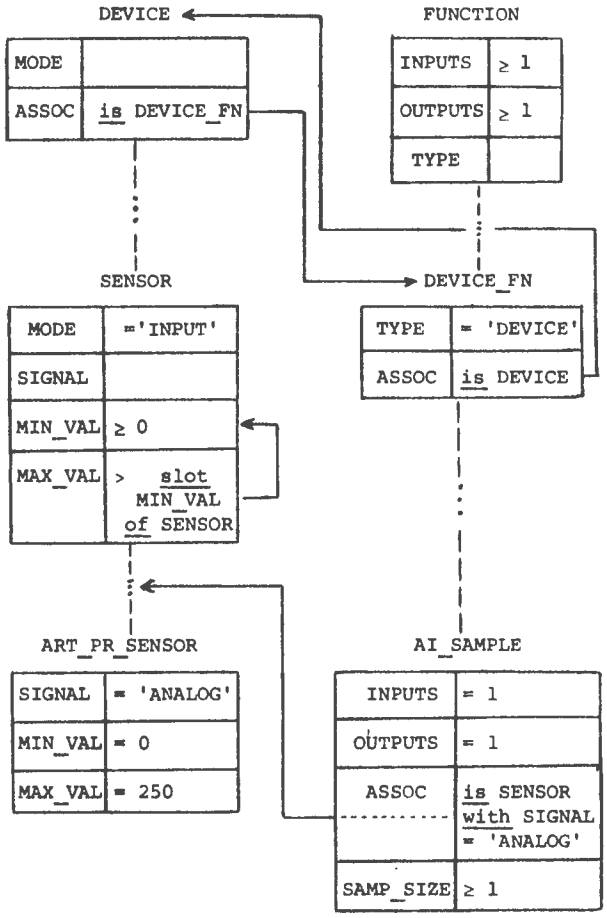



Fig. 2. Sample Entity Structures

MIN_VAL slot which must be filled with a nonnegative number, and a MAX_VAL slot which must be filled with a number larger than the contents of the MIN_VAL slot. The ART_PR_SENSOR entity is used to represent the subclass of arterial blood pressure sensors. Although it does not define any new slots, it specifies that its SIGNAL, MIN_VAL, MAX_VAL slots must respectively contain the values 'ANALOG', 0, and 250.

The other hierarchy is concerned with describing a class of processing functions. At its top level is the FUNCTION entity, which defines three slots: INPUTS/OUTPUTS slots which indicate that a function must have at least one input and one output, and a TYPE slot which here is unconstrained. One of its descendants, the DEVICE_FN entity, represents the class of functions which service devices, as opposed to those that say perform arithmetic operations or data transformations. This entity constrains its inherited TYPE slot to contain the value 'DEVICE'. It also defines an ASSOC slot which must contain a reference to a DEVICE entity or one of its

descendants. At a still lower level, the AI_SAMPLE entity is used to represent the subclass of analog input sampling functions. It states that it has exactly one input and one output, and further that its ASSOC slot must contain a reference to a SENSOR entity which itself has the value 'ANALOG' in its SIGNAL slot. (Note: ART_PR_SENSOR and its descendants satisfy this latter constraint.) AI_SAMPLE also defines a SAMP_SIZE slot which must be filled with a number whose value is at least one.

As an aside, it is easy to imagine several schemes for arbitrarily extending the set of defined relationships: (1) New edge types could be explicitly introduced into the language, each with an accompanying syntactic construct. (2) New kinds of relationships could be represented implicitly by introducing dedicated linkage entities and using these to capture the attributes and referents of each relationship. For instance, either of these techniques could be used to implement an entity-level notion of connectivity or flow among the various elements of a system. In [7], CONNECTION entities indicate data and control flow among device/function elements in an instantiated user application.

## 3. SEMANTIC ATTRIBUTES

An attribute grammar is a context-free grammar for which fixed sets of synthesized and inherited attributes are associated with each nonterminal X. Also, bound to each production

X ::= B

is a set of semantic rules which define how to calculate the values of the synthesized attributes of X and the inherited attributes of the nonterminals appearing in B.

Typically, the parse (tree) which results from the successful syntactic analysis of a piece of text is used to drive its semantic processing. Each node of the tree corresponds to a phrase in the text and to the application of a single production in the parse. One can view each node as being serviced by the appropriate set of semantic rules which compute its attribute values. In order to perform these calculations that rule set requires the values obtained at the parent and child nodes to respectively handle inherited and synthesized attributes. Therefore, observed as a whole, the evaluation process causes attribute values to flow up and down the tree.

The remainder of the paper is devoted to developing sets of attributes and semantic rules for the syntactic grammar given in Fig. 1. The goal here is to be able to associate with each phrase, Q, information pertaining to the entity/slot context which constitute its environment, the nature of the constraint which it imposes upon that environment, and the degree to which static analysis has been able to resolve those constraints. An important point to note here is that in addition to providing a semantic characterization of the network, attribute evaluation can serve both a consistency-checking and a binding function with respect to the rather loosely-coupled elements out of which it is fashioned. In the discussion which follows, semantic rules are developed relative to these semantic attributes:

### Inherited Attributes

enx(Q): internal index of entity context for evaluation of Q.
slx(Q): internal index of slot context for evaluation of Q.

### Synthesized Attributes

stat(Q): resolution status derived from Q; an element drawn from the ordered set of symbolic values RESOLVED>PARTIAL>UNDEFINED>CONFLICT.
val(Q): value(s) resulting from the semantic evaluation of Q.
con(Q): constraint type(s) associated with val(Q).
found(Q): result of the search, if any, specified within Q.

## 4. THE METHOD

The ensuing description is organized into a number of sections. Each section contains (1) a name which identifies a type of construct, (2) a syntactic production, (3) its associated semantic rules, coded in pseudo-Algol form, and, where appropriate, (4) a set of comments on those rules. For brevity, L and R (possibly subscripted) are used to denote, respectively, the nonterminals appearing in the left-hand and right-hand sides of the production. Also, any attribute which does not explicitly receive a value during an execution of a rule set is deemed to have been implicitly assigned the value NULL.

---

### ENTITY SPECIFICATION

<entity> ::= <*ent-name> <slot-1..N>

enx(R) ← ENTINDEX(<*ent-name>)
stat(L) ← MIN(stat(R[1..N]))

Comment: Remember identity of entity; process its slots.

---

### SLOT SPECIFICATION

<slot> ::= <*slot-name> <clause-1..N>

SLNX ← SLOTINDEX(enx(L),<*slot-name>)
slx(R[1..N]) ← SLNX
ATTR ← CONMATCH(<con(R),val(R),stat(R)>)
<con(L),val(L),stat(L)> ← ATTR
SLOTBIND(SLNX,ATTR)

COMMENTS: Remember identity of slot; process each of its clauses; use CONMATCH to consistency-check and resolve clauses; record result.

---

SPECIALIZATION SLOT

<slot> ::= spec <entref>

```
stat(L) ← stat(R)
enx(R) ← enx(L)
IF stat(R) ≥ PARTIAL THEN
   SPECBIND(enx(L),found(R),val(R))
```

Comments: Process entity reference; establish and record specialization relationship.

---

CLAUSE SPECIFICATION

<clause> ::= <*relop> <valspec>

```
CASE stat(R) OF
  [ RESOLVED:  [ IF INCOMPAT(<*relop>,val(R))
                 THEN stat(R) ← CONFLICT
                 ELSE IF <*relop> = '='
                      THEN stat(R) ← RESOLVED
                      ELSE stat(R) ← PARTIAL;
               con(L) ← <*relop>;
               val(L) ← val(R) ]
    PARTIAL: PARTIAL;
    UNDEFINED: UNDEFINED;
    CONFLICT: CONFLICT ]
```

Comments: Process value spec; check operator/value conformance; record value, constraint, and status.

---

CLAUSE SPECIFICATION

<clause> ::= is <entref>

```
con(L) ← 'ENT'
val(L) ← found(R)
stat(L) ← stat(R)
```

VALUE SPECIFICATION

<valspec> ::= <*value>

```
val(L) ← <*value>
stat(L) ← RESOLVED
```

---

VALUE SPECIFICATION

<valspec> ::= <slotref>

```
val(L) ← val(R)
stat(L) ← stat(R)
```

---

ENTITY REFERENCE

<entref> ::= <*entname> [ with <entcon> ]

```
ENTNX ← enx(R) ← ENTINDEX(<*entname>)
  IF LENGTH(ENTNX) = 0
  THEN stat(L) ← UNDEFINED
  ELSE IF NOENTCON
       THEN [ stat(L) ← RESOLVED;
              found(L) ← ENTNX ]
       ELSE [ val(L) ← val(R);
              stat(L) ← stat(R)
              found(L) ← found(R) ]
```

---

ENTITY REFERENCE CONSTRAINT

<entcon> ::= <slotcon-1..N>

```
enx(R[1..N]) ← enx(L)
stat(L) ← MIN(stat(R[1..N]))
IF stat(L) ≥ PARTIAL
THEN [ val(L) ← val(R);
       found(L) ←
          ← INTERSECT(found(R[1..N])) ]
```

Comments: Resolve individual slot constraints; find their (most specialized) intersection.

---

SLOT CONSTRAINT

<slotcon> ::= <*slot-name> <clause>

```
SLNX1 ← SLOTINDEX(enx(L),<*slot-name>)
IF LENGTH(SLNX1) = 0
THEN stat(L) ← UNDEFINED
ELSE [ SLNX2 ← SPECINDEX(enx(L),<*slot-name>)
       ATTRL ← SLOTFETCH(SLNX),SPECFETCH(SLNX2);
       ATTR ← <con(R),val(R),stat(R)>;
       ATTRN ← CONMATCH(ATTRL,ATTR);
       ENTNX ← MOSTSPEC(enx(L),ATTRN);
       SLNX ← SLOTINDEX(ENTNX,<*slot-name>)
       IF LENGTH(ENTNX) ≥ 1
       THEN stat(L) ← CONFLICT
       ELSE [ ATTRS ← SLOTFETCH(SLNX);
              ATTRQ ← CONRES(ATTRN,ATTRS);
              stat(L) ← STATUS(ATTRQ);
              val(L) ← <SLNX,ATTRQ>;
              found(L) ← ENTNX ] ]
```

Comments: Check for existence and binding of slot; check consistency of binding with new clause; find most specialized entity which fits; determine residual constraint over and above its binding;

---

SLOT REFERENCE

<slotref> ::= slot <*slot-name> of <entref>

```
SLNX ← ENSLMATCH(found(R),SLOTINDEX(<*slot-name>))
  IF LENGTH(SLNX) = 0
```

```
    THEN stat(L) ← UNDEFINED
    ELSE IF stat(R) = RESOLVED
         THEN [ ATTR ← SLOTFETCH(SLNX);
                IF LENGTH(ATTR) = 0
                THEN stat(L) ← UNDEFINED
                ELSE <con(L),val(L),stat(L)>
                       ← ATTR ]
         ELSE stat(L) ← stat(R)
```

Comments: Process entity reference; select proper slot and get its binding.

---

## 4. SAMPLE EVALUATIONS

Fig. 3 exhibits the parse trees for the clauses

> slot MIN_VAL of ART_PR_SENSOR

and

is SENSOR with SIGNAL = 'ANALOG'.

For convenience, each nonterminal node is labelled with a number and a phrase symbol, and each

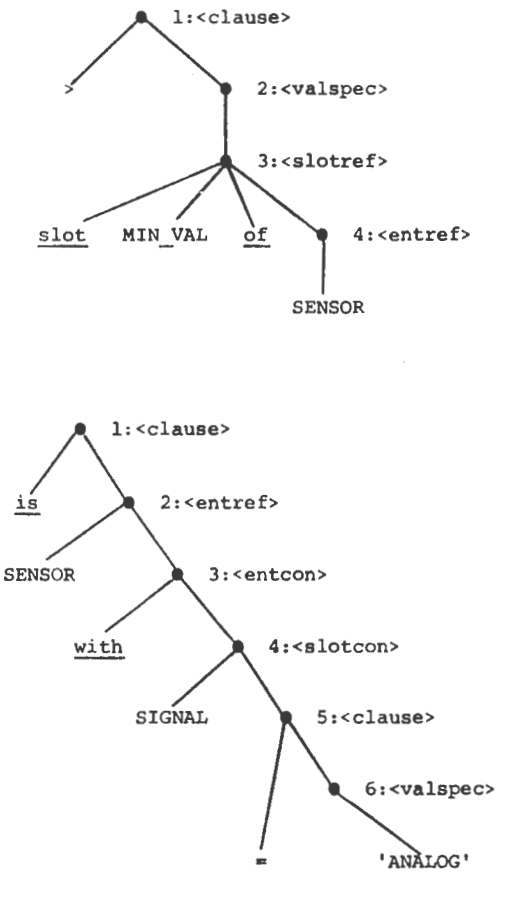



Fig. 3. Sample Parse Trees

terminal node contains the appropriate token.

Attribute evaluation will now be performed relative to these parse trees. In each case, the calculation and flow of attribute information is traced node by node. For brevity, nodes are referenced by tag number. Motion between nodes is explicitly indicated via a trace line. The general scenario at each node is: (1) compute inherited attributes, (2) begin computing synthesized attributes until a descendant node's attribute values are need, (3) process descendant node(s), (4) resume computing synthesized attributes.

```
====== BEGIN CASE-1 ======

*** > slot MIN_VAL of ART_PR_SENSOR ***

====== MOVING TO NODE 1 ======

      case stat(2) ...

====== MOVING TO NODE 2 ======

      val(2) ← val(3)

====== MOVING TO NODE 3 ======

   SLNX ← ... found(4) ...

====== MOVING TO NODE 4 ======

   ENTNX = e(ART_PR_SENSOR)
    *** = entity index ***
     stat(4) = RESOLVED
  found(4) = e(ART_PR_SENSOR)

====== RESUMING NODE 3 ======

SLNX = s(ART_PR_SENSOR.MIN_VAL)
     *** = slot index ***
   ATTR = <'=',0,RESOLVED>
       con(3) = '='
       val(3) = 0
     stat(3) = RESOLVED

====== RESUMING NODE 2 ======

       val(2) = 0
     stat(2) = RESOLVED

====== RESUMING NODE 1 ======

      stat(1) = PARTIAL
        con(1) = '>'
        val(1) = 0

====== END CASE-1 ======


====== BEGIN CASE-2 ======

*** is SENSOR with SIGNAL = 'ANALOG' ***

====== MOVE TO NODE 1 ======

      con(1) = 'ENT'
    val(1) ← found(2)
```

```
====== MOVE TO NODE 2 ======

ENTNX = enx(3) = e(SENSOR)
    val(2) ← val(3)

====== MOVE TO NODE 3 ======

    enx(4) = e(SENSOR)
    stat(3) ← stat(4)

====== MOVE TO NODE 4 ======

  SLNX1 = s(SENSOR.SIGNAL)
  SLNX2 = p(SENSOR.SIGNAL)
    *** = spec index ***
      ATTRL = NULL
    *** no binding ***
  ATTR ← <con(5), ...>

====== MOVE TO NODE 5 ======

      case stat(6) ...

====== MOVE TO NODE 6 ======

    val(6) = 'ANALOG'
    stat(6) = RESOLVED

====== RESUMING NODE 5 ======

    stat(5) = RESOLVED
      con(5) = '='
    val(5) = 'ANALOG'

====== RESUMING NODE 4 ======

  ATTR = <'=','ANALOG',RESOLVED>
  ATTRN = <'=','ANALOG',RESOLVED>
    ENTNX = e(BLOOD_PR_SENSOR)
*** = parent of ART_PR_SENSOR ***
    *** abbreviate BPS ***
      SLNX = s(BPS.SIGNAL)
  ATTRS = <'=','ANALOG',RESOLVED>
        ATTRQ = NULL
   *** no residual constraint ***
    stat(4) = PARTIAL
*** PARTIAL because BPS = subclass, ***
*** so reference not fully resolved ***
    val(4) = <s(BPS.S),NULL>
    found(4) = e(BPS)

====== RESUMING NODE 3 ======

    stat(3) = PARTIAL
  val(3) = <s(BPS.S),NULL>
    found(3) = e(BPS)

====== RESUMING NODE 2 ======

  val(2) = <s(BPS.S),NULL>
    stat(2) = PARTIAL
    found(2) = e(BPS)

====== RESUMING NODE 1 ======

    val(1) = e(BPS)
    stat(1) = PARTIAL

====== END CASE-2 ======
```

## 5. CONCLUSIONS

This paper has explored the idea of employing an attribute grammar as the basis for formally defining, calculating, and propagating arbitrary amounts of semantic information within a broad class of network-based knowledge representation schemes. This approach gave rise to a straightforward mechanical procedure for generating structured collections of attribute/value pairs, which could then aid in efficiently characterizing the contents of the network.


## ACKNOWLEDGEMENTS

The author would like to thank H. C. Reinstein for his substantial contributions to the ideas discussed here and to their realization in an actual system [7].


## REFERENCES


[1] Fikes, R. and Hendricks, G., A network based representation and its natural deduction system, Proc. 5th IJCAI, M.I.T., 1977, pp. 235-246.

[2] Waterman, D. and Hayes-Roth F. (ed.), Pattern-Directed Inference Systems, Academic Press, New York, 1978.

[3] Feigenbaum, E., The art of AI: themes and case studies in knowledge engineering, Proc. 5th IJCAI, M.I.T., 1977, pp. 1014-1049.

[4] Knuth, D., Semantics of context-free languages, Math. Syst. Theory 2, pp. 127-145.

[5] Neel, D. and Amirchahy, M., Semantic attributes and improvement of generated code, Proc. ACM Annual Conf., San Diego, 1974, pp. 1-10.

[6] Bobrow, D. and Winograd, T., An overview of KRL, a knowledge representation language, Cognitive Science 1, 1 (Jan. 1977), pp. 3-46.

[7] Hollander, C. and Reinstein, H., A knowledge-based application definition system, Proc. 6th IJCAI, Tokyo, Japan, 1979, pp. 397-399.

# SELECTIVE INFERENCING

Jerry R. Hobbs
SRI International
Menlo Park, California

Abstract

Ten thorny problems in the study of language comprehension are discussed and shown to depend upon arbitrarily detailed world knowledge and access to context. A "selective inferencing system" is sketched for formalizing context and the use of world knowledge, and it is shown how each of the ten problems reduces to the single problem of how the system selects the appropriate inferences. This spawns a new set of subproblems, but they are problems of a very different flavor from the original problems.

## 1. Some Linguistic Problems

Problems and solutions often cut the world in different ways. Research in language frequently takes the form of choosing a particular linguistic problem, like pronoun resolution, or compound nominals, or metaphor, and seeking a solution. But there is no guarantee that solutions will partition the linguistic space in the same way that problems do. The solution of a particular problem may require a number of components, but each of these components may generalize over many other problems as well. In this paper I want to propose just such a repartitioning of linguistic space, in which many traditional problems collapse into a single problem, that of selecting the appropriate inferences, which in turn decomposes into several subproblems with a flavor quite different from the original problems.

I want to look specifically at ten problems in language comprehension:

1) Compound nominals: First let us consider noun-noun combinations, like "wine glass". To comprehend this, a language user must discover the relation implicit between the two entities, some unspecified portion of wine and the glass. Levi (1978) proposed that the implicit relation must be one of a small fixed set of primitive predicates, including FOR, IN and PRODUCT. Her work is an example of an attack on a single problem, and the "solution" is particular to that problem, in that it solves no other problems on the side.

But there are several difficulties with her proposal. First of all, it fails to capture the complexity that usually characterizes the implicit relation. Thus, "wine glass" is not just a glass FOR wine, but a glass FOR wine to be IN, and that is a simple case. In Hobbs (1979b) I analyzed my favorite compound nominal, from a Newsweek article, "veto pitch", which is a bill Congress "pitches" to the President and which is easy for him to veto just as a pitched baseball might be easy for a batter to hit. We could weaken the claim to say that the implicit relation must be one of the primitive predicates at some level of generality, but that leads to the second difficulty -- the set of predicates are so general they almost cover the world. There is little they exclude, so the proposal lacks empirical content. The relation in "veto pitch" would be perhaps "pitch PRODUCES veto", but that would be unilluminating at best. The proposal does make one prediction however, and that is that the relation cannot contain negation. Thus, a wine glass cannot be a glass that is not for wine. But here's the third difficulty -- the prediction is wrong. On a Gray Line tour of Vancouver last December I learned about the monkey tree. A monkey tree is a tree with spines pointing downward so that it is the only tree that a monkey can NOT climb.

Downing (1977) gives further examples of arbitrary, highly context-dependent relations between the nouns of compound nominals. For example, in a particular context, "apple-juice seat" is the seat at the dinner table that has the apple juice at it. The implicit relation can be anything, and what it is is very dependent on context and arbitrarily detailed world knowledge.

2) Denominal verbs: Clark and Clark (1980) give several thousand examples of nouns used as verbs, such as

The paper boy porched the newspaper.

Like Downing, they argue that virtually any relation can obtain between the noun that has surfaced as a verb and the explicit arguments of the verb. For example, suppose we both know my cousin Max has a habit of sneaking up behind people and rubbing the back of their legs with a teapot. Then you will understand me if I say

Oh no, Max just teapotted a policeman.

Again, arbitrarily detailed world knowledge and sensitivity to the context is required for interpretation.

3) Metonymy, or indirect reference: We commonly refer to one thing as a way of referring to something related to it. For example, we can say

T points to a binary tree,

as a way of saying

T points to the root node of a binary tree.

Nunberg (1978) has investigated this phenomenon extensively and produced a wealth of examples that lead to a similar conclusion. Consider for example,

John sold his Ford for $3000.
John sold his Ford for 57 3/8.

In the first, we mean

John sold his car manufactured by Ford for $3000.

In the second, we mean

John sold his stock issued by Ford for 57 3/8.

Nunberg shows that recovering the implicit function that will take us from the explicit referent to the intended referent can require arbitrarily detailed world knowledge and is highly dependent on context.

4) Metaphor: Consider

John is a real hog. He weighs 300 pounds.
John is a real hog. He wouldn't shut up and let me talk.

How do we pick out the features John and hogs share? In the first case, it is physical characteristics; in the second, the overconsumption of a limited resource. For the interpretation of metaphors, again world knowledge and context.

Now some coreference resolution problems:

5) Definite noun phrase resolution: In a text we will work again and again,

(1) John can open Bill's safe. He knows the combination,

how do we know what "the combination" refers to?

6) Pronoun resolution: In (1) how do we know whether "he" refers to John or Bill? Knowing the combination can be inferred both from being able to open and owning.

7) Resolution of omitted arguments: In (1), the combination of what? And how can we find out?

Finally, some ambiguity problems:

8) Lexical ambiguity: In (1), what sense of "combination" is being used, the combination of a lock or the combination of several objects? And how do we know?

Syntactic ambiguity:

9) The prepositional phrase problem: In

I drove down the street in a car,

we need to know about streets, cars, and driving to be able to parse the sentence correctly. In the old favorite,

I saw the man in the park with the telescope,

it is not enough to know about seeing, men, parks, and telescopes. We need specific knowledge of the situation. Once we have that knowledge, how do we apply it?

10) Very compound nominals: Compare "Stanford Research Institute" and "Cancer Research Institute". In the first "Stanford" bears some direct relation to "Institute", not to "Research". In the second, "Cancer" bears the direct relation to "Research", not to "Institute". This kind of ambiguity infects the syntactic analysis of complex texts almost as much as the prepositional phrase problem.

All these problems point in the same direction. Arbitrarily detailed world knowledge is required to solve them, and the interpretation has to be done against a very specific context. Linguists frequently see in this observation the hopelessness of finding solutions. Workers in natural language processing (NLP) tend to be a bit more brash and see in it a challenge. It is a challenge I propose to take up for the rest of this paper.

First we should be clear about the nature of the solution we are looking for. For many of the problems, there are efficient methods that work on the majority of the cases. For example, there is a fairly simple algorithm for pronoun resolution, working strictly on the parse trees of the sentences in the text (cf. Hobbs 1976a), that is correct more than 90% of the time in published texts. But it doesn't work everywhere, for instance, on the kinds of examples Charniak (1974) came up with.

For many compound nominals, such as "wine glass", the computationally efficient solution is simply to stick it into your lexicon as a phrase. This is even principled in cases like "wine glass", for there are things we know about wine glasses, e.g., they have stems, that can't be deduced from our knowledge of wine and glasses. But this method does not help with novel combinations, like "turpentine jar".

On a higher level, if we have strong script-like knowledge about what we're trying to comprehend and our understanding need not be deep, the solution to many of these problems simply falls out or becomes unimportant in script-based processing (Schank and Abelson 1977, Schank, Lebowitz, and Birnbaum 1980). Suppose for example I'm skimming reviews of TV shows. I know the show has a title, a topic, and some actors, and that the reviewer will evaluate the show. The title is in bold-face: the topic will come immediately after and will be a description of a sequence of related events; the actors' names will be capitalized: and the evaluation is likely to be expressed in terms of evaluative adjectives used literally. This is my script, and using it, I can process the text quite rapidly and understand enough for me to decide whether to watch the show. But if I want to understand something about the source of the reviewer's opinions and how much his evaluation would coincide with my own, neither this TV-review script or any other TV-review script will suffice. Single-script processing does not help us when we are trying to make sense out of a novel text, or make deep sense out of any text.

What is common to all these efficient partial methods is that an interpretation problem is transformed into a simpler recognition problem. It is important for NLP researchers to study these methods, but they should not lead us to abandon the search for full solutions.

Even in cases where the simple methods work, such as treating "wine glass" and "monkey tree" as single lexical entries, it is still of interest to know why the particular usage is motivated (Fillmore 1979): "wine glass" is not an entirely arbitrary way of referring to wine glasses, in contrast with "jug", which is an entirely arbitrary way of referring to jugs. We would like the motivation behind an expression to be given by the way its interpretation would be computed, even though it usually isn't computed. But a finer distinction is necessary, between the computable, like "wine glass", and the noncomputable but motivated, like "monkey tree". While both are stored and recognized as single lexical entries, the correct interpretation of the former could be computed. For the latter, there is a reasonable explanation of the correct interpretation, but a naive language user would not necessarily be able to choose that interpretation over other possible interpretations if he had to compute it himself. Figure 1 summarizes all this.

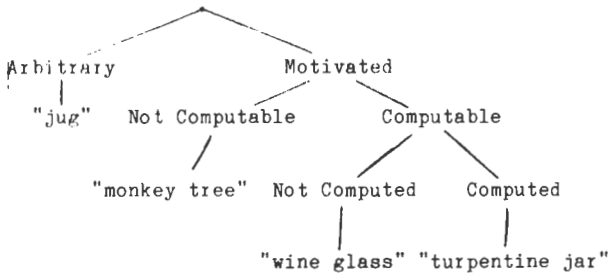



Figure 1.

As processes of comprehension, the solutions we will be seeking in this paper are relevant only to the rightmost case, the computed. But as the processes that underlie or motivate the cultural existence of a particular expression, the solutions will be relevant to all but the leftmost.

## 2. Selective Inferencing

### 2.1. Deductive Systems in Mathematics

If we are going to take up the challenge of full solutions to these ten problems, we need a mechanism for dealing with world knowledge and context, for going beyond what is given explicitly. Perhaps the best guide we have in this enterprise is the deductive systems of mathematics (DS), so these are outlined first. But if our aims are not the development of mathematical theories, but the analysis of linguistic texts, we need to modify the DS in certain ways, and these are discussed next. The result is what I will call a "selective inferencing system" (SIS). Much of it is a recapitulation of common practices and principles of AI, but I hope it includes a few new suggestions. In the course of developing the SIS, I will "solve" the ten linguistic problems, by reducing them to the single problem of how a SIS selects the right inferences. The subproblems this spawns are discussed as we go along and summarized in Part 3.

A DS can be characterized by the formalism used, the process of using it, and the mathematician's commitment to it. We will look at each in turn.

1) Formalism: The formalism consists of a set of symbols and means of combining them into expressions. Certain expressions are designated axioms. There are a few rules of inference, such as modus ponens and universal instantiation, according to which expressions can be manipulated. When applied to axioms, they yield theorems.

2) Process: Anything derivable from the axioms by application of the rules of inference is a theorem. But in practice, the mathematician, viewed as the one who turns the crank on the deductive engine, doesn't go about enumerating all theorems; he only proves the interesting ones. At any given moment in the deductive process, some axioms and theorems are highly focused, in the spotlight, so to speak, while most remain very much in the background. And most theorems he simply lacks the resources to prove. I mention these facts because we will have to formalize aspects of this process.

3) Commitment: The mathematician assigns to each symbol in the formalism an individual or set in some Platonic universe whose existence he accepts as unproblematic, at least for the purposes of the enterprise. Legal ways of combining symbols into expressions are assigned corresponding set construction operations in the universe, in such a way that the axioms correspond to statements about the universe. In positing an expression as an axiom, the mathematician commits himself to the truth of the corresponding statement about the universe. The rules of inference preserve truth, so the mathematician also commits himself to the truth of the statements about the universe corresponding to the theorems.

I am not calling all this by its usual name "semantics". "Semantics" is by now hopelessly ambiguous in an interdisciplinary field like cognitive science. Philosophers mean one thing, psychologists another. ("Meaning" is even worse, since it means something to ordinary people as well.) But the real difficulty is that logical semantics assumes we can talk reasonably about the world in a way that is independent of someone's perception and interpretation of the world. Like many people in AI, I find myself very uncomfortable in that mode of discourse, and bereft of intuitions. But I can make sense of talk about the social enterprise of building and using formal systems. What surfaces in logical semantics as "meaning" surfaces here as "commitment".

It is good to take a DS as our starting point and depart from it only where we have to. In this way we run the least risk of that plague of AI, reinvention.

### 2.2. A Selective Inferencing System

We wish to construct a mechanism which, when presented with a text in a particular context, will apply the appropriate world knowledge, including knowledge of the context, to solve the linguistic problems posed by the text, including those laid out in Part 1. We will assume a syntactic component has already recognized predicate-argument relations and reduced the text to the mechanism's internal language, perhaps with some annotations signaling where the problems are. Later in the paper I will discuss some features of the internal language necessary to make this assumption reasonable.

We need a label for this component of language processing. The undesirability of "semantic" was discussed above. "Deduction" won't do, since that's what we're contrasting this mechanism with. "Entailment" is a bit too strong for what I have in mind. "Implicature" means two things, conversational implicature and conventional implicature, and these turn out to require two quite different mechanisms when made precise. "Reasoning" implies a conscious activity, whereas the processing of the sort we will try to simulate ranges from barely conscious to deeply unconscious. "Interpretation", or more metaphorically, "comprehension", is what the mechanism does, but not how it does it. "Understanding" is too loaded to be a useful technical term: hermaneuticists tell us it's obvious that machines will never be able to understand; there's something to that; machines without toes will obviously never understand "John stubbed his toe" in the way we do.

So we will settle upon the label AI has already chosen -- "inference". To verb the label, we will say the mechanism "draws inferences". But how to nominalize the process is a problem. "The drawing of inferences" is too cumbersome. "Inferring" seems more appropriate for the single act of drawing an inference than for the process that underlies those acts. So we will again bow to AI custom and use the illiterte term "inferencing". The mechanism will be designed to draw inferences selectively: hence, the title "Selective Inferencing".

We can divide our discussion into five sections. Corresponding to the formal language of a DS, we need to specify our language of representation. Corresponding to the axioms of a DS, we need to discuss the knowledge to be represented. Corresponding to the rules of inference, we need to talk about the operations that use the knowledge by manipulating the representations. Corresponding to the process of applying rules of inference to derive theorems is the mechanism of control for the operations. Finally, we need to discuss our commitment to our formal system.

By cleanly separating these five issues for the purposes of this discussion, I do not mean to imply they should be cleanly separated in implementation. There may be many good implementation reasons for mixing the categories.

For the rest of Part 2, differences between the deduction systems of mathematics and the selective inference system required in NLP will be forced upon us. But first I will tip my hand. Figure 2 summarizes the differences and should help the reader keep track of where we are.

| Deduction System | Selective Inferencing |
|---|---|
| Representation: | |
| few predicates | about 1 predicate / English morpheme |
| few constants & variables play a role at given time | about 1 constant or variable / morpheme in text |
| Knowledge: | |
| few axioms | about 1 axiom / fact we know about the world |
| deep proofs | shallow proofs |
| Operations: | |
| simple calculus: rules of inference only | complex calculus: rules of inference under control of higher discourse operations |
| Control: | |
| in theory, no resource constraints, no control over proof process | means for controlling proof process formalized |
| Commitment: | |
| commitment at symbol level | heuristic commitment to axioms as used by by operations; ultimate commitment to external behavior. |

Figure 2.

### 2.3. Representation

Our formal language should satisfy two criteria:

1) It should be close to English. The mechanism is responsible for comprehending, and ultimately, generating English texts. The translation between English and the formal language is likely to be easier the closer to English the formal language is. The ideal choice by this criterion is English itself, but it fails monumentally on the second criterion.

2) It should have a simple syntax. This eases the manipulation of the representations, and manipulation is what the mechanism does most of the time. Much of the complexity of English syntax, e.g. the division of predicates into nouns, adjectives, verbs, adverbs, and prepositions, reflects a conceptual scheme that is better captured in the axioms than in the syntax of our formal language. Hence, we will stick to the language of predicate calculus, a language of predicates, constants, variables and quantifiers. To satisfy criterion (1), there will be nearly a one-to-one mapping of the morphemes of English onto the predicates of the formal language; "onto" since there will be other predicates as well. For any text processed, there will be about one predication and one constant or variable for every morpheme in the text. Rather than taking the time to define the language formally here (it is done in Hobbs (forthcoming)), I will simply paraphrase all the formal expressions I have occasion to use.

Thus in our language of representation, we need to depart very little from what is provided by a DS, although a SIS, because of its profusion of symbols, will have a very different look to it.

## 2.4. Knowledge

In a DS, we typically build on a small number of carefully crafted axioms, and proofs of theorems are generally quite deep. A SIS, on the other hand, must be axiom-rich. The axioms encode the knowledge we have about the world, and I would expect there to be about one axiom for every fact we know -- that is, lots. There would be little point in an effort toward an elegant, independent set of axioms. Because of resource limitations, proofs will have to be shallow, and many expressions that could be proven as theorems will have to be stored as axioms anyway. For example, even if it were possible to deduce many of the details of our economic system from general ideological principles, the SIS should not have to do so every time it reads a text about supermarkets. Nevertheless, facts should be stated at an appropriate level of generality. Thus, some of the things we know about supermarkets are just facts about supermarkets. But most are facts about food stores, or stores, or buying and selling in general.

My preference is for the facts about the world to be stated in as atomic a form as possible. For example, I prefer a system to have a bunch of small facts about baseball, along with the facts that they are about baseball, easily accessible from each other, rather than having a large-scale baseball schema that has to be used as a unit. Then we can use our knowledge about baseball selectively. For example, in interpreting "veto pitch" we could access facts about pitchers and batters without accessing facts about third basemen and outfielders. However, I have no objection to including such large-scale schemata, and would view them simply as other axioms.

I don't think it's useful to distinguish among kinds of knowledge, except as we are forced to by differences in the way the operations manipulate the knowledge. In particular, the distinction between the dictionary and the encyclopedia, or the distinction between lexical and world knowledge, as well as the distinction between analytic and synthetic truths, will probably turn out to be useless in our system. Thus, both

A husband is a male,

and

A husband usually lives with his wife,

will both be expressed by axioms of the same sort and will have the same status in the system.

## 2.5. Discourse Operations

So far we have had to depart very little from a DS. But now a radical departure forces itself upon us.

In theory, in a DS, there are only a few rules of inference, like modus ponens and universal instantiation, and whenever they apply, we have to live with whatever they produce. In practice, only a few things are really proven; the rest just lies in reserve.

For us, things cannot be as simple as a DS is in theory. We have to formalize the practice. The standard rules of inference will have to be under strict higher control. One reason for this is the likely inconsistency of our set of axioms; this is taken up below. Another reason is that of the vast number of inferences we might draw from the information in a text, only a small number will be relevant.

We need to divide the expressions into two categories: a knowledge base, our repository of axioms waiting passively to be used, and a spotlight (to use a technical term no one else will be tempted to adopt), which contains

1) the current sentence being processed,
2) a representation of the previous text, and
3) some representation of the external environment,

together with the inferences drawn from the knowledge base that are determined to be relevant to the interpretation of the text. The inferences are not the axioms themselves, but the instantiated conclusions of the axioms.

Given this structure and our language of representation, what would it be for our SIS to comprehend a sentence in the text? It would have to relate the sentence to itself, to the previous text, and to the representation of the external environment. The problem of relating the sentence to itself arises from two phenomena of English discourse:

1) English allows predicates to be applied to arguments rather freely. Metaphor and metonymy

are examples. Predicates and their arguments have to be interpreted in such a way that they become congruent. This leads to the operation of predicate interpretation.

2) English allows some predicates to be implicit. Compound nominals and denominal verbs are examples. These predicates should be made explicit. This leads to the interpretation of implicit predicates.

Relating the sentence to the previous text involves two kinds of problems:

3) English is linear and when something appears in several grammatically unrelated predications, it has to be mentioned more than once. If the text is to be comprehended, the identity between the mentioneds has to be recognized. Hence the operation of coreference resolution.

4) What the current sentence asserts should be related to what was asserted in the previous text. Thus, we should discover coherence relations.

5) Finally, the sentence has to be related to the external environment. This could be broken down in a number of ways, but in this paper, I won't.

These five discourse operations are dictated by the nature of the mechanism and the language of representation. They each seek to satisfy certain requirements imposed by the text. The requirements are expressed in terms of inferences to be drawn from the knowledge base. The discourse operations work by searching the knowledge base for inferences satisfying the requirements. When they find them, they instantiate them into the spotlight, and perhaps make certain minor modifications to the original representation of the text. These five discourse operations encode our hypothesis about what it is to comprehend a text. Hence, rather than drawing inferences freely as in a DS, we will draw inferences only as dictated by these discourse operations. They constitute the fundamental mechanism for selecting appropriate inferences. Moreover, it is here that we see our ten linguistic problems translated one by one into the problem of selecting the appropriate inferences.

Let us look at each operation in greater detail:

1. Predicate interpretation: Elements of the text have to fit into their local environments. This can be viewed as a more active version of the old check on selectional constraints. It can be stated roughly as follows:

(2) Given proposition p(A), from PROPS(A), infer REQ(p),

where "PROPS(A)" is the set of properties of A, or the set of propositions in which A occurs as an argument, and "REQ(p)" is the set of requirements associated with the predicate p. In the simplest case this checks selectional constraints. For example, the sentence

T points to the root of a binary tree,

might be represented

point(T,R) where root(R,B), binary-tree(B).

That is, T points to R where R is the root of B and B is a binary tree. REQ(point) would be "node(R)"; that is, variables can only point to nodes. PROPS(R) would be "point(T,R), root(R,B)". The knowledge base would include the axiom

(3) (Ax,y)(root(x,y) --> node(x)),

that is, if x is the root of y then x is a node. In Katzian terms, this is equivalent to having the feature +NODE attached to "root". The axiom would be used by the predicate interpretation operation to satisfy requirements of (2). "Node(R)" would be instantiated and placed in the spotlight.

But predicate interpretation is in general more active than this. Consider a case of metonymy.

T points to a binary tree.

or

point(T,B) where binary-tree(B).

"REQ(point)" is the same. PROPS(B) is "point(T,B), binary-tree(B)". There is an axiom

(4) (Ax)(Ey)(binary-tree(x) --> root(y.x)),

that is, a binary tree has a root. Axioms (4) and (3) lead to the satisfaction of (2), "root(R,B)" and "node(R)" are instantiated with some new constant R, and the original assertion is changed to "point(T,R)". In this modification, it resembles the type coercion found in some programming languages.

It is often a problem to determine what the requirements REQ of a local environment are. In

John sold his Ford for 57 3/8,

we would have to determine from "sold ... for" that "57 3/8" has to be interpreted as money. Only stocks are priced in eighths, so we need to infer a relation between stocks and Ford. Once we know that, we can use axioms that Ford is a corporation and that corporations issue stock. But the reasoning leading to "REQ = stocks" is not simple.

2. Interpretation of implicit predicates: The operation for finding the implicit relation between the nouns in a compound nominal can be stated roughly,

> Given r(N1,N2), find P such that P can be inferred from PROPS(N1) and P can be inferred from PROPS(N2) (Find an intersection between PROPS(N1) and PROPS(N2)).

For example in resolving "wine glass", we have r(N1,N2) where wine(N1) and glass(N2), that is, N2

is a glass and N1 some non-specific portion of wine. We can determine the relation r by using the axioms

```
(Ax)(wine(x) --> liquid(x))
(Ay)(Ex)(glass(y) --> purpose(y, contain(y,x))
            & liquid(x)).
```

That is, wine is a liquid and the purpose of glasses is to contain liquids. The intersection P is "liquid(x)". We instantiate into the spotlight the proposition "liquid(N1)" and replace "r(N1,N2)" by "purpose(N1,contain(N1,N2))".

Denominal verbs are similar to compound nominals. In the pattern "N1 N2ed N3" we need to find some plausible action, or "doing", of N1 that involves N2 and N3. Roughly,

> Find an intersection P between PROPS(N1), PROPS(N2), and PROPS(N3) such that from P, "do(N1)" can be inferred.

For example, in

> The paper boy porched the newspaper,

we might use the axioms

```
(Ax)(Ey,z,w)(paper-boy(x) -->
        deliver(x,y,z) & newspaper(y)
        & near(z,w) & front-door(w)),
(Az)(Ew)(porch(z) --> near(z,w) &
        front-door(w)),
```

saying that paper boys deliver newspapers to places near front doors and that porches are near front doors, to interpret the denominal verb "porch" as "deliver to the porch".

In giving the example in Part 1,

> Max teapotted a policeman,

what I did by setting up the context was to place an action by Max involving teapots and people into your spotlight. That action would thus be used to interpret the sentence.

3) Coreference resolution: This includes resolution of definite noun phrases, pronouns and implicit arguments.

There are two methods for resolution. The first is direct. We need to find something in the previous text from which we can prove the properties of the constant to be resolved. Roughly,

> To resolve A, from PREVIOUS.TEXT, infer P where P is in PROPS(A).

For example, returning to (1),

(1) John can open Bill's safe. He knows the combination.

we look for something in the previous text that implies the existence of a combination. We find it using the axiom

```
(Ax)(Ey)(safe(x) --> combination(y,x))
```

or a safe has a combination. Similarly, the pronoun "it" in

> John can open Bill's safe. He knows its combination.

would be resolved using the same axiom.

The second method is more common for pronouns and implicit arguments. It is what in Hobbs (1979a) I called "petty implicature". When other discourse operations would succeed if only a certain identification were made, we assume the identity as a kind of conversational implicature, and solve two problems at once. For example, in (1) the second sentence can be seen as an elaboration of the first by the coherence operation mentioned below if only we identify "he" with John. So we do.

Coreference resolution can also be used to resolve many syntactic ambiguities. But let's work into this gradually, looking first at a real coreference problem -- reflexives. Usually there is no problem. In

> Jane gave Bill a picture of himself,

"himself" can only refer to Bill. But in rare instances, such as

> John gave Bill a picture of himself.

"himself" is ambiguous. The resolution is a problem for selective inferencing to solve. But we would like syntax to pass on to the inferencing component all that it has been able to find out. In this case, it's quite a bit -- "himself" can only refer to John or Bill. We can encode this by representing "himself" with the "ambiguous constant", "[John/Bill]". It must be resolved, but it can only be resolved to one of the two entities.

The same device will extend to some syntactic ambiguities, including the prepositional phrase problem. Our representation of the classic

> I see the man in the park with the telescope,

will include

```
see(S,I,M), in([S/M],P), with([S/M/P],T)
```

That is, S is a seeing action by I of the man M, either S or M is in the park P, and either S or M or P has the telescope T. The inference component can then solve the problem by finding something that implies the properties of the ambiguous constant, or by petty implicature. This reduction of the prepositional phrase problem to a coreference problem is a fairly natural one, as seen by the pair

> John drove down the street in a car.
> John drove down the street. It was in a car.

In both cases we have to decide whether the street or the driving is in the car.

Similarly this device extends to the problem of "very compound nominals". In "Stanford Research Institute" there are two implicit relations, one "r1(R,I)" between research and the institute, and the second "r2(S,[R/I])" between Stanford and either research or the Institute. In seeking to interpret r2, we look for a link both from the properties of R and the properties of I. Whichever results in success will cause the ambiguous constant to be resolved.

I do not mean to imply that all syntactic ambiguities translate so elegantly into coreference problems. For example, I see no easy way to deal with the old favorite

They are flying planes,

in this way. For such examples, we may just have to turn the inference mechanism loose on all the parses to see which it can make sense out of first, or alternatively, that frequently urged approach, let the inferencing get a piece of the parsing action.

4. Discovering coherence relations: Hence, discovering the structure of the text. In Hobbs (1978) I proposed a reasonable number of possible coherence relations and made the rash claim of exhaustivity. To give an example, one of the coherence relations is "Elaboration", and its requirements can be stated roughly,

> Given current sentence S1, find a sentence S0 in PREVIOUS.TEXT for which there is an intersection P between ASSERTION(S0) and ASSERTION(S1).

That is, infer an intersection between the assertion of the first sentence and the assertion of the second. Thus in (1), we would use an axiom saying that if someone can cause a state to come about, then he knows an action that will cause it, an axiom that it is common knowledge that one can dial the combination of something to open it and an axiom that one knows the implications of what one knows. The intersection would be that John/he knows some action (dialing) that will cause the safe to be open. The relevant propositions are instantiated and some representation of the text structure is encoded.

Klapholz and Lockman (cf. Lockman 1978) see some of the coherence relations as examples of coreference resolution between structures in the text larger than simple constants, and this may be a useful point of view.

5) Relating the text to the world: This seems like a broad requirement that defies formalization, but in AI we have been able to described formally certain well-behaved portions of the world by means of task models, scripts, plans, grammars and the like. Where we can do this, it becomes one of the discourse operations to relate the text to the model of what is going on in the environment. I take most work in NLP to be efforts on this problem. For example, Grosz (1977, 1980). A. Robinson (1980), Hobbs and J. Robinson (1979) attempt to relate task-oriented dialogs to a model of the task. Mann, Moore and Levin (1977) relate dialogs to dialog games encoding the expected course of the dialog. Schank and Abelson (1977) and Wilensky (1978) relate what is said in stories about the characters to the characters' conjectured plans. Allen (1979) attempts to relate utterances to the speaker's plan.

A simple version of this discourse operation might be stated

> Given PLAN in the spotlight and a sentence S to be interpreted, from Grow(PLAN), infer ASSERTION(S).

We assume what is said is a general statement referring to the specifics of the plan, so we try to infer the general statement from some specific information in the plan, growing the plan to a deeper level of detail if necessary.

Two of our ten problems do not seem to reduce easily to one of the five discourse operations -- metaphor and lexical ambiguity. The problem of metaphor is treated from the perspective of selective inferencing elsewhere (Hobbs 1979b). Here I will mention the approach just briefly. When a speaker uses a metaphorical predicate, as in "John is a hog", he intends the listener to see certain similarities between John and hogs, say the similarity that both overconsume. On the other hand, it is intended that other properties of hogs, such as four-leggedness, won't be inferred. That is, to comprehend a metaphor, the listener must select certain inferences as appropriate and reject others -- just the process of selective inferencing. In Hobbs (1979b), it is shown how the discourse operations frequently lead to the correct interpretation of the metaphors, especially the operation of predicate interpretation in the case of spatial metaphors, by selecting the right inferences.

Now lexical ambiguity: Ambiguity is not a property of expressions but rather a relation between an expression in one representational system and a second representational system. For example, "men" is not ambiguous between "two men" and "more than two men" unless we are translating into a language with dual as well as plural forms. If we are translating into a pictorial representation that requires a specific number of men, it is infinitely ambiguous. Otherwise the term is not ambiguous but merely vague.

It is usually possible to devise a propositional target representational system in a way that will make an expression's translation vague rather than ambiguous. One way of doing this, say for the pair "bank1" of a river and "bank2" where you get loans, is to have a general predicate "bank" that is implied by both of the specific predicates:

(5a) (Ax)(bank1(x) --> bank(x))
(5b) (Ax)(bank2(x) --> bank(x))

This may seem implausible -- what do the two kinds of banks have in common? But what they have in common is precisely what is captured by the axioms (5) -- they are both called banks.

How is the ambiguity resolved, or the vagueness made more precise? Let's return to our workhorse example:

John can open Bill's safe. He knows the combination.

We have the axioms

(Ax)(Ey)(safe(x) --> combination1(y,x))
(Ax,y)(combination1(y,x) --> combination(y,x))
(Ax,y)(combination2(y,x) --> combination(y,x))

The solutions to the definite noun phrase and coherence problems use the first two axioms and not the third and thus require us to instantiate an expression with "combination1", not "combination2". So the lexical ambiguity problem is solved as a by-product of other discourse operations.

I would guess that this is typical, that most lexical ambiguities that matter would be solved by the ordinary workings of the discourse operations.

## 2.6. Control

In defining a DS, we need not be explicit about the proof process itself. In defining our SIS, we must, in particular about the order in which searches for proofs are conducted. The problem of AI is how to control inferencing and other search processes, so that the best answer will be found within the resource limitations, fairly quickly, and before other plausible candidates. In an NLP system with a rich knowledge base, the last of these problems is likely to be the most severe. The difficulty will not in general be that no proof satisfying the requirements of the discourse operations will be found. It is that too many will be found, each leading to a different interpretation. This remains a major research issue, but I can suggest three leads.

First, the axioms should have associated with them a measure of salience. In part this would include a measure of a fact's "natural salience": for example, the fact that an animal has a head is naturally more salient than the fact than an animal has a pancreas. But it would also include a component of dynamically varying salience which would encode the likely relevance of the axiom to the particular context: when reading an article about politics, axioms about ice hockey would have a low salience, but if ice hockey is mentioned, say, in a passage about the President congratulating the Olympic hockey team, then the salience of axioms about ice hockey is increased. Such an increase could overcome an axiom's natural low salience.

The search for the appropriate inferences would then be conducted in an order determined by the salience of the axioms and the length of the proofs. The inference chosen would be the first one encountered in this process that satisfied the requirements.

This already gives us a concise explanation of certain features of some of the linguistic problems. Nunberg (1978) developed a framework for analyzing indirectly referential expressions, in which one got from an explicit referent to the intended referent by applying functions derived from world knowledge, such as "stocks issued by", "owner of", and "father of". He proposed what he called the "Identity Principle", which says roughly that if a function or sequence of functions has already arrived at a referent that satisfies the requirements of the local linguistic environment, we cannot apply a further function to get to the intended referent. This explains the fact that, while we can point to a car and say "He's crazy," meaning the owner of the car is crazy, we cannot point to a picture of George IV and say "He was crazy," meaning the father of George IV (George III) was crazy. George IV already satisfies the requirements imposed by the predicate "was crazy".

Nunberg's functions translate directly into our inferences. Then in terms of a salience-and-length-ordered search, the Identity Principle simply says that we draw the first inference that satisfies the requirements of the discourse operations. Predicate interpretation on the predicate "crazy" requires that it's argument be human. In the first case, the fact that a car has an owner gets us to the intended referent. In the second case, George IV already satisfies the requirements so we do not look further.

My second suggestion is that we should take advantage of the natural redundancy of almost all texts (Joos 1972). The nature of the language makes texts highly redundant and we do well to assume the maximum redundancy possible in our interpretation of a text. Let us look at (1) again:

(1) John can open Bill's safe. He knows the combination.

There are at least five plausible solutions to problems posed by the text. Solution 1: If someone owns the safe then he knows the combination, so Bill is a candidate for the referent of "he". Solution 2: One sense of "combination" is of an arrangement of several similar entities, and the set consisting of John and Bill satisfies that. Solution 3: The inferences and associated petty implicatures involved in recognizing the Elaboration relation described above, result in the resolution of "he" and "the combination" both and solve the text structure problem as well. Solution 4: Since if someone can open a safe, he probably knows the combination, John is a good candidate for the referent of "he". Solution 5: Since safes have combinations, a good guess is that "the combination" of the second sentence refers to the combination of the safe of the first sentence.

There are two ways in which redundancy operates. First, in Solution 3, the fact that one solution solves three problems reflects the redundancy of the information in the text. Secondly, in Solutions 3, 4, and 5, we have consistent, redundant solutions to the coreference problems. Figure 3 summarizes this.

| Interpretation 1 | Interpretation 2 |
|---|---|
| Solution 1 | Solution 2 |
| he = Bill | comb = (J,B) |

Interpretation 3

| Solution 3 | Solution 4 | Solution 5 |
|---|---|---|
| Elab(S0,S1)<br>he = John<br>comb of safe | he = John | comb of safe |

Figure 3.

Here we see one solution, Solution 3, solving three problems and two examples of two solutions, Solution 3 + Solution 4 and Solution 3 + Solution 5, solving the same problem consistently. This example suggests that the way to capitalize on the text's redundancy is by looking for single solutions that solve multiple linguistic problems and for consistent multiple solutions for single problems, and favor the resulting interpretations over others.

My final suggestion is to use metarules, as suggested by Davis (1977), to guide the inferencing. Few such rules have been proposed, but one possibility arises in the problem of reasoning about someone else's reasoning (e.g. McCarthy 1979). In one approach, you need an axiom that says someone will draw the valid inferences of what he knows. It turns out that much of the inferencing in this domain is driven by that axiom. Thus, it would be good to have a metarule specifying the circumstances under which that axiom should be tried first (cf. Hobbs forthcoming).

It might be useful to collect into one place all of the ways in which context enters into the selective inferencing system. First of all, the salience on the axioms varies with the context, thus varying the order in which the knowledge base is searched. Secondly, what is in the spotlight has the highest salience, so the text itself and the previous inferences are part of the context that influences interpretations. Thirdly, the previous text has a structure deriving from its coherence relations, and this influences the salience of expressions in the text (cf. Hobbs 1976b). Finally, the model of the external environment has a structure that influences search order (cf. Grosz 1977). In short, context is formalized as what is known and the order in which what is known is accessed.

## 2.7. Commitment

Let's begin with an analogy of the sort we AI-ers are fond of. Suppose you build a very complex clock, telling the time of day, the day of the week, the date, the phase of the moon, the season, perhaps even the sign of the zodiac each planet is in. In other words, your clock simulates a number of astronomical phenomena. When you sell the clock to me with this description, you are committing yourself to the described correspondence of its external behavior to the astronomical phenomena, and if the correspondence fails to hold, I will have a right to complain and you will have an occasion to worry about what went wrong.

But suppose I open up the clock and demand that you tell me which astronomical object corresponds to each gear -- where's Mars, for example. At this point I have exceeded the level of detail of your commitment.

In a DS, the level of commitment is maximal. We can not only demand that it produce correct formulae for us to compute trajectories with, we can look inside the formal system and ask what each symbol stands for, learning for example that "2" stands for the Platonic two. The axioms correspond to true facts in the universe, and the rules of inference preserve truth. In short, there is a correspondence between each gear and something in the world.

There is a serious problem when we try to carry this level of commitment over to the inference component of an NLP system. It is probably impossible to axiomatize any complex domain consistently in a way that would be useful. Achieving a consistent axiomatization of set theory required some of the world's greatest minds working for half a century, and set theory is about the simplest domain imaginable. When we get to domains as complex as the entities occupying our world, our management of the events in our lives, and social relationships, there seems to be no hope for a consistent axiomatization. For example, we would like our system to know that birds can fly without worrying each time about all the circumstances in which they can't. We frequently have and use contradictory general principles:

> Haste makes waste.
> A stitch in time saves nine.

When we come to axiomatizing our knowledge about social relationships, there is enough variation among people that there are probably no useful statements that are universally true.

There are several possible approaches to this difficulty. One is to try to axiomatize complex domains consistently anyway, such as Pat Hayes is doing in his interesting work on naive physics (Hayes 1978b). Another is to develop nonmonotonic logics (McDermott and Doyle 1978), in which there are axioms of the form

$$p(x) \;\&\; M\, q(x) \dashrightarrow q(x).$$

That is, if p(x) is true and it is consistent to assume q(x), then q(x) is true.

These researchers' efforts are still tentative and it is not clear whether they can succeed in a way that will satisfy our criteria of a formalism, closeness to English and a simple syntax. While I wish them well, I am skeptical about their prospects, and being impatient like most people in AI, I want to get on with the problems of primary interest to me in the meantime.

So I would like to see mechanisms for dealing with inconsistent sets of axioms. Selective inferencing can be just such a mechanism. We need it anyway to avoid being swamped by irrelevant inferences. We may as well use it to avoid inconsistencies, by building into the discourse operations the feature that they back away from placing contradictory propositions in the spotlight. When about to place a proposition P in the spotlight, if not-P is already there, the system retracts one of the two inferences, presumably on the relative strengths of what warrants each inference (cf. Carbonell 1979, de Kleer & Harris 1979, Doyle 1979). This is just a generalization of common practice in AI as old as Collins and Quillian (1971) who said that birds can fly but penguins can't.

Now that we have decided to allow inconsistent sets of axioms, let us look at a couple of extreme cases. We can not only include normative facts such as "Birds can fly," along with the marginal exceptions. We can include directly contradictory axioms,

$$(Ax)(\text{chair}(x) \dashrightarrow \text{has-arms}(x))$$
$$(Ax)(\text{chair}(x) \dashrightarrow \text{Not}(\text{has-arms}(x)))$$

The first axiom would be used for chairs whose arms were mentioned or implied, the second for chairs whose armlessness was relevant, and the discourse operations would insure that no chair was assumed to be both armed and armless.

Another example is Fregean set theory, which I've always felt was right, in some intuitive sense. It is too compelling on other grounds for Russell's paradox to be more than an obscure bug that we can live with. Frege's offending axiom schema, the principle of comprehension, says that for all predicates p,

$$(Ex)(Ay)(\text{member}(y,x) \leftrightarrow p(y))$$

Corresponding to every description p there is a set of entities satisfying that description. I think we use this axiom all the time, for example, in making sense out of

> Will all those who disagree please stand up and be counted.

We assume there is a set that can be counted that contains all those who satisfy the predicate "x disagrees". When we try to use the axiom schema for the predicate "y is not a member of itself" and spot the paradox, we simply shrug our shoulders and call it an exception, and so should our system.

All this, of course, makes it impossible to maintain the maximum level of commitment to our formalism. We have to pull back somehow. The extreme possibility is the clockmaker's stance. We are writing programs that will perform well linguistically and if our program does that you have no business looking inside at the gears. Ultimately, that is the only commitment I think we should accept. Just because I call one of my gears "bird" doesn't mean it has anything to do with real birds. As long as the mechanism says appropriate things about birds, you have no grounds for complaint.

But as Pat Hayes has pointed out (personal communication), it is highly unlikely that we could build such a mechanism with no more to go on. We need at least a heuristic commitment at a more detailed level. Very complex processing takes place in the system, and if we don't have some intermediate checks on the correspondence between the formalism and the world, it is unlikely that we will have the proper correspondence between the ultimate behavior and the world.

In order to state a reasonable heuristic commitment, we need to back up and state again what the problem is. I have frequently said that model theoretic semantics is a translation from some formal language, say, lambda calculus, into the language of set theory. Logician friends have objected vehemently, calling it a mapping from the formal language into things in the world. That raises difficulty for those of us who are methodological solipsists, verging on real solipsism. We can't make sense out of "things in the world" independent of some conceptual framework. The solution is to move up a level, climb to the top of Mount Olympus with Zeus. Zeus looks down and sees a society of people, two of whom we'll call the Maker and the Buyer; he sees a formal system that the Maker has just made and is trying to sell to the Buyer; and he sees a number of languages, or conceptual frameworks, that exist as "social objects" in this world and are used by the Maker and the Buyer for communicating. From this perspective, what looked like semantics when we were in the hurly-burly of earthly life, now looks like a translation from a representational system or conceptual framework at one level (the formal system) into a conceptual framework at the next level up (the society). The advantage of this perspective is that it allows us to talk about which of the conceptual frameworks at level two, the society, is the most appropriate target of the translation. The question is -- when the Maker makes a commitment to the Buyer about the nature of his formal system, what language should they use?

Model theoretic semantics assumes that the target should be the language or conceptual framework of set theory. This works well for implication and quantification, and thus is an

appropriate choice for a deductive system in mathematics. But it sheds little light on the bulk of what we talk about in natural language. It does little good to be told that

America is the bastion of democracy

means that the ordered pair of America and democracy is in the set of ordered pairs denoted by the relation "is the bastion of".

A common way around this objection is to enrich one's ontology with received theories and use the entities they provide as targets of model theoretic interpretation. For example, one might attempt to map our language of spatial relationships into coordinate geometry, our language of motion into Newtonian physics, and presumably our language of beliefs and emotions into Freudian psychology. That is, one candidate for a target representation is a "suitably enhanced" set theory.

Another perfectly viable candidate is the conceptual framework provided by natural language itself. All of us, as Makers and Buyers, are just as comfortable in natural language as we are in set theory. The usual objection to natural language is that it is frequently ambiguous, so we will modify our candidate to "suitably unambiguous" English.

The choice is thus between a suitably enhanced set theory and suitably unambiguous English. Suitably enhancing set theory is a project science has been embarked on for centuries and will continue on for centuries to come. Suitably disambiguating English, on the other hand, is something all of us do all the time. So the latter choice seems the more judicious one. Moreover, since it is a mechanism for simulating natural language behavior that the Maker and Buyer are discussing, the correspondences will be easier to state with the latter choice, for natural languages and scientific theories tend to carve nature at very different joints.

The intermediate check, the heuristic commitment, that I as the Maker am willing to give to the Buyer then is this: I claimed a fairly straightforward translation between expressions in the formal language and English sentences. If in the course of processing a sentence in a text, the discourse operations place an expression in the spotlight, I will feel committed to it in the following sense: Translate the expression into English and put it in the frame "You mean ...?" The resulting utterance should be an appropriate request for clarification in the circumstances and should be answered positively. Suppose the system is told "John is a hog." Then it is appropriate for it to draw the inference

overconsume(J,F) & food(F),

since the response "You mean John overconsumes food?" would usually be answered positively. It would not be appropriate for it to draw the inference

four-legged(J),

since the response "You mean John has four legs?" would be answered negatively. Similarly for an expression that was read in as part of the text, if the requirements of the discourse operations have been satisfied and any necessary modifications made, then I am willing to make the same commitment. Turn it, as modified, into an English request for clarification and the answer should be "yes". The commitment associated with including an axiom in the knowledge base is that there is some situation in which the discourse operations should draw that inference and place it in the spotlight, with the associated commitment.

Rather than map the formal language into a scientific language at the level of the social enterprise and judge truth, we map the formal language into natural language at the level of the social enterprise and judge appropriateness.

## 3. Summary: What Are the Real Problems?

Here's a possible program for advancing the study of language. Take our ten linguistic problems and give one each to our ten best graduate students, and let them work for several years. The arguments of this paper lead to a prediction of the outcome. Ten percent explorations of aspects special to the particular problems, ninety percent overlap. The space of what we don't know about language would have been cut along the wrong axis.

I have suggested another way of partitioning the space. In this, the problems are devising a language of representation, encoding various chunks of world knowledge, refining the statements of the discourse operations, and discovering methods of control. The first of these is probably the least interesting; most work on the problem has just resulted in a reinvention of predicate calculus. This leaves three problems to focus on.

In the project of encoding knowledge, we should concentrate on those basic areas that lie at the core of language and occur in almost all discourse -- space, the meaning of prepositions (cf. Herskovits 1979), time, tense and aspect, naive physics (Hayes 1978a, 1978b), belief (Moore 1979), number, and so on. A difficulty with such efforts in the past has been that definitions of terms won't stay constant under shifts of context. Ways of representing normative knowledge and mechanisms for context-dependent selective inferencing, such as presented here, should ease this difficulty. Much work has been done on these areas already in AI, but there is a common pitfall here. Many such efforts fail to pass the Grandmother Test. They tell us no more than our grandmother could have told us. There may be domains in which all there is to do is write down the obvious, but it's probably best not to waste our time on those domains for now. Our approach to the basic areas can be sharpened by the existence of several simple but challenging puzzles, such as McCarthy (1979) has suggested for

the domain of belief. A few good puzzles in interpreting linguistic expressions would also contribute.

The program indicated by the discourse operations has two stages. In the first stage, various linguistic phenomena are reduced to one or more of the five discourse operations. The ten linguistic phenomena discussed in this paper by no means exhaust the field. There are also the problems of interpreting adverbials, resolving quantifier and conjunction ambiguities, and resolving "one" anaphora (Webber 1978), to name just a few. The second stage is to define the discourse operations in terms of the selective inference process. The definitions I have given are only first approximations. For predicate interpretation, we need methods for computing the requirements of complex predicates, such as "x sell y for 57 3/8". For textual coherence, we need to search for further constraints on the definitions of the coherence relations. For the probem of relating the text to the world, the time is probably ripe for someone to look at the diverse work that has been done in the field and summarize it all in a coherent framework, characterizing in the most general terms the structures used to model the world, and the procedures used to relate various aspects of English discourse to the models. Such a consolidation would give us a plateau from which to advance.

For control, we need to follow out the three leads that were suggested -- salience measures, using redundancy, and metarules -- as well as other leads, such as parallel search mechanisms (Fahlman 1979). The idea of using redundancy is particularly intriguing, because it involves exploiting an inherent property of natural language to solve the search problem. It would be good to find other inherent properties of natural language with similar computational payoffs.

## Acknowledgments

I have profited from discussions with Armar Archbold, Herb Clark, Dave Evans, Chuck Fillmore, Barbara Grosz, Pat Hayes, Stan Rosenschein, the members of the Bay Area Discussion Group on Semantics and Pragmatics 1978-9, and the members of the TINLUNCH Discussion Group at SRI. This work was supported by the National Science Foundation under Grant No. MCS-78-07121 and by the Defense Advanced Research Projects Agency under Contract No. N00039-79-C-0118 with the Naval Electronic Systems Command.

## REFERENCES

Allen, J. 1979. A plan-based approach to speech act recognition. Technical Report No. 131/79, Dept. of Computer Science, University of Toronto.

Carbonell, J. 1979. Subjective understanding: Computer models of belief systems. Ph.D. Thesis. Yale University.

Charniak, E. 1974. Toward a model of children's story comprehension. AI-TR-266, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Clark, E. & H. Clark 1980. When nouns surface as verbs. Language, Vol. 48, 767-811.

Collins, A. and M.R. Quillian. 1971. How to make a language user. In Tulving and Donaldson Eds. Organization of memory, 309-351.

Davis, R. 1977. Interactive transfer of expertise: Acquisition of new inference rules. Proceedings, International Joint Conference on Artificial Intelligence, 321-328, Cambridge, Massachusetts. August 1977.

de Kleer, J. & G. Harris 1979. Truth maintenance systems in problem solving. Paper delivered to Artificial Intelligence Workshop, Electrotechnical Laboratory, Tokyo, Japan. August 1979.

Downing, P. 1977. On the creation and use of English compound nouns. Language, Vol. 53, No. 4, 810-842.

Doyle, J. 1979. A truth maintenance system. AI-TR-521, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Fahlman, S. 1979. NETL: A System for Representing and Using Real-World Knowledge. Cambridge, Mass.: MIT Press.

Fillmore, C. 1979. Innocence A second idealization for linguistics. Proceedings, Fifth Annual Meeting, Berkeley Linguistics Society, 63-76. Berkeley, California.

Grosz, B. 1977. The representation and use of focus in dialogue understanding. Stanford Research Institute Technical Note 151, Stanford Research Institute, Menlo Park, CA. July 1977.

Grosz, B. 1980. Focusing and description in natural language dialogues. In A. Joshi, I. Sag, & B. Webber (eds.), Elements of Discourse Understanding: Proc. of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting. Cambridge University Press, Cambridge, England.

Hayes, P. 1978a. The naive physics manifesto. Working Papers. Institute for Semantic and Cognitive Studies. Geneva.

Hayes, P. 1978b. Naive physics I: Ontology for liquids. Unpublished manuscript. August 1978.

Herskovits, A. 1979. Space, prepositions, and situation types. Unpublished manuscript. October 1979.

Hobbs, J. 1976a. Pronoun resolution. Research Report 76-1, Department of Computer Sciences, City College, City University of New York. August 1976.

Hobbs, J. 1976b. A computational approach to discourse analysis. Research Report 76-2, Department of Computer Sciences, City College, City University of New York. December 1976.

Hobbs, J. 1978. Why is discourse coherent? SRI Technical Note 176, SRI International, Menlo Park, California. November 1978. To appear in F. Neubauer (Ed.) Coherence in natural language texts.

Hobbs, J. 1979a. Coherence and coreference. Cognitive Science. Vol. 3, No. 1, 67-90.

Hobbs, J. 1979b. Metaphor, metaphor schemata, and selective inferencing. SRI Technical Note 204, SRI International, Menlo Park, California. December 1979.

Hobbs, J. forthcoming. Representing beliefs about belief flatly.

Hobbs, J. & J. Robinson 1979. Why Ask? Discourse Processes, Vol. 2, 311-318.

Joos, M. 1972. Semantic axiom number one. Language, Vol. 48, 257-265.

Levi, J. 1978. The syntax and semantics of complex nominals. New York: Academic Press.

Lockman, A. 1978. Contextual reference resolution in natural language processing. Ph.D. thesis, Department of Computer Science, Columbia University. May 1978.

Mann, W., J. Moore and J. Levin 1977. A comprehension model for human dialogue. Proceedings, International Joint Conference on Artificial Intelligence, 77-87, Cambridge, Mass. August 1977.

McCarthy, J. 1979. Formalization of two puzzles involving knowledge. Unpublished manuscript.

McDermott, D. and J. Doyle 1978. Non-monotonic logic I, A.I. Memo 486, Artificial Intelligence Laboratory, Massachusetts Institute of Technology. August 1978.

Moore, R. 1979. Reasoning about Knowledge and Actions. Ph. D. Thesis. Massachusetts Institute of Technology. Cambridge, Mass.

Nunberg, G. 1978. The pragmatics of reference. Ph.D. thesis. City University of New York, New York, NY.

Robinson, A. 1980. The interpretation of verb phrases in dialogs. SRI Technical Note 206. SRI International, Menlo Park, California. January 1980.

Schank, R. and R. Abelson 1977. Scripts, plans, goals, and understanding. Hillsdale N.J.: Laurence Erlbaum Associates.

Schank, R., M. Lebowitz, & L. Birnbaum 1980. An integrated understander. American Journal of Computational Linguistics, Vol. 6, 13-30.

Webber, B. 1978. A formal approach to discourse anaphora. BBN Report No. 3761, Bolt. Beranek, and Newman Inc. Cambridge, Mass. May 1978.

Wilensky, R. 1978. Understanding goal-based stories. Yale University Dept. of Computer Science Research Report 140. September 1978.

# INTERPRETING VERB PHRASE REFERENCES IN DIALOGS

Ann E. Robinson
Artificial Intelligence Center
SRI International
Menlo Park, California 94025

## ABSTRACT

This paper discusses two problems central to the interpretation of utterances: determining the relationship between actions described in an utterance and events in the world, and inferring the "state of the world" from utterances. Knowledge of the language, knowledge about the general subject being discussed, and knowledge about the current situation are all necessary for this. Presented and discussed are the kinds of knowledge necessary for interpreting references to actions, as well as algorithms for using that knowledge in interpreting dialog utterances about ongoing tasks and for drawing inferences about the task situation that are based on a given interpretation.

## I INTRODUCTION

Two problems central to the interpretation of utterances are determining the relationship between actions described in an utterance and events in the world, and inferring the "state of the world" from utterances.* Knowledge of the language, knowledge about the general subject being discussed, and knowledge about the current situation are all necessary for this. The problem of determining an action referred to by a verb phrase is analogous to the problem of determining the object referred to by a noun phrase. Although considerable attention has been given to the latter (Donellan, 1977; Grosz, 1977a, 1977b; Sidner, 1979; Webber, 1979), little has been done with the former.**

The need to identify an action is obvious in utterances containing verbs like "do", "have", and "use", as in "I've done it", "what tool should I use?", or "I have it". In these utterances the verb does not name the action, but rather refers to it more generally, much as pronouns or "nonspecific" nouns (e.g., "thing") refer to objects. Even when more specific verbs are used, complex reasoning may be required to ascertain the particular action being referred to. For example, the utterance "I've glued the pieces together" can refer to different steps in a task--depending on what objects "the pieces" refers to, because each gluing action is a different step in the task (Werner, 1966). Similarly, the verb "cut" refers to different types of cutting actions when used with different objects, as in "cut grass", "cut wood", or "cut cake" (Searle, 1978).

This paper presents algorithms that combine knowledge about language, the problem domain, and the dialog itself to interpret references by verbs. The algorithms have been implemented and tested in a computer system (TDUS) that participates in a dialog about the assembly of an air compressor (Robinson, 1980). The system acts as an expert, guiding an apprentice through the steps of the task.

## II KNOWLEDGE NEEDED

Interpreting any utterance and relating it to a task requires knowledge about the language and the task, as well as the relationships between them. This paper will outline briefly some of the knowledge needed to identify actions. The research builds directly on the concepts of global and immediate focusing, through which certain entities are highlighted (Grosz, 1977a, 1977b; Sidner, 1979). General familiarity with that research will be assumed. More detailed descriptions of other aspects of the knowledge needed for interpreting utterances can be found elsewhere (Grosz, 1977a; Hendrix, 1977, 1979; Robinson, 1980; J. Robinson, 1980).

### A. Actions and Goals

Interpreting verbs requires knowing about events that have occurred, are occurring, or can occur in the domain. Knowledge about events typically includes the steps necessary to perform the actions associated with the events, the possible participants, the conditions that must be true before the actions can be performed, and their results (e.g., the goals they achieve or their possible side effects). Knowledge about actions and events includes both general knowledge about possible actions and events and more specific knowledge about those that occur during a particular task.

A recently developed formalism, process models (Grosz et al., 1977; Appelt et al., 1980) is used in TDUS for encoding information about actions. The knowledge encoded includes a

---

* This research has been funded under three-year NSF Continuing Research Grant No. MCS76-22004. This paper and the research reported in it have benefited from interaction with all the members of the natural-language research group at SRI. Barbara Grosz, Jerry Hobbs, Gary Hendrix, and Jane Robinson have been particularly helpful in the preparation of this paper.

** A problem related to identifying verb phrase referents--interpreting verb phrase ellipsis--has been investigated by Webber (1979).

specification of hierarchical decomposition of actions into subactions, as well as a description of individual types of actions. The representation is an extension of the network formalism used for representing other knowledge about objects and relationships, as described by Hendrix (1979).

Related to knowledge about current actions is knowledge about the goals of the dialog participants. The goals that are expressed or implied by an utterance can be of many types. These include "domain goals" related to the subject domain; "knowledge-state goals" related to changing the knowledge of one or more of the dialog participants; and "social goals" arising from both the social context in which a dialog takes place and the interpersonal relationships of the dialog participants.*

The goals of dialog participants affect the interpretation of verbs in at least two ways: (1) interpreting verbs entails recognizing the speaker's goals as expressed or implied by the utterance; (2) current goals are a part of the context within which verbs are interpreted.**

The TDUS system handles two kinds of goals: domain goals and certain knowledge-state goals. Domain goals concern states to be achieved by task-related actions in the domain, while knowledge-state goals concern states to be achieved by acquiring a specific piece of information.

Figure 1 illustrates the relationship between actions and goals. The hierarchy shown is a simplification of a portion of the assembly task hierarchy currently encoded in TDUS.*** Each node represents an action and its associated goal. The hierarchy encodes the substep relationships: child nodes represent substeps of their parent nodes. The top-level node in the tree, node (1), represents the action of attaching a pump whose associated goal is that the pump be attached. Nodes (2) and (3) represent substeps of this attaching process--the actions of positioning the pump and tightening the bolts, with the associated goals that the pump be positioned and that the bolts be tight. The action of locating bolts, represented by node (4), is not an explicit step in the task, but is necessary for its performance. Node (4) has an associated knowledge-state goal: "know the location of the bolts". All these goals have associated actions that, in the process model formalism, are specific instantiations of actions, not action schemata.

---

* The distinction between domain and knowledge-state goals was drawn by Appelt (1979).

** The current implementation of goals in TDUS is an extension and partial revision of one by Sidner, described in her dissertation (1979).

*** Although the assembly task currently encoded in TDUS involves strong structuring of actions and goals, our representations and procedures are applicable to less structured domains.

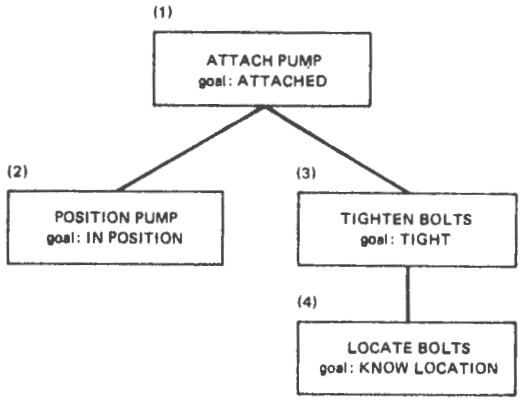



Figure 1 Goal/Action Tree

We distinguish two classes of goals: "direct goals" achieved by actions the apprentice has explicitly or implicitly said are being performed now or have been performed; "potential goals" mentioned by either participant that have not been acted upon but might possibly be. In the context of the task steps shown in Figure 1, "I am attaching the pump" states that the speaker is performing an attaching action represented by node (1). Thus the utterance establishes that the pump be attached as a direct goal. "Should I tighten the bolts?" indicates that the speaker might perform the tightening action represented by node (3) and thus establishes that the bolts be tight as a potential goal.

A knowledge-state goal can also be a direct goal. For example, the goal associated with node (4), "know the location of the bolts", can be introduced by utterances like "where are the bolts?" In the current implementation, knowledge-state goals cannot be potential goals. This limitation arises primarily because knowledge-state goals have not been as fully explored as task goals.

Direct and potential goals are distinguished from one another because of the different roles they play in the interpretation of verbs. Basically, direct goals are those that are known as existing or former goals associated with actions that are being or have been performed. Potential goals are possible near-term goals associated with possible future actions. Depending on the type of utterance, one or the other class of goal might be considered first. The different roles of the two goal classes will be illustrated when the interpretation of verbs is discussed in detail below.

Besides recognizing a goal, it is necessary to recognize whether the goal is the current one, one that has already been achieved, or one that has been abandoned. It is also necessary to recognize when goals are no longer potential.

A direct goal is assumed to be current when an utterance states that an action that will achieve the goal is in progress. A goal is assumed to have been achieved when one of the following conditions has been satisfied:

(1) An explicit statement such as "I have attached it" or "I'm done" or "OK"* indicates completion of the action that achieves the goal.

(2) An explicit statement indicates completion of an action intended to achieve the goal.

(3) The start of a new action implies completion of its predecessor and thus achievement of the associated goal.**

An utterance such as "never mind" is interpreted as signaling the abandonment of a goal.

Potential goals are not achievable as such. Rather, they can either become direct goals through the mechanisms for establishing direct goals or disappear when a new potential goal is recognized.

B. Knowledge about Language

Knowledge about language is also required, encompassing what is generally characterized as syntactic, semantic, and discourse knowledge. Syntactic and semantic knowledge includes knowledge about tense and aspect and about the relationship between words/phrases and domain entities.

Discourse knowledge is knowledge about how the domain and dialog contexts in which an utterance occurs contribute to and are influenced by the interpretation of the utterance.

A key element of discourse knowledge is knowledge about discourse focus, through which the participants in a dialog focus their attention on only a small portion of what each of them knows or believes. As a dialog progresses, the participants continually shift their focus and thus form an evolving context within which utterances are produced and interpreted. This research builds directly upon the concept of discourse, or global, focusing (Grosz, 1977a, 1977b, 1980).

In addition to global focusing, we have built upon the concept of immediate focus (Sidner, 1979) through which a single discourse entity is isolated. This is a more localized focusing phenomenon that is closely related to the use and recognition of anaphora, as well as to changes in global focusing.

---

* See the discussion in Grosz(1977a) of the roles of OK.

** As Sidner(1979) points out, in the first two cases the information comes from the utterance, while in the third case it is from the task model.

C. Shared and Joint Knowledge

In our framework, the dialog participants are assumed to share knowledge about processes in the task model*** and the history of the task performance to date, along with knowledge about direct and potential goals and focused entities. We view this shared knowledge as composed of at least two parts: (1) the processes in the task model and the history of its performance comprise knowledge about the world that is assumed to be shared by the participants independently of the dialog; (2) knowledge about the goals and focusing, which is assumed to be shared as a result of the dialog. We will distinguish these two types of shared knowledge and their roles in the interpretation of utterances, and use the terms shared and joint to refer to them.

We use shared knowledge to refer to what is known by both participants because of their common background and experiences, and is assumed by them to be shared, but has not been explicitly discussed by them. This includes knowledge of both language and the domain.

We use joint knowledge to refer to what has been explicitly communicated between the dialog participants. The steps of the task that are explicitly mentioned are joint knowledge, as are other focused entities that have been mentioned. Since we are considering dialogs in which the only mode of communication is verbal (there is, for example, no shared visual context) only what is actually said is assumed to be known jointly.

This analysis identifies as "joint knowledge" essentially what Clark and Marshall (1980) characterize as the mutual knowledge that results from "linguistic co-presence." Our use of the term "shared knowledge" covers the mutual knowledge they describe as resulting from "cultural co-presence" and a limited form of "physical co-presence".****

Assumptions about things that are jointly known play a critical role in the interpretation and production of utterances (Clark and Marshall, 1980), as the use of anaphora illustrates. Pronouns and pro-verbs (when used felicitously) always refer to jointly known concepts, so that any utterance containing a pronoun or pro-verb must drawn upon joint knowledge.

---

*** Note that the apprentice knows neither all the steps in the task nor their ordering--otherwise there would be no need for the expert. However, the apprentice does know how to perform most of the basic actions, such as bolting and tightening.

**** Physical copresence is limited by the sensory constraints of the computer system. The system can assume that both it and the apprentice are aware of the physical situation, but it can verify its assumptions only on the basis of the apprentice's actual utterances in the dialog.

## III INTERPRETING VERBS

In this section we address issues that arise in using domain and linguistic knowledge to interpret verbs and to infer the current situation on the basis of that interpretation.

The possible referents of a verb phrase are constrained by both the context and the utterance itself. Coordination of the constraints is necessary for interpreting verbs in a computer system.

Contextual constraints are derived from two sources: the domain and the dialog. Knowledge about the domain and, in particular, the task being performed, is part of the knowledge shared by the participants at the beginning of the dialog, including knowledge as to which actions can be performed, how to perform them, and when. The dialog provides knowledge about the actual progress of the task; it causes certain entities to be focused, as well as providing information about the goals of the participants. This knowledge is the joint knowledge we described previously.

Utterance constraints include tense and aspect information and the type of action denoted by the verb. The tense and aspect of the utterance restrict the alternatives within the task model and limit the goals that might be considered as referents. Generally, present tense and progressive aspect are used when referring to a new action, indicating that it has been started. Only if the utterance is somehow marked, as in "I'm still tightening the bolts", will the reference be to an action that has already been mentioned as in progress. Consequently, when TDUS is interpreting a present and progressive utterance, the actions considered in the task model are those closely related to the most recent action performed. The only goal considered is the potential one since a direct goal is associated with an action already under way.

Past tense and/or perfective aspect indicate that an action has been finished. However, the hearer may or may not have known that the action was in progress. Consequently, the actions known to have been in progress and those that can be subsequent steps are possible referents, as are actions associated with all direct goals and the potential goal.

The search for the referent of a verb can be conducted either top-down or bottom-up. The top-down search uses contextual constraints to find the place in the task that the utterance fits and utterance constraints to limit alternatives. The bottom-up mode uses information from the utterance, such as verb type, to find its relationship to the task. If the top-down search is successful, the action and its place in the task are identified simultaneously.

In the current domain, in which all the utterances are directly related to the task and in which the system has already encoded all the relevant steps to be performed, top-down constraints are strong enough to allow a top-down search to be conducted first--and only if that fails is a bottom-up search conducted. In a domain where there is less structure provided by the task, a bottom-up search will clearly play a more central role. This search can be improved by doing more extensive reasoning based on the verb in the utterance. For such domains, we have been examining what other linkage between actions should be introduced.

One of the major limitations of previous natural-language systems has been a lack of coordination of the strategies for identifying referents of noun phrases and pronouns with one another or with the interpretation of the verb. In fact, except for the pronoun resolution procedure that used a very simple goal recognition algorithm (Sidner, 1979), the verb phrase was not even taken into account. However, since the interpretation of each of these utterance elements cannot be carried out in isolation, the previous strategies have been modified and now the procedures for identifying noun phrase and pronoun referents are coordinated with the search for the verb phrase referent. Details of this modified strategy will be discussed in conjunction with elucidation of the verb phrase strategy.

### A. The Top-Down Algorithm

Different types of utterances can draw upon different contextual constraints. Three major factors are considered by the interpretation algorithm in determining which contextual constraints to draw upon. The factors are (1) whether or not a pronoun is present in the utterance; (2) whether or not all the noun phrases in the utterance refer to focused entities; (3) whether or not the main verb is "do". The presence of a pronoun indicates that joint knowledge, particularly goals and immediate focus, is being drawn upon. If no pronoun is present, other factors weigh more heavily in determining constraints. When all the definite noun phrases refer to focused entities, focusing information is also a key in interpreting the verb. If the referents are not focused, knowledge about the task and its structure must be used. When "do" appears as the main verb, joint knowledge plays a more central role than when other verbs are used. The particular usage of "do", as signaled by the other constituents, indicates which aspects of joint knowledge are most important.

We will discuss the interpretation algorithm by examining the interpretation of utterances resulting from various combinations of these factors. The utterances we will discuss are those containing the verb "do", those containing verbs other than "do" plus pronouns, and those containing verbs other than "do" plus definite noun phrases.

Within the first type of utterance--those containing "do"--we further distinguish utterances like "I've done it" from utterances like "I've done the screws." In the former, "do" refers to the performing of an action, "it" to the action itself. In the latter, "do" refers to a particular action, such as remove. Our discussion will first cover these two types of utterances containing "do", then utterances with other verbs and pronouns, finally utterances with other verbs and definite noun phrases.

1. Do and Pronouns

In interpreting verb phrases such as "do it", knowledge about the context is used first to determine possible referents. If "it" has been used felicitously, it must refer to an action jointly known to the dialog participants. As we have discussed, joint knowledge in TDUS is represented by goals and focusing. Goals are a subset of all focused entities and, by definition, those actions that could possibly be performed by the apprentice. Consequently, possible referents are contained in the subset of joint knowledge represented by the most current direct goals and by the potential goal.

The main utterance constraints are derived from tense and aspect, which, as we have observed, limit the goals whose associated actions could be referents. The three cases we distinguish are past tense, present tense and imperfective aspect, and future tense.

As we have discussed, direct and potential goals can be referred to in a past-tense utterance. For such utterances, the algorithm examines the most recent direct goal first. If it is associated with a domain action (i.e., not a knowledge-state goal), the action is taken to be the referent of "it" because that is the action known to be in progress. Utterance (3) illustrates such a reference to a task goal.

(1) A: I'm doing the brace now.
(2) E: OK
(3) A: I've done it.

Here "it" refers to the action of installing the brace, the action associated with the current goal.

Because of restrictions in our current implementation, the most recent direct goal is not considered as a referent if it is a knowledge-state goal. Instead, the action associated with the potential goal is taken to be the one referred to, since it is always a domain action. Clearly, if potential goals were extended to include knowledge-state goals, a more sophisticated test would be required.

Utterances (4) through (10)--taken from an actual dialog with TDUS--illustrate reference to a potential goal.

(4) A: What should I do now?
(5) E: Install the aftercooler elbow on the pump.
(6) A: I've done it.
(7) E: OK.
(8) A: Should I install the aftercooler?
(9) E: Yes.
(10) A: I've done it.

The apprentice's Utterance (4) establishes a direct knowledge-state goal of knowing what action to perform, while the expert's reply establishes a potential goal that the aftercooler elbow be installed. Utterance (6) refers to the potential goal. Utterance (8) similarly establishes a direct knowledge-state goal of knowing about the action--in this case, whether the action is installation of the aftercooler; here the apprentice's utterance establishes the potential goal that the aftercooler be installed. Utterance (10) refers again to the potential goal.

An utterance that is present-tense and progressive (e.g., "I'm doing it") refers to an action that has been previously mentioned but only just started. As we have seen, a potential goal is associated with such an action, so that the latter is taken as the referent. For example, Utterance (10) could have been "I'm doing it", referring to the action of installing the aftercooler.

For a question referring to a future or a hypothetical action (e.g., "What should I do now?"), no attempt is made to identify the action as part of the interpretation. Instead, the reasoning process makes use of the task model to identify the appropriate reply.

2. Do and Definite Noun Phrases

For the other use of "do" (e.g., "I'm doing the screws"), where "do" refers to an action, an action of that type must be part of joint knowledge. However, only the action type may be jointly known and not the specific action referred to. For example in the sequence:

(11) A: I've attached the pump.
(12) E: OK.
(13) A: I'm doing the pulley now.

Utterance (11) makes joint the attaching action for the pump. In Utterance (13), "do" refers to another attaching action, but this one involves the pulley, and is thus a separate action. "Do" is not referring to the same specific action, but rather to the same type of action, i.e., "attaching".

To interpret such utterances, the contextual knowledge used is joint knowledge and knowledge about the task. The joint knowledge used is focusing information, because an action of the same type as the one referred to should be focused.* The interpretation algorithm scrutinizes focused actions for a type capable of having the newly mentioned participating objects. For example, the algorithm might find "attach pump" as a focused action, determine that it is an "attach"

---

* Goal information could be used by examining the types of the actions associated with domain goals. However, access to the action type is more direct through focusing information.

and that therefore a pulley can also participate in an "attach" action. If an action is found, task knowledge is used to determine if an action of that type with the indicated participants is an appropriate action in the current situation. Thus, if attach + pulley is an appropriate action, "attach pulley" is taken as the referent of "do".

Tense and aspect information from the utterance help determine which actions in the task model are appropriate. As we noted, a present-progressive utterance indicates initiation of a new step, whereas the past tense could be used with either a new step or one in progress.

Utterances (14)-(16)

(14) A: Should I install the pulley now?
(15) E: No.
The next step is:
Install the aftercooler elbow on the pump.
or
Install the brace on the pump.
(16) A: I'm doing the brace now.

illustrate a related situation. Here two steps have been mentioned and are essentially equally focused and both potential goals, so "do it" could not refer unambiguously to one of the actions. However, both actions are "install" actions, so "do" can refer to an "install" type of action. The interpretation algorithm outlined above works for this case as well.

### 3. Pronouns with Verbs Other Than Do

For utterances containing verbs other than "do" and pronouns, contextual constraints also stem from joint knowledge, since the object or objects referred to by the pronoun must be joint knowledge--in our case, mentioned in the dialog. The way the referent of the pronoun was introduced into the dialog affects the interpretation of utterances that contain pronouns. The distinction we make is whether the object was mentioned as a participant in an action comprising part of the task, (e.g., "I attached the pump.") or was not mentioned as a participant in an action (e.g., "Where is the pump?"). In the first case, if the object has been mentioned as participating in an action, the action will be recognized as a direct or potential goal and all its participating objects will be focused. In the second case, if no action has been mentioned but the object is a participant in some task action, the action will be inferred through the potential-goal recognition mechanism and thus become a potential goal. However, in this case only the object mentioned will be focused and not the other participants in the action. An example of the second case is:

(17) A: Where are the bolts?
[Immediate focus = bolts]
[Potential goal = THE BOLTS ARE BOLTED]
(18) E: OK
(19) A: I've tightened them with the wrench.
[with the wrench not in focus]

In this situation, the first reference to the bolts has established the potential goal that the bolts be bolted.

In both these situations the object mentioned is focused and, when appropriate, an action it participates in is established as a goal. The difference between the two is whether the actions and the other participating objects are also focused. This difference affects the interpretation of successive utterances containing pronouns.

Three cases are distinguished in the algorithm: (1) If there is a pronoun and there are no definite noun phrases, the actions associated with the most recent direct goal and the potential goals are considered as possible referents of the verb, since either of the two cases described above could obtain. (2) If there are definite noun phrases, all of which refer to focused entities, then the actions associated with the most recent direct goal and the potential goal are the most likely referents. Since all the objects are focused, the action was presumably mentioned, as in the first case described above. (3) If a pronoun and definite noun phrases occur together, but not all of the latter refer to focused entities, then only an action associated with a potential goal is a possible referent. Since a direct goal associated with this object could not have been established, only the second case described above could obtain.

In all three cases, utterance information about tense, aspect, and action type (from the verb) is used either to verify that the action associated with the goal is a possible referent or to choose a matching action type among possible referents.

### 4. No Pronoun or Do

When there is no anaphora in the utterance, the contextual knowledge used for interpretation is provided by focusing and the task model. Focusing is used to determine the relationship between the utterance and focused entities, including the current action. The task model, including the record of task progress, is used to determine which actions can reasonably be talked about in the context. First, focusing information is used to determine if the referents of any definite noun phrases associated with the verb are currently focused.

#### a. All Noun Phrases in Current Focus

If the noun phrase referents are focused, it indicates that the action involves objects currently being discussed by discourse participants and that the action is related to the current step (because it involves the same objects). The task model provides information about actions the apprentice can perform and has performed. Tense and aspect information from the utterance and the verb type restrict alternatives within the task model.

As we discussed earlier, present-progressive utterances generally refer to newly

initiated actions. Thus, the actions in the task model considered are those closely related to the most recent action performed and that involve objects referred to in the utterance. Possible actions might be a substep of the last step started but not completed; the potential goal; a step, not involving any different objects, that is closely linked in the plan to the most recent step started or completed (i.e., a step that is a substep of or successor to the last step, or succeeds a parent of the last step).

For example, "I am attaching the pump" is a present-progressive utterance with a noun phrase referring to a focused object. In this instance, the pump-attaching step is a substep of the last step started--installing the pump.

For utterances that are past tense and/or perfective aspect, actions in the task model known to have been in progress and those that could be next steps are possible referents. The alternatives considered during interpretation are: a step in progress; the potential goal; a substep of the last step started; a substep of any step in progress; and a step closely linked to the last step started or completed. For example, "I attached the pump" refers to a completed action that was a step in progress--attach pump. The verb in the utterance "I've installed the pulley" refers to a completed action that was the next step to perform, but was not explicitly mentioned as having been started, i.e., install pulley.

b. Not all Noun Phrases in Current Focus

If the referents of the noun phrases are not currently focused, the focusing hierarchy is searched because it indicates previously focused objects that might become focused again. If the noun phrase referents are identified somewhere in the focusing hierarchy, the action named in the utterance is matched against any action occurring at that place in the hierarchy.

If the utterance contains noun phrases referring to objects participating in the action and those objects cannot be identified among focused entities, the actions associated with direct goals are eliminated as possible referents of the verb. This happens because all actions associated with direct goals have been mentioned, which has caused all their participants to be focused.

Possible referents of such verbs include: the action associated with the potential goal; a substep of the current step in progress; a substep of all the steps in progress (if the utterance is past and/or perfective); any action which can achieve some current goal (e.g., knowing a location -> found the object). Since the objects described in the noun phrases and the action both have to be tested when the substeps are examined, the algorithm first checks the objects described by the noun phrases to see if they are participants in any of the substeps (by looking at the binding space); and if so, it then examines the actions to ascertain whether one of them matches the input action.

B. Bottom-Up Search

Currently the bottom-up algorithm consists of a search for the most specific occurrence of an event in the model whose participants are compatible with those in the utterance. This strategy is being expanded to include a search for a more general event that can then be found in the task. This can be either the most specific event type compatible with all the elements in the utterance, or a more general or 'similar' event type that is both compatible and can be found in the task. An example of the first is an utterance containing "tighten the bolt". The verb "tighten" refers to a general tightening action that can have more specific applications--such as tighten screws, tighten bolts, etc. From the knowledge that one kind of tightening is bolt tightening and from the concomitance of "bolts" in the utterance, it can be inferred that the "tighten bolts" action is intended. In the second case, a more specific verb might have been used (e.g., bolt the pump) to mean securing the bolts. The verb "bolt" might be initially interpreted as referring to a specific action of tightening bolts. However, that might not be an explicit step in the task, but rather, perhaps, only some general securing step. From the bolting action and knowledge of the more general actions of which it is a subset (e.g., securing), the relation of that action to the task model can be found.

C. Setting Limits to a Search

Knowing when to stop searching for a referent of a verb is another important element of the interpretation process. In general, the extent to which a verb reference is interpreted depends on the type of utterance. For example, a verb may refer to an action that does not fit into the current task context, such as one that could not or should not be performed at that time. If the verb is contained in a question (e.g., "Should I cut the end off now?"), a reasonable assumption may be as follows: if the action cannot be identifies, it is not the appropriate one to take, as illustrated in Utterance (14). On the other hand, if the verb is contained in a statement (e.g., "I have cut off the end."), it is more important to identify the specific action performed, since a model of the current situation could not otherwise be maintained. Thus, any process for identifying a verb referent should be able to determine what resources it should expend in each situation.

Another factor to be considered when determining how much effort to expend in identifying the referent is the extent to which the speaker can be assumed to be cooperative, and, consequently, his or her utterances to be

relevant. If some fairly direct connection between the utterance, the task, and/or dialog context can be postulated, devoting more effort to the search for a connection is more reasonable than in a less task-oriented dialog, in which such a connection may not even exist. In the TDUS system it is assumed that the user is cooperative and that all his or her utterances are relevant. Thus, considerable effort is expended, when necessary, to relate a statement about an executed step to the task of which it is a part.

## IV FUTURE DIRECTIONS

In this paper, we have discussed the problem of identifying the actions and events referred to by verbs. In particular, we have considered dialogs about an ongoing task. We have examined some of the knowledge needed for identifying the actions and have presented a strategy for finding them. This problem is of interest both because it is an important part of interpreting utterances and because it illustrates the need for combining knowledge of many types in the course of that process.

The research discussed here shows how the knowledge about language and about the domain that is currently identified and represented in a computer system can be used when interpreting verbs. Important extensions of this research include determining: (1) how top-down and bottom-up searching can be combined more effectively; (2) on what basis decisions can be made to stop looking for a connection between an action and a plan; (3) what extensions of this algorithm are necessary for handling dialogs in which the lack of a strong model of the task being performed results in weaker top-down constraints. Further research on finding referents of verb phrases, building on the algorithm presented here, should contribute to solving the more general natural-language processing problems of determining what other knowledge is necessary for interpreting utterances and how that knowledge can be used most effectively.

## V REFERENCES


Appelt, D. E., 1979. Planning Natural Language Utterances to Satisfy Multiple Goals, Thesis proposal, Stanford University, unpublished.

Appelt, D. E., Grosz, B. J., Hendrix, G. G., and Robinson, A. E., 1980. "The Representation and Use of Process Knowledge". Technical Note 207. Artificial Intelligence Center, SRI International, Menlo Park, California.

Clark, H. H., and Marshall, C. 1980. "Definite Reference and Mutual Knowledge". In Elements of Discourse Understanding, A. K. Joshi, I. A. Sag, and B. L. Webber, eds., Cambridge University Press, Cambridge, England.

Donellan, K. S., 1977. "Reference and Definite Description." In Naming, Necessity, and Natural Kind., S. P. Schwartz, ed., Cornell University Press, Ithaca, New York.

Grosz, B. J., 1977a. "The Representation and Use of Focus in Dialogue Understanding." Technical Note 151, Artificial Intelligence Center, SRI International, Menlo Park, California.

Grosz, B. J., 1977b. "The Representation and Use of Focus in a System for Understanding Dialogues." Proceedings of the Fifth International Joint Conference on Artificial Intelligence, pp. 67-76, Cambridge, Massachusetts, 22-25 August 1977.

Grosz, Barbara J. 1980. Focusing and Description in Natural Language Dialogues. In Elements of Discourse Understanding, A. K. Joshi, I. A. Sag, and B. L. Webber, eds, Cambridge University Press, Cambridge, England.

Grosz, B. J., Hendrix, G. G., and Robinson, A. E., 1977. "Using Process Knowledge in Understanding Task-Oriented Dialogs." Proceedings of the Fifth International Joint Conference on Artificial Intelligence, pp. 90. Cambridge, Massachusetts, 22-25 August 1977.

Hendrix, G. G., 1977. " Some General Comments on Semantic Networks." Panel on Knowledge Representation, Proceedings of the Fifth International Joint Conference on Artificial Intelligence, pp. 984-985, Cambridge, Massachusetts, 22-25 August 1977.

Hendrix, G. G., 1979. "Encoding Knowledge in Partitioned Networks." In Associative Networks-The Representation and Use of Knowledge in Computers, N. V. Findler, ed., Academic Press, New York, New York.

Robinson, A. E., 1980. "Interpreting Natural-Language Utterances in Dialogs About Tasks." Technical Note 210. SRI International, Menlo Park, California.

Robinson, J. J., 1980. "DIAGRAM." Technical Note 205. SRI International, Menlo Park, Calif.

Searle, J. R., 1978. Literal Meaning. Erkenntnis, 13, pp. 207-224.

Sidner, C. , 1979. "Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse." Massachusetts Institute of Technology, Cambridge, Massachusetts, PhD Th.

Webber, B. L., 1978. "A Formal Approach to Discourse Anaphora," BBN Report No. 3761, Bolt, Beranek, and Newman Inc. Cambridge, Massachusetts.

Werner, O. 1966. Pragmatics and Ethnoscience. Anthropological Linguistics, 1966, 8.8 42-65.

# Correcting Misconceptions About Data Base Structure

Eric Mays
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104

## ABSTRACT

This paper presents a method for computation of intensional failures of presumptions in queries to a natural language interface to a data base system. These failures are distinguished from extensional failures since they are dependent on the structure rather than the content of the data base. A knowledge representation has been investigated that can be used to recognize intensional failures. When intensional failures are detected, a form of corrective behavior is proposed to inform the user about possibly relevant data base structure that is related to the failure.

## INTRODUCTION

In the course of interacting with a natural language data base query system a casual user may pose queries based on beliefs about the domain which are incompatible with those of the system. Kaplan [Kaplan 79] has investigated one such class of beliefs which can be computed from a query and corrected, namely, extensional failures of presumptions. This paper introduces another class, that of intensional failures of presumptions, outlines the kind of knowledge representation needed for their computation, and proposes an appropriate form of corrective behavior.

A presupposition is a proposition that is entailed by all the direct answers of a question(*). A presumption is either a presupposition or it is a proposition that is entailed by all but one of the direct answers of a question [Kaplan 79]. Hence, presupposition is a stronger version of presumption, and a presupposition is a presumption by definition. For example, question (1a) has several direct answers such as "John", "Sue", etc., and, of course, "no one". Proposition (1b) is entailed by all the direct answers to (1a) except the last one, i.e., "no one". Therefore, (1b) is a presumption of (1a). Proposition (1d) is a presupposition of (1c), since it is entailed by all of the question's direct answers.

1a) Which faculty members teach CSE110?
1b) Faculty members teach CSE110.
1c) When does John take CSE110?
1d) John takes CSE110.

Presumptions can be classified on the basis of what is asserted — i.e., an "intensional" statement about the structure of the data base or an "extensional" statement about its contents. Thus an extensional failure of a presumption occurs based on the current contents of the data base, while an intensional failure occurs based on the structure or organization. For example, question (2a) presumes propositions (2b), (2c), and (2d). Presumption (2b) is subject to intensional failure if the data base does not allow for the relation "teach" to hold between "faculty" and "course". An extensional failure of presumption (2b) would occur if the data base did not contain any faculty member that teaches a course. Also note that the truth of (2b) is a pre-condition for the truth of (2c).

---

(*) The complete definition of presupposition includes the condition that the negation of a question, direct answer pair entails the presupposition.

2a) Which faculty members teach CSE110?

2b) Faculty members teach courses.

2c) Faculty members teach CSE110.

2d) CSE110 is a course.

Although a presumption which fails intensionally will of neccesity fail extensionally, it is important to differentiate between them, since an intensional failure that occurs will occur consistently for a given data base structure, whereas extensional failure is a transitory function of the current contents of the data base. This is not meant to imply that a data base structure is not subject to change. However, such a change usually represents a fundamental modification of the organization of the enterprise that is modelled. One can observe that structural modifications occur over long periods of time (many months to years, for example), while the data base contents are subject to change over relatively shorter periods of time (hourly, daily, or monthly, for example).

The problem this paper addresses is the recognition of presumptions which fail intensionally. In that case, the failure should be communicated to the user and a form of corrective response produced which informs the user about the relevant data base structure.

## DATA BASE MODEL

A data base model based primarily on the entity-relationship model of Chen [Chen 76] with the addition of an inheritance hierarchy can be used to detect the intensional failure of a presumption. This model is similar to that proposed by Lee and Gerritsen [Lee and Gerritsen 78], which incorporates the generalization dimension developed by Smith and Smith [Smith and Smith 77] into Chen's model. Although Lee and Gerritsen, and Chen allow entities to participate in n-ary relationships, this discussion will be restricted to binary relationships. Entities participate in relationships along two orthogonal dimensions, aggregation (among dissimilar entities) and generalization (among similar entities), as well as having attributes that assume values. Along the generalization dimension an entity inherits the attributes and relationships of its super-entities. All individuals of a particular entity set are members of any of that set's super-entity sets. Some individuals in an entity set may be members of a sub-entity set, therefore participating in relationships of the sub-entity set and having attributes of the sub-entity set.

A simple subset operator is not adequate for generalization in this context however, as is illustrated by the following example. Consider the data base model fragment shown in figure 1. Entity sets are designated by ovals, aggregation relationships by diamonds, and generalization relationships by edges from the super-entity set to the sub-entity set. Here "men", "women", "faculty", and "students" are all subsets of "people", with "students" participating in a "take" relationship with "courses". From this it can be determined that a "take" relationship can exist between "men" and "courses", since it is possible that there are some "people" who are both "men" and "students". But by this same reasoning we may also assert that a "take" relationship might exist between "faculty" and "courses", which is certainly not the case in most universities. The essential difference that needs to be noticed is that a non-empty intersection is possible between "men" and "students" and is not possible between "faculty" and "students".

The incorporation of an operator that partitions an entity set into several mutually exclusive sub-entity sets eliminates this problem. This distinction can be made by prohibiting the traversal of a path in the data model that includes two entity sets which are mutually exclusive. Furthermore, the path in the generalization dimension is restricted to "upward" traversals followed by "downward" traversals. An upward (downward) traversal is from a sub-entity (super-entity) set to a super-entity (sub-entity)

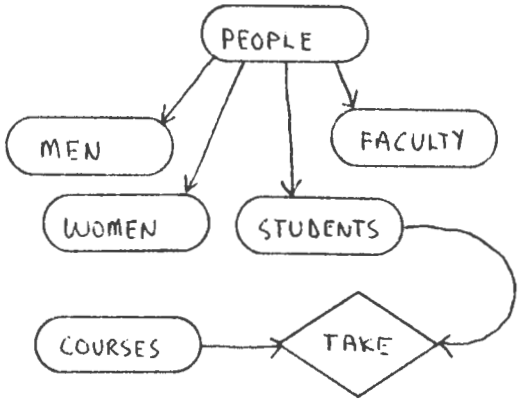



FIGURE 1

set. This restriction is made to prevent over-specialization of an entity set when traversing downward edges. The set of inferences that can be made in the presence of this restriction is not overly constrained, since any two entity sets that have a common intersection (sub-entity set) will also have a common union (super-entity set). As an example of this type of structure, consider figure 2, where partitioning is denoted by parallel arcs across edges. (Usually some attribute of an entity serves as the basis for the partition. For example, "sex" partitions "people" into "men" and "women".) In this fragment of information about university organization the possibility of a "take" relationship existing between "faculty" and "courses" is precluded by the fact that "faculty" and "students" are mutually exclusive. Observe that the path from "students" to "unemployed" would include "people" rather than "undergrads" or "unsupported". If either "undergrads" or "unsupported" were included, "students" would be unnecessarily restricted.

Although it might seem at first that a "teach" relationship might be possible between "undergrads" and "courses" -- since all "undergrads" are "students", and "students" and "teachers" are not mutually exclusive -- this is not the case. Closer inspection reveals that all "undergrads" are "unemployed", and "unemployed" and "teachers" are mutually exclusive, thus eliminating the possibility. The inferencing about mutual exclusion required to produce this result would proceed in a fashion similar to that proposed by Fahlman [Fahlman 79]. Very briefly, markers are propagated upward from the two entity sets which are assumed to be disjoint. If a split node (which denotes mutual exclusion) detects markers from both entity sets, they are not

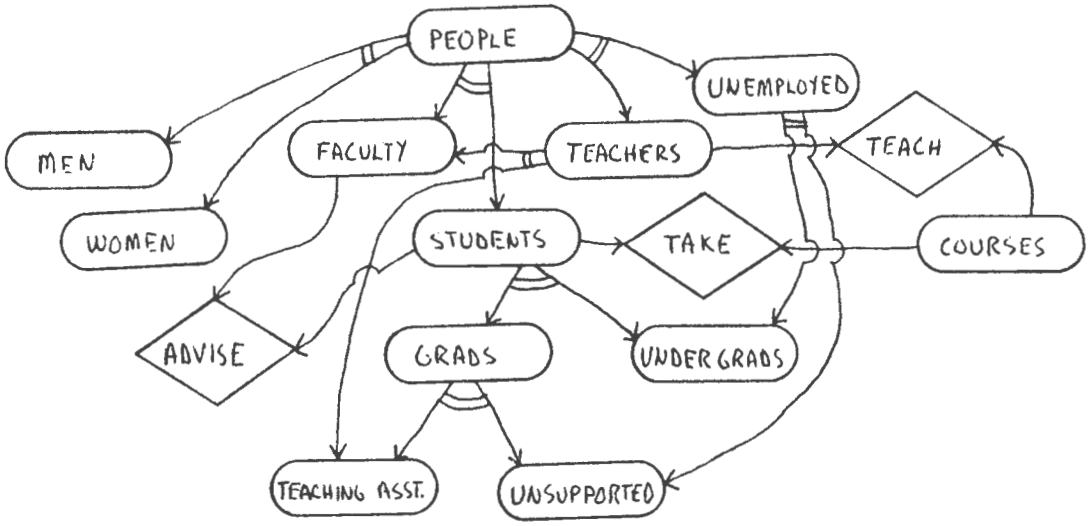



FIGURE 2

disjoint. Fahlman uses this operator to enforce restrictions on updates to a knowledge representation.

## INTENSIONAL FAILURE

In this data base model, intensional knowledge can be equated with the ability of an entity to participate in a relationship with another entity. Here, intensional failure occurs when such a relationship can not be established. For instance, the question "Which faculty take courses?" incorrectly presumes that a "take" relationship can exist between "faculty" and "courses" entities.

A method for the computation of a significant class of presumptions in the data base query domain is described by Kaplan [Kaplan 79]. The approach taken there involves the generation of the meta-query language (MQL) from the natural language input. The MQL is essentially a modified parse tree that closely reflects the surface structure of the input query. An example is shown in figure 3 for the question, "Which students in computer science took CSE110?". Kaplan computes the extensional failures of presumptions in a query from the MQL by checking the result of the formal data base query of each connected sub-graph of the MQL for emptiness. That is, the contents of the data base are accessed to determine if a presumption has a non-empty extension.

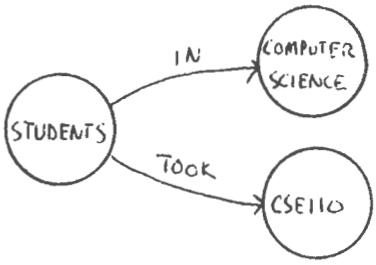



FIGURE 3

The intensional failure of presumptions in a query can be computed in a similar fashion. The essential difference being that the data model image of the MQL representation must be checked to insure that each relationship can be established in the data model. The data model image of a node or arc in the MQL is the entity set or relationship set, respectively, in the data model which is designated to contain the referent or set of referents for it. This is basically equivalent to disambiguating the lexical items, since the arcs and nodes in the MQL have lexical items associated with them. Consider the question, "Which faculty take CSE110 ?" and its corresponding MQL representation in figure 4. Here the entity set "courses" is designated as the data model image for "CSE110" since it is most likely to refer to a "course" entity. This query contains the presumption that "faculty take courses" which can be recognized as failing intensionally because a "take" relationship does not exist between "faculty" and "courses".

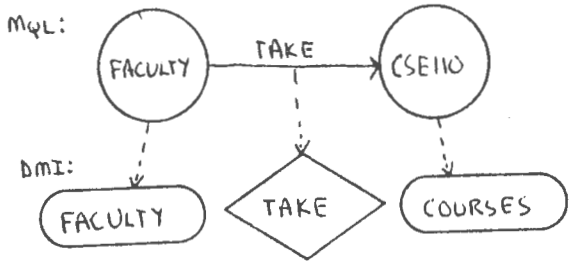



FIGURE 4

Recognizing the intensional failure of presumptions is only part of the problem -- it is also useful to provide the user information with respect to related intensional knowledge. Given a relation R, entities X and Y, and a failed presumption (R X Y), salient intensional knowledge can be found by abstracting on either R, X, or Y to create a new relation. For example, using the university data base model fragment, consider the following hypothetical exchange:

Q: "Which faculty take courses?"

A: "I don't believe that faculty can take courses.
Faculty teach courses.
Students take courses."

Here the presumption that faculty take courses can

be recognized as failing intensionally. This can be communicated to the user by paraphrasing its negation, noting as well what possible relevant relationships do hold.

## HIGHER ORDER FAILURES

A more complicated interaction of presumptions with the data model can also cause a presumption to fail intensionally. These failures occur in sub-graphs of the MQL which contain two or more arcs. It may be the case that a relationship can be established for each arc that connects two nodes in the MQL, but there is still a connected sub-graph (a presumption) that fails intensionally. The relationships in a particular sub-graph may impose restrictions on the nodes that will form empty response sets which can be recognized solely from intensional knowledge. An example of this is shown in question (3a). The restrictions on "teachers" involve two entities in the same partition. Question (3b) contains the same intensional failure. Both presume identical propositions, although in (3a) it is not as apparent.

3a) Which teachers that advise students take courses?

3b) Which teachers are both faculty and students?

A corrective response for this type of failure involves identifying the entities that participate in the relationships in addition to the failed presumption. In response to (3a), for example:

"Faculty advise students.
Students take courses.
I don't believe that a teacher can be both a faculty member and a student."

It doesn't appear that any related knowledge need be communicated, although some information regarding the various partitions of an entity set might be helpful. An adequate procedure for determining relevant knowledge along the generalization dimension has not been thoroughly investigated.

## RELATING RELATIONSHIPS

An interesting situation arises when attempting to determine related intensional knowledge for a failed presumption with regard to relationships. Consider an enterprise which has a matrix organization as in figure 5. The "in" relationships are conceptually similar but must be represented distinctly. The following behavior is desired for this data model:

Q: "Which employees are in areas?"

A: "I don't believe that employees are in areas.
Employees are in divisions.
Projects are in areas."

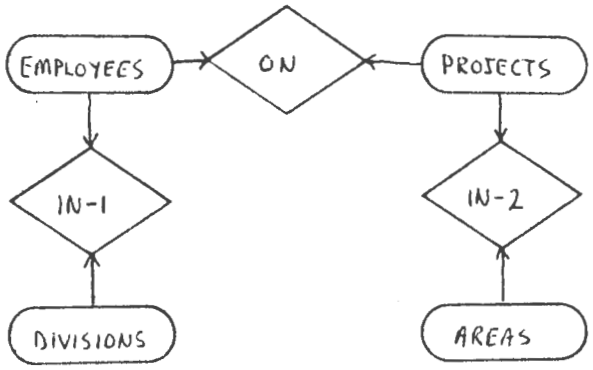



FIGURE 5

But this will not be achieved given the method outlined earlier of abstracting on one of R, X, or Y for a failed presumption (R X Y). If "in-1" is picked as the data model image for "in", the response will not include the fact that "projects are in areas". Similarly, if "in-2" is chosen, "employees are in divisions" will not be included. This can be remedied by introducing an operator (R-SET) which denotes the conceptual similarity of relationships as in figure 6. The procedure for determining salient intensional knowledge can be modified to include relationships in the same "R-SET" when abstracting on a relationship. Although this might appear ad hoc, it should be noted that this would be the first

step towards developing a hierarchy for relationships.

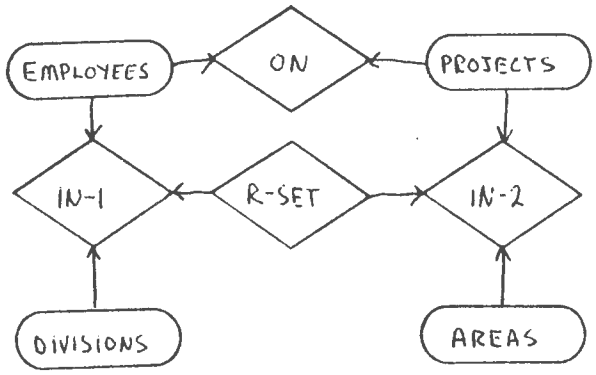



FIGURE 6

Note that there may be some basis for choosing the domain of a particular predicate from a semantic relatedness measure. For instance, if two distinct "teach" relationships existed, between "faculty" and "courses", and "grads" and "courses", the question "Which undergrads teach courses?" would indicate that the "teach" between "grads" and "courses" should be chosen.

## CONCLUSION

Intensional failures of presumptions in queries occur when the user's beliefs about the structure of the data base diverge from those of the system. The use of a partitioned subset hierarchy is essential here to determine those intersections of entity sets that are empty by definition. It is important to distinguish between structure and content, since there is a significant difference in the rate in which they change. When responding to intensional failures of presumptions, simply pointing out the failure is in most cases inadequate. The user must also be informed with regard to related knowledge about the structure of the data base in order to formulate queries directed at solving his/her particular problem. A straightforward, but effective, method for producing such responses was outlined here.

## ACKNOWLEDGEMENT

I would like to thank Aravind Joshi, Bonnie Webber, and Kathy McKeown for their comments and suggestions on the various drafts of this paper. This work was partially supported by a grant from the National Science Foundation, NSF-MCS 79-08401.

## REFERENCES

[Chen 76]
Chen, P.P.S., "The Entity-Relationship Model — Towards a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, 1976.

[Fahlman 79]
Fahlman, Scott E., NETL: A System for Representing and Using Real-World Knowledge, MIT Press, Cambridge, Ma., 1979.

[Kaplan 79]
Kaplan, S.J., Cooperative Responses From a Portable Natural Language Data Base Query System, Ph.D. Dissertation, Computer and Information Science Department, University of Pennsylvania, Philadelphia, Pa., 1979.

[Lee and Gerritsen 78]
Lee, R.M. and Gerritsen, R., "A Hybrid Representation for Database Semantics", Working Paper 78-01-01, Decision Sciences Department, University of Pennsylvania, 1978.

[Smith and Smith 77]
Smith, J.M. and Smith, D.C.P., "Database Abstractions: Aggregation and Generalization", ACM Transactions on Database Systems, Vol. 2, No. 2, June 1977.

# SEMANTICS AND PARTS OF SPEECH

Abe Lockman

Department of Computer Science
Rutgers University
New Brunswick, N.J.08901

Researchers in Natural Language Processing have long recognized the need for an adequate semantic representation language (SRL) in computer systems for the analysis of general natural language texts. One of the primary requirements for any SRL is that it be computable whether or not two formulas in the SRL have the same meaning. (We are, in this paper taking no position on whether this computation should be done through the use of some canonical form or through the use of meaning-preserving transformations.) This requirement forces the creator of an SRL to define its primitive terms by fully specifying the inferences that may be drawn when they appear in a formula; i.e., we define a predicate P to be used in an SRL by specifying which other propositions may be inferred to be true whenever we know that P(a) (possibly in conjunction with some set of other propositions) is true for some a.

While many researchers have proposed various approaches to SRL's, the full definitions of the primitives proposed have rarely been well-defined in terms of the inferences to be drawn. In particular, little attention has been paid to the problem of how an SRL might represent the meanings of words which, while different parts of speech, seem to refer to the same basic concept. Consider, for example, the adverb "speedily," the adjective "speedy," the abstract noun "speed," the agentive noun "speeder," the nominalization "speeding," and the verb "to speed." Obviously the meanings of these words share a common concept, which we might approximate as "some event or set of events taking relatively little time." Unless an SRL can rigorously represent the proper relationships in meaning between the representations which it uses for the meanings of these words, it cannot satisfy the previously mentioned requirement of computable meaning equivalence. That is, an adequate SRL must not represent "x has speed" by P(x) and "y occurs speedily" by Q(y) unless the inferences defining P and Q are properly related. Failure to meet this requirement would result in the inability to note the contradiction in a text such as: "That horse has speed. He does not run speedily." (Of course, a reader could force a consistent interpretation of such a text, in the example presented perhaps by inferring that the horse is sick, drugged, etc., but this forcing is caused by his noting that a normal, i.e. using default knowledge, interpretation causes a contradiction.) The problem of representing such commonalities of meaning has been noted, as in Fillmore 1971, Schubert 1974, and Cercone 1975; in Leech 1974 a sketch is made of what is needed to capture it.

We investigate such commonalities of meaning by exploring and characterizing the ocurrence of part of speech variants for various verb categories of English. The fact that certain groups of verbs do not have certain part of speech variants suggests the usefulness (in semantic decomposition) of new verb categories beyond the traditional ("activities," "situations," etc.) ones. We formulate sets of general expressions for various verb categories, whereby the meanings of part of speech variants may be expressed in terms of one basic meaning which constitutes the component common to a set of variants. The question of what form such a basic meaning should take (i.e. which part of speech variant's meaning, if any, might be considered basic) is discussed for various verb categories.

The evaluation of such expressions during the processing of a text must, in general, incorporate the use of context and world knowledge (CWK). For example, the sentence "x is speedy" means (to a reader) that each (or the typical) member of a certain set Y of events in which x plays an agent/theme role takes relatively little time. Exactly which events Y contains depends, however, on context and world knowledge. If x were known to be a bricklayer (and/or the sentence appears in a bricklaying context) Y would be different than if x were known to be a racehorse (and/or the sentence appears in a horseracing context).

An attempt is made to seperate this task of CWK-dependent evaluation

from that of capturing the variation in meaning (from a common concept) due to part of speech. To this end, we utilize in our expressions special quantifiers which represent calls to be made to a CWK maintenance mechanism when an expression is to be evaluated; these calls ask this mechanism to supply appropriate instantiations for those constituents which are CWK-dependent variables in the expressions. For the above-mentioned "x is speedy," for example, such a special quantifier would be attatched to the "certain set Y of events in which x plays an agent/theme role" as a call to the CWK maintenance mechanism to find and instantiate the most appropriate such set. Our expressions, therefore, are an approximation to the definition of that portion of the meaning of a word that is carried by its part of speech marking, a portion which must be extracted during natural language analysis in order to ensure proper semantic representations.

## References


Cercone, Nick [1975]: Representing Natural Language in Extended Semantic Networks," Technical Report TR75-11, Department of Computing Science, The University of Alberta, Edmonton.

Fillmore, Charles J. [1971]: "Types of Lexical Information," in D.D. Steinberg and L.A. Jakobovits (eds.) Semantics: An Interdisplinary Reader in Philosophy, Linguistics, and Psychology, Cambridge University Press, Cambridge.

Leech, Geoffrey [1974]: Semantics, Penguin Books Ltd., Harmondsworth.

Schubert, L.K. [1974]: Extending the Expressive Power of Semantic Networks, Technical Report TR74-18, Department of Computing Science, The University of Alberta, Edmonton.

# PSI-KLONE

## Parsing and Semantic Interpretation in the BBN Natural Language Understanding System

**Robert J. Bobrow**
*Bolt Beranek and Newman, Inc.*

**Bonnie L. Webber**
*Department of Information and Computer Science,*
*University of Pennsylvania*

### Introduction

This paper describes the syntactic and semantic processing components of a natural language understanding system currently under development at BBN. There are several interesting features of this system which this paper will highlight. The first is a framework for natural language parsing (called the *RUS parser*) which combines the efficiency of a semantic grammar with the flexibility and extensibility of modular syntactic-semantic processing. The second (the *PSI-KLONE interface*) comprises two descriptive taxonomies represented in the KL-ONE formalism [Brachman, 1979] which represent, first the system's knowledge of interpretable syntactic-semantic patterns, and, second, the system's semantic knowledge of possible objects, events and relationships. These taxonomies facilitate the two major tasks of the system's semantic processor:

1. providing feedback to the syntactic processor, and
2. providing semantic interpretations for individual phrases.

A third interesting feature of the system will be touched upon only briefly - its treatment of natural language quantification in terms of a combinatoric problem to be solved, to whatever extent necessary, by a pragmatics/discourse component.

### 1. An overview of the BBN natural language system[1]

The task of the system in which *PSI-KLONE* and *RUS* are embedded is to provide a natural language interface to an intelligent display system in a command and control environment. As well as being able to create and modify displays, the system should be able to answer factual questions about what is on the display screen. Questions and commands addressed to the system typically

1. make use of elements of the preceding dialogue,
2. can be expressed indirectly so that the surface form does not reflect the real intent, and
3. can refer to a shared non-linguistic context (the graphic display)

The issues of anaphora, (indirect) speech acts, and deixis are thus of principal concern.

The natural language system is organized as follows. The user sits at a bit-map terminal equipped with a keyboard and a pointing device. Typed input from the keyboard (possibly interspersed with coordinates from the pointing device) is analyzed by a version of the *RUS parser*. The parser produces a KL-ONE representation of the syntactic structure of an utterance. The production of syntactic constituents incrementally triggers the creation of the local, sentence-level (non-discourse) semantic interpretation of these constituents. This interpretation structure is then processed by a discourse expert that attempts to determine what was really meant. In this process, anaphoric and quantifier-related aspects of the utterance must be resolved and indirect speech acts recognized. Finally, on the basis of what is determined to be the intended force of the utterance, the discourse component decides how the system should respond. It plans its own speech or display actions, and passes them off to a language generation component (not yet implemented) or display expert.

[1] This section and the appendix describing KL-ONE are slightly revised and shortened versions of portions of [Brachman, 1979]. We are grateful to Ron Brachman for giving us permission to use them here.

## 2. The organization of this paper

The next section of this paper discusses the RUS parsing framework - first, the structure of its cascaded interactions with the semantic interpreter, then, techniques used to minimize backtracking in RUS. Section 2 discusses semantic interpretation in PSI-KLONE, with a detailed example of the dialogue that the parser and interpreter carry on in parsing a sentence and constructing the descriptive part of its semantic interpretation. Combinatoric aspects of a sentence's interpretation are discussed in the latter part of this section. For readers unfamiliar with the KL-ONE formalism, an appendix provides a brief introduction.

## 1. The RUS natural language parsing framework

### 1.1. Introduction

RUS is a framework for natural language processing that is as efficient as a semantic grammar, and as flexible and extensible as a modular syntactic/semantic processor. It is based on a non-deterministic ATN parser, but it parses without backup in virtually all cases that Marcus's "deterministic parser" does [Marcus, 1977]. In addition, because of the ATN's ability to operate non-deterministically, RUS can handle phenomena not covered by Marcus' parser.

We have achieved this combination of efficiency and extensibility by *cascading* (see [Woods, 1980]) the syntactic and semantic processors - making calls to the semantic processor at significant points in the parsing process. The near-determinism results in part from two new arc-types - *GROUP arcs* and *almost-GROUP arcs* - and in part from a new control structure for ATNs.

The following two sections describe the features of the syntactic processor. Section 1.2 covers those features that are important for the cascaded interaction of syntax and semantics. Section 1.3 discusses the modifications to the grammar and the normal ATN control structure that increase the determinism of the parsing process.

### 1.2. Syntactic Labelling and Cascaded Interactions

#### 1.2.1. Syntax and Functional Relations among Constituents

We view parsing as a mechanism for providing a functional description of the relations that hold among the pieces which form a syntactic unit (phrase). This description notes the phrase's constituent syntactic units, as well as labelling the functional relations that hold between the constituents and the phrase as a whole. These labels are based on a constituent's functional role in the higher phrase, and not simply on its internal syntactic structure. For example, a noun phrase (NP) can serve various functions in a clause, including logical subject (LSUBJ), logical object (LOBJ), logical indirect object (LINDOBJ), surface subject (SSUBJ), and first NP (FIRSTNP). It is up to the syntactic processor to determine which of these possible labels is appropriate for a given NP constituent of a clause.

These functional labels are primarily intended to provide information for semantic interpretation and discourse processing, and not to screen out ungrammatical constructions. Logical labels such as LSUBJ and LOBJ provide a coupling to the case relationships that are the basis of lexical semantics, while FIRSTNP helps determine discourse focus [Sidner, 1979] and SSUBJ constrains the use of a clause as the source of later verb phrase ellipsis [Webber, 1978].

#### 1.2.2. Cascade interaction between syntax and semantics

The parser does not interact with the semantic interpreter by sending it a complete syntactic analysis of a sentence labelled with the functional relations discussed in section 1.2.1. Rather, the parser and interpreter engage in a dialogue consisting of a sequence of *transmissions* from syntax and *responses* from semantics.

An individual transmission consists of a *transmit triple*, which represents a proposal by syntax of the addition of (1) *a new constituent* with (2) *a label* indicating a particular functional relation to (3) *the phrase* currently under construction by both syntax and semantics. Semantics either rejects the proposal or returns a pointer to a data-structure which represents semantics' knowledge of the resulting phrase. These pointers are all that the RUS syntactic processor knows about the internal operation of the semantic component, and they are simply saved to act as part of the third component of later transmission triples. Thus the RUS framework has no commitment to any particular internal structure for semantic interpretations.

A transmission occurs as part of an arc action in the ATN, with the success of that arc depending on semantics' response to the transmission. The failure of an arc because of a semantic rejection is treated exactly like the failure of

an arc because of a syntactic mismatch; alternative arcs on the source state are attempted, and if none are successful, a back-up occurs.

Transmit actions only occur when enough syntactic structure has been analyzed to confidently propose a functional label for the transmitted constituent. In particular, transmit actions are always postponed until after the head of the current phrase has been recognized. In a simple active sentence[2] like

"The three boys ate two pizzas."

the NP "The three boys" can be labelled as FIRSTNP immediately, and as SSUBJ and LSUBJ immediately after the head verb is recognized. In passive sentences like

"The dog was given a steak bone."

"The dog was given to the first boy who asked for it."

it is impossible to tell if the FIRSTNP "The dog" should be labelled LOBJ or LINDOBJ until the NP after the main verb is parsed.

Note that in this paradigm the parser does not *per se* produce a static syntactic structure. For any given path through the ATN the syntactic structure is implicitly represented in the sequence of transmissions, however, and a parse tree can easily be constructed from these transmissions.

Semantics' responses to a transmission from syntax will be discussed in more detail in section 2.2.2. The important thing to note here is that this response is *not necessarily* the incremental interpretation of the phrase currently under construction. It may simply verify the existence of an interpretation (projection) rule (or rules) by means of which the interpretation of the phrase could be extended by the addition of the proposed new constituent. This buys efficiency by rejecting constructs which have no hope of semantic interpretation and not paying for the construction of a semantic interpretation until a phrase is syntactically checked.

[2] That is, with the exception of sentences such as "John I like.", or any active sentence in which topicalization or Y-movement has occurred.

### 1.3. Approaching Deterministic Parsing

The basic ATN is a non-deterministic parsing mechanism: when more than one arc leaves a state in the ATN, the parser must treat that state as a potential *branch-point*. That is, the parser must select an arc to follow, and if its path from that arc becomes blocked, it must be prepared to back-up to previous branch points and try alternative arcs. A deterministic parser, on the other hand, must be able to treat a state with many arcs as a *choice point*, and make the correct choice of which arc to follow, without allowing for any back-up to that state.

By analysing the back-ups that occurred in a typical non-deterministic ATN parser (i.e., an early version of the RUS system), we found them to have three major causes:

1. the existence of *unnecessary branch-points* in the ATN,
2. the preponderance of "hypothesis-driven" (as opposed to "data-driven") characterizations of English grammar found in the ATN, and
3. the interaction of the normal depth-first control structure of the ATN with the capability for semantic rejection of constituents.

In a typical ATN there are many states that are not true non-deterministic branch-points. That is, for any given sentence there is at most one acceptable arc from such a state. In those cases, the parser should be able to take the correct arc and not have to provide for back-up to that state. In the RUS parser, we have taken advantage of an extension to the normal ATN notation [Burton, 1976] that permits any set of arcs from a single state to be combined into one *GROUP arc*. The arcs within a *GROUP* are then treated as strict alternatives -- at most one can succeed at any point in a parse, and so there is no need to allow for any back-up. The arc sets of many states in RUS could be *GROUP*ed immediately. When this was not the case, it was often possible to *GROUP* arcs by allowing them to examine not only the current word but one or two words ahead.

We have also introduced the notion of an "almost-*GROUP*." This effectively splits a single node in two, with one *GROUP* splitting the situation into deterministic and non-deterministic cases, and another *GROUP* for the deterministic case. This captures our intuition that most sentences could pass deterministically through a given state, and moreover, it would be easy to distinguish the sentences which had to be treated non-determinsitically.

The second cause of back-up mentioned above has been pointed out by Marcus [1977] - the typical use of ATN's as a top-down, hypothesis-driven parsing mechanism. That is, when a point in the parsing is reached where it is possible for a constituent of type X to appear, the parser PUSHes to a network which actually looks for an X. In top-down analysis this is done purely on the basis of the structure found up to that point in the sentence. Back-up can be avoided if such PUSH arcs are not taken when it is clear that the current word (or the next few words) precludes such a constituent. For example, there are places in the analysis of a clause where a PP is optional. We do not want to PUSH for a PP there if the next word clearly precludes its presence - e.g. if the next word is not a preposition.

After analyzing situations where the RUS ATN PUSHed for constituents that were "obviously" not present, we inserted tests that blocked the offending PUSH arcs when the next words were obviously inconsistent with the PUSH arc. These tests required looking no further than the next three words, and often no further than the next word. This is consistent with Marcus' "three chunk" look-ahead. Although there are cases where a three constituent look-ahead would have been required to completely avoid backup, three word look-ahead suffices to drastically reduce the back-up normally caused by top-down parsing.

The third source of back-up lay in the very heart of the RUS approach, namely the incremental semantic testing of constituents, coupled with the ATN's standard *depth-first* control structure. For example, PUSH actions admit the possibility that several constituents of the type pushed for (e.g. several PPs) are present at the given place in the string, differing in length or in internal structure. RUS may reject the first result of the PUSH because it is semantically unacceptable in the context of that PUSH. A "depth-first" control structure will produce all possible alternative constituents of the desired category before trying any alternatives to the PUSH.

However, as the parser becomes more nearly deterministic, the first semantically meaningful result returned from a PUSH is likely to be the best description of what actually occurs at that position. This is particularly true for optional constituents, such as prepositional phrase modifiers (especially those specifying location or time). A frequent case is where an embedded NP PUSHes for a PP, the parser finds one, and the semantic interpreter rejects it as a modifier of the NP. This situation can occur when the first PP found by the parser is actually a modifier of the matrix clause or NP.

For example, consider the sentence

> "That professor teaches undergraduates about languages for processing complex types of list structure."

When the parser is processing the embedded noun phrase "undergraduates" it will PUSH for a PP and find "about languages for processing complex types of list structure" as a semantically coherent PP. This is indeed the correct PP at this point in the string, as opposed to "about languages" or "about languages for processing", and so on, but it is not a PP that can modify "undergraduates." In this situation a depth-first control structure will generate useless parses of meaningful but irrelevant PPs before determining that in this sentence the NP "undergraduates" has no PP modifiers, and that "about languages for processing complex types of list structure" is actually a modifier of the clause.

To avoid this difficulty we have implemented a control structure that produces the first semantically acceptable result of each PUSH[3] but postpones branch-points that might produce alternative results[4] for that PUSH. When this control structure is combined with the *well-formed substring* facility (WFS) which is a normal part of the parser we get an efficient technique for placing optional modifiers where they are semantically acceptable. If an optional constituent is semantically rejected because it was PUSHed for by the wrong level of network, it is stored in the WFS. If some other phrase then PUSHes for the same type of constituent at the same place in the string it will find that constituent in the WFS without any further parsing.

The net effect of these changes has been to remove almost all instances of backtracking in the operation of the parser. Most of the cases where the parser actually has to back up are ones which cannot be resolved on the basis of local evidence, and in which humans often *garden path*.

---

[3] This usually is the longest semantically coherent constituent of the type PUSHed for.

[4] often by dropping off semantically acceptable but syntactically optional post-modifiers

## 2. Semantic interpretation in PSI-KLONE

### 2.1. Introduction

This section describes both the semantic interpretation assigned to an input sentence and the process by which it is assigned. As we indicated in section 1, semantic interpretation is merely an intermediate stage in the processing of a sentence. The final stage is processing by a discourse component which has access to

- the results of the syntactic analysis of the sentence
- the semantic interpretation of the sentence
- general pragmatic knowledge
- evolving models of
  - *the speaker's knowledge, beliefs and current focus
  - *the objects, events and relationships under consideration in the current discourse

The semantic interpreter produces a representation of the input sentence based on the functional syntactic analysis of the sentence (see Section 1.2.1) and a knowledge of lexical semantics to be described here. There are two distinct types of information included in the output of the semantic interpreter - *combinatoric* information and *descriptive* information. This distinction can be viewed as a generalization of the distinction between quantifiers and formulas with free variables (matrices) in quantified predicate logics, and we introduce it by means of an analogous distinction in typed-quantifier predicate logic.

Consider a typed-quantifier predicate logic with the following properties:

1. quantified variables are typed - each variable is limited to range over a particular domain, which is specified by a predicate,
2. variables are allowed to stand for sets as well as for individuals,
3. types are not limited to simple predicates on individuals or sets, but can be complex predicates that may themselves depend on the binding of other variables in the expression, and
4. expressions are written in *Prenex Normal Form*, with all quantifiers pulled out to the left, leaving an open formula to the right.

The advantages of such a logic as a representation for the semantics of English sentences are discussed by Webber [Webber, 1978]. The first three properties allow the information conveyed by noun phrases to be kept separate from the information conveyed in the clause. Properties 1 and 2 reflect the fact that in English one predicates attributes of a set, such as cardinality, in addition to predicating attributes of its members. Finally, property 3 provides for both explicit and implicit dependencies between noun phrases, by allowing the type-predicate for one variable to explicitly depend on the value of another variable.

To illustrate this, consider the sentence

"Each boy gave each girl he knew three peaches"

which we can represent by the typed predicate logic formula

```
(Ax: Boy)
 (Ay: λ(u: Girl)[Know x,u])
   (Ez: λ(w: Setof(Peach))[|w| = 3])
   Gave x,y,z
```

Here the representation of the clause is simply the open formula

```
"Gave x,y,z",
```

while the noun phrases correspond to elements in the quantifier prefix. The variable *x* is shown to range over individual boys, the variable *y* is shown to range, for each boy, over individual girls he knows - an explicit (non Skolem-function) dependency - while the variable *z* ranges over sets of individual peaches whose cardinality is 3. Note that cardinality is a property of sets rather than of individuals. (This particular notation is discussed further in Webber [Webber, 1978], where its value is pointed out for understanding various anaphoric and elliptic phenomena. In PSI-KLONE, we are using the KL-ONE formalism, which provides these properties, as well as an inheritance hierarchy for the efficient indexing of relevant inference rules.)

The reason we have introduced this typed predicate calculus representation is that in Prenex Normal Form, the open formula to the right of the quantifier prefix can be

viewed as a *pattern* - a way of describing a *set* of ground literal formulas by giving their *syntactic shape*. The literals in this set will vary according to how individual constants are substituted for the variables in the pattern. The quantifier prefix, on the other hand, can be viewed as a *combinatoric specification* which determines what ordered combinations of constants can be assigned to the variables to instantiate or stamp out copies of the pattern.

To summarize, we view a semantic representation as having both *a descriptive part* and *a combinatoric part*. In the representation we are using, the descriptive part of a semantic interpretation consists of an interlocking and interdependent collection of Generic descriptions in KL-ONE, to be instantiated to Individual Concepts in ways specified by the combinatoric part.[5] Among the combinatoric constraints on individual instantiations are dependency, distribution and cardinality. All of these will be discussed in section 3.3.

Finally, we believe that a quantified sentence like

> "Which windows were delivered to each house?"

poses an *underconstrained combinatoric problem* which the listener must solve, in order to respond appropriately to the sentence. It is our view that semantic interpretation is only responsible for delineating the problem to be solved, whereas it is the responsibility of the discourse component - using whatever pragmatic and discourse information is available to it - to solve the problem to the extent required to respond appropriately. The procedure to be used by the pragmatic/discourse component to solve this problem is an active area of research.

## 2.2. Semantic Interpretation: Descriptive Information

### 2.2.1. Introduction

This section further describes the dialogue between syntax semantics. There are two things that a cascaded or interactive semantics must do:

1. provide semantic interpretations for individual phrases, and
2. provide feedback to the syntactic processor.

[5] These are not necessarily descriptions of things in the outside world, but rather of objects, events and relationships consistent with the system's long term semantic knowledge.

If one considers two major existing models for computer based parsing - the framework used in the LUNAR system [Woods et al., 1972], and semantic grammar framework [Burton, 1975] - one can see that in both cases there is one mechanism that checks properties of particular constituents and, if those constituents satisfy those properties, then there is another mechanism that shows how to build or add to the interpretation of the whole phrase depending on how those properties are satisfied.

In LUNAR, the pattern-match on the left-hand-side (LHS) of a semantic interpretation rule corresponds to the first mechanism, while the actions specified on the right-hand-side (RHS) of the rule correspond to the second mechanism. In a semantic grammar, on the other hand, PUSHing for a particular syntactic/semantically shaped constituent (e.g. "an NP which is interpretable as a measurement") corresponds to the first mechanism, while some "BUILD action" into a register corresponds to the second.

In the PSI-KLONE interface, each interpretable syntactic/semantically shaped pattern corresponds to a KL-ONE Generic Concept. These Concepts are arranged into a KL-ONE taxonomy which can be used both as a discrimination net and as a mechanism for inheriting appropriate interpretation rules. Semantic checking of potential assignments of constituents to particular functional syntactic roles in a phrase involves information that may be used in building the interpretation of the completed phrase. On the other hand, semantic interpretation only occurs after the entire phrase has been recognized, and the possible rules for semantic interpretation have been collected.

### 2.2.2. Using KL-ONE taxonomies to build semantic interpretations

To illustrate the use of the taxonomy of syntactic/semantic shapes in the PSI-KLONE interface, consider the sentence

> "That professor teaches undergraduates about Lisp on Thursday."

Figure 2-1shows a fragment of a possible syntactic/semantic taxonomy that covers some statements on teaching. We will concentrate on the activity at the clause level and ignore the details of parsing at the NP and

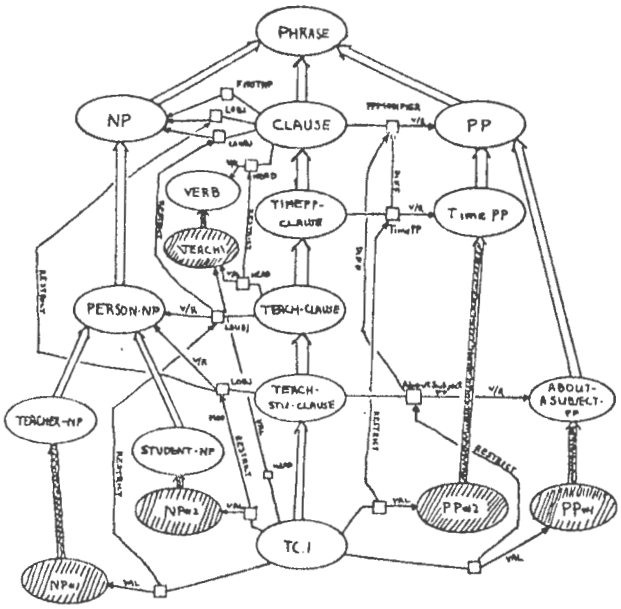



**Figure 2-1:** A KL-ONE syntactic/semantic taxonomy

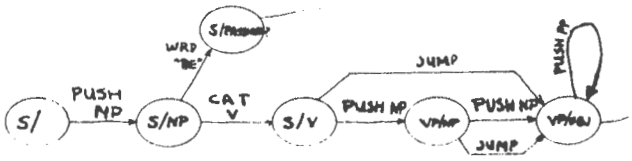



**Figure 2-2:** A simplified ATN for clauses

PP levels. Figure 2-2shows a fragment of a toy ATN that could be used as a (non-deterministic) parser of various types of clauses.

The first step in parsing the example is PUSHing for an NP. This parses the string "That professor" and produces the Individual Concept *NP#1* which is an instance of the Generic *TEACHER-NP*[6], with an associated semantic interpretation not shown in this diagram.

At this point *NP#1* is transmitted as the **FIRSTNP** of the (currently empty) clause, although the parser does not yet have enough information to decide on other roles it fills.

[6]The justification for having a special class of NPs which can be interpreted as teachers is based on the fact that various modifiers like "tenured" are specifically applicable to such NPs, and others, like "at Berkeley", may receive special treatment.

The parser then discovers that "teaches" is the main verb of the clause, and transmits the Individual Concept *\TEACH* (we use the character \ to bracket the names of concepts that stand for morphological units, *\TEACH* corresponds to the morphological root of "teaching") as the **HEAD** of the current clause. The PSI-KLONE interface can now begin to place the clause within the syntactic/semantic taxonomy, as a subConcept of *TEACH-CLAUSE*. This Generic Concept carries the information common to two types of "teach" clauses -- those whose LOBJ is a subject of study like "John teaches calculus" (represented by *TEACH-SUBJECT-CLAUSE*), and those (represented by *TEACH-STU-CLAUSE*) whose LOBJ is human (or at least sentient). The interpretation of a clause of either type is an individuator of the Concept *TEACHING*, and both types of clauses must have an LSUBJ whose interpretation is an instance of *PERSON*. Additionally, both clauses are examples of clauses that can take PP time modifiers. Such clauses correspond to the Generic Concept *TIMEPP-CLAUSE* and *TEACH-CLAUSE* is a subConcept of *TIMEPP-CLAUSE*.

The PSI-KLONE interface responds to RUS with a pointer to a newly created subConcept *TC.1* of *TEACH-CLAUSE*, with its **HEAD** role filled by *\TEACH* and its **FIRSTNP** role filled by *NP#1*. Since the clause is not passive, the parser transmits *NP#1* as the LSUBJ of *TC.1*. From the point of view of semantics, since *NP#1* is an instance of a *PERSON-NP* (by inheritance through *TEACHER-NP*), it can fill the **LSUBJ** role of *TC.1*. Thus semantics fills the **LSUBJ** Role with *NP#1* and returns a pointer to *TC.1* to RUS[7]

RUS then parses "undergraduates" as an NP, producing the Individual Concept *NP#2* which individuates *STUDENT-NP*. The parser cannot transmit this NP yet, because it can function as either the LOBJ or LINDOBJ of a "teach" clause.

We have glossed over an interesting point here -- the fact that it was a restriction on the **PPMODIFIER**s of *STUDENT-NP* that prevented "about Lisp" from being included as a PPMODIFIER of "undergraduates." This is an example of the use of semantic information to reject syntactically plausible parsings.

[7]Actually, a new subConcept of *TC.1* is created with its **LSUBJ** role filled by *NP#1*. This strategy facilitates sharing of information between alternative paths in the parser, but we will ignore it in the remainder of this example.

Once RUS determines that no NP directly follows "undergraduates" it can transmit *NP#2* as the LOBJ of the clause *TC.1*. This is done on the JUMP arc between VP/NP and VP/OBJ. In this case, PSI-KLONE notes that there is a subConcept *TEACH-STU-CLAUSE* of *TEACH-CLAUSE* which allows a *PERSON-NP* as the filler of its **LOBJ** role, and so PSI-KLONE makes *TC.1* a subConcept of *TEACH-STU-CLAUSE* and fills in its **LOBJ** role with *NP#2*.

RUS then parses "about Lisp" as a PP, producing *PP#1*, an instance of an *ABOUT-SUBJECT-PP*. Although a PP in this position may play a special syntactic role in a clause, like a "by ..." PP in a passive clause, *PP#1* does not, so the parser transmits it to PSI-KLONE as simply a **PPMODIFIER** of the clause *TC.1*. Since *TC.1* is now a subConcept of *TEACH-STU-CLAUSE*, it can take such a **PPMODIFIER**.[8] In fact, there is a specialized version of the **PPMODIFIER** role present at *TEACH-STU-CLAUSE*, the role **AboutSubjectPP**, which can accept *PP#1* as a filler. The response to this transmission is a pointer to *TC.1*, which now has *PP#1* filling its **AboutSubjectPP** role.

Finally, RUS parses the PP "on Thursday", producing *PP#2*, an instance of *TimePP*. This is transmitted as a **PPMODIFIER** to *TC.1*, and PSI-KLONE determines that it can fill the **TimePP** role that *TC.1* inherits from *TIMEPP-CLAUSE*. PSI-KLONE returns a pointer to *TC.1* with its **TimePP** role filled by *PP#2*.

At this point the parser is at the end of the clause (and string) and signals this by a transmit triple whose label is **POP**. This signals semantics to check that all necessary Roles are filled and that all inter-Role restrictions are satisfied. Now PSI-KLONE creates the descriptive part of the semantic interpretation of the clause by collecting the *projection rules* that *TC.1* inherits by virtue of its position within the syntactic/semantic taxonomy. These rules are attached as data on various Roles and Concepts in the taxonomy.

PSI-KLONE expresses semantic projection rules in a formal, stylized subset of English called *JARGON* [Woods, 1979b]. There are two reasons for this: one, JARGON is easily read and understood, and two, its interpreter implements an algorithm - the *MSS algorithm* [Woods, 1979a] - that automatically inserts KL-ONE Concepts described in JARGON at the appropriate place in the taxonomy of Concepts. This makes it possible for the descriptive part of a semantic interpretation to inherit all appropriate inference rules that are stored in the long-term semantic taxonomy.

A slightly simplified form of the JARGON phrase that describes the semantic interpretation of the sentence "that professor teaches undergraduates about Lisp on Thursday" is

> A TEACHING WHOSE TEACHER IS THE INTERP OF (LSUBJ) AND WHOSE STUDENT IS THE INTERP OF (LOBJ) AND WHOSE TIMEPREDICATE IS THE INTERP OF (TimePP)

In JARGON, phrases can refer to both concepts and their roles. For example, the construction "THE INTERP OF (LOBJ)" refers to the Role named INTERP of the Concept which is the value of the variable LOBJ.

The JARGON phrase given above is a conjunction of smaller parts, including "A TEACHING", "WHOSE TEACHER IS THE INTERP OF (LSUBJ)", and so on. These parts indicate the source of a particular piece of information on the syntactic side (e.g. "THE INTERP OF (LSUBJ)") and the Role that the piece of information is to fill in the semantic interpretation (e.g. "WHOSE TEACHER", which means the TEACHER Role of the semantic interpretation of the clause).

These pieces of JARGON constitute the semantic projection rules hung on roles in the syntactic/semantic taxonomy. For example, the rule "WHOSE TIMEPREDICATE IS THE INTERP OF (TimePP)" hangs on the Role **TimePP** of the Concept *TIMEPP-CLAUSE*. *TC.1*, the Concept describing the syntactic/semantic shape of the sentence "that professor ...", is a subConcept of *TIMEPP-CLAUSE*, and has an explicit filler (*PP#2*) for the Role **TimePP**, it inherits the projection rule. Similarly, *TC.1* inherits the rule "WHOSE TEACHER IS THE INTERP OF(LSUBJ)" from Role **LSUBJ** of *TEACH-CLAUSE* and the rule "WHOSE STUDENT IS THE INTERP OF (LOBJ)" from the Role **LOBJ** of *TEACH-STU-CLAUSE*, and so on.

[8] Note that *TEACH-SUBJECT-CLAUSE* cannot take such a modifier, so that in a string like "a professor who teaches algebra about Lisp", the PP "about Lisp" would have to be a modifier of something else other than the "teach" clause, as in "John told a professor who teaches algebra about Lisp" where "about Lisp" is unambiguously a modifier of the "told" clause.

When the RUS parser transmits the triple labelled *POP*, the PSI-KLONE interpreter creates a new Individual Concept *TC#1* as an individuator of *TC.1*. This action triggers an attached procedure hung on the highest-level syntactic/semantic concept, *PHRASE*, which collects the projection rules inherited by *TC#1* and forms a JARGON phrase. It then binds the variables occurring there to the fillers of the appropriate Roles (e.g. the variable LSUBJ is bound to *NP#1*, the filler of LSUBJ of *TC.1*), and then calls the JARGON intepreter which builds the KL-ONE concept described by the JARGON phrase, and inserts it at the proper position in the semantic taxonomy. Finally, it fills the INTERP Role of *TC#1* with the Concept in the semantic taxonomy produced by the call to the JARGON interpreter.

### 2.3. Semantic interpretation: Combinatoric information

There seems to be a point in the processing of a sentence where there is some indication of the type of events, objects and relationships being described, but where things have not been resolved into a form which can be represented in an unambiguous predicate calculus type of quantification. People often believe that they have understood the sentence without further elaboration of this part of the interpretation, without realizing that there are remaining quantifier scope ambiguities [Van Lehn, 1978]. Van Lehn suggests that correlations between syntactic structure and quantifier scope interpretation are epiphenomenal - i.e., that there are no processes based purely on syntactic information that can disambiguate quantifier scope. Our belief is somewhat stronger - that there are few, if any, processes that can completely disambiguate quantifier scope simply on the basis of syntactic and semantic information, without making use of discourse-level and pragmatic information.

We believe that this is not accidental - i.e. not a performance error - but rather represents a natural split between the results of the syntactic/semantic component and the activity of later discourse and pragmatically based processes. That is, it is not at the level of the sentence that the information needed to resolve things is available; if it is available at all, it is at the level of the discourse. Moreover, the degree to which scope ambiguities will be resolved is itself dependent on the purposes of the discourse. In some cases in fact, those purposes can be met without raising the spectre of ambiguity at all.

#### 2.3.1. The combinatoric aspects of semantic interpretation

The purpose of this section is to illustrate the combinatoric aspects of a sentence's interpretation that should be identified by semantics, and, if necessary, resolved by pragmatics. Although the current system does not yet treat the combinatoric part of a sentence's interpretation in line with this presentation, we are currently designing a semantic component for PSI-KLONE which does.

There are qualitatively three types of combinatoric constraints embodied in an English sentence:

1. dependencies
2. iterations
3. cardinalities

To illustrate these types of constraints, consider the following sentence

"Two windows were tested in each house."

and the situations in which someone might generate it. In any such situation, the use of "each house" indicates, at a syntactic level, that the speaker has in mind a definite set of houses. (This treats "each house" as equivalent to the phrase "each of the houses".) For this example, label the elements in this set of houses h1,...,hk. There is also something being said about some set (or sets) of two windows. "Two" is *cardinality information* about the number of windows **in each set**. What is not specified is how many sets there are. This can be determined only after implicit dependencies have been made clear. There are three possibilities: there is *no dependency* of one thing on anything else, or there is a *minimal constraint* (Skolem-functional) dependency or an *discourse or definitional* dependency on some other variable.

*No dependency.* In this case, the speaker has in mind two particular windows (call them w1 and w2). There is no dependency, since independent of house, it is w1 and w2 that were tested there. We might represent this in terms of ground literals as

```
Tested-in(w1, h1)
Tested-in(w2, h1)
.
.
.
Tested-in(w1, hk)
Tested-in(w2, hk)
```

Notice that there are two terms for windows and k terms for houses. Pragmatically, there are as many referents for windows and houses as there are terms.

windows tested in some other house[9]. In terms of ground literals,

Tested-in($f_1$(h1), h1)
Tested-in($f_2$(h1), h1)
.
.
.
Tested-in($f_1$(hk), hk)
Tested-in($f_2$(hk), hk)

where $f_1$(hi) =/= $f_2$(hi). Notice that there are 2k different terms for windows here and k different terms for houses. However, all we know about the number of different referents for windows is that there are at least 2.

*Discourse or definitional dependency on the one iterative variable.* Here again the speaker is iterating over houses in the set. For any house hi, the two windows tested there ($w1_{hi}$ and $w2_{hi}$) not only depend on the house, but are members of some previously established, definite set of windows W(hi) that either belong to that house or have been associated with the house through the discourse.[10] In terms of ground literals this can be represented as follows:

Tested-in($w1_{h1}$, h1)
Tested-in($w2_{h1}$, h1)
.
.
.
Tested-in($w1_{hk}$, hk)
Tested-in($w2_{hk}$, hk)

where $w1_{hi}$ =/= $w2_{hi}$, and W is a function from a house to the set of windows belonging to (or associated with) that house. Here again we have 2k different terms for windows and k different terms for houses. Moreover, since W(hi) is a previously established set, one may have additional information by which the referents of $w1_{hi}$ and $w2_{hi}$ can be further constrained. For example, in the case of definitional dependency, pragmatic knowledge tells us that, since a window can only belong to one house, there are 2k different referents for windows.

[9] This may be clearer in the analogous sentence "Two songs were sung by each boy.", in which it is possible that more than one boy sings some particular song.

[10] For example, consider the sequence

"The contractor delivered some experimental windows to each house on the block.

Two windows were tested in each house."

**2.4. Next steps in representing combinatoric semantics**

An important goal of our current research in semantic interpretation is to develop a formalism in which there is a clean split between the descriptive and combinatoric aspects of semantic representation. The number of alternative ground level interpretations of a sentence increase rapidly, as the number of noun phrases (and hence quantifiers) increase in the sentence, and as the number of possible dependencies among entities increase. It is inefficient to try to represent large numbers of such alternatives as an explicit disjunction, both because of the amount of space such a representation would normally require, and because of the complexity of the case analysis that would be necessary to reason forward from such a representation.

We want to provide an efficient representation for that part of the meaning of a sentence in a discourse that can be provided on the basis of its internal syntactic/semantic structure alone. This would include explicit information on the cardinality restrictions on variables associated with NPs, restrictions on which variables are likely to be iterated, and information on possible dependencies among variables, including those suggested by long-term semantic knowledge, and restrictions on dependence based on syntactic structure. We believe that it should be possible to represent such knowledge as a set of constraints on the set of ground level literals to which the sentence might possibly expand.

Such a representation would provide an input to the pragmatics/discourse component, which could refine it and add constraints based on discourse information and perhaps some variants of the heuristics suggested by van Lehn. It is not always necessary for the pragmatics/discourse component to totally disambiguate the combinatoric aspects of semantics in order to satisfy the requirements of the discourse.

We are investigating a number of possible representations in KL-ONE, in the context of a broader study of the use of *meta-description*, the use of KL-ONE structures to describe (classes of) other KL-ONE structures.

## 3. Summary

In this paper, we have tried to give a feeling for the methods and scope of syntactic/semantic processing in the natural language system being developed at BBN. What we feel is most significant about the work to date is the near-determinism of the RUS parsing framework, the effective use of cascaded semantics both to guide the parser and to construct the descriptive part of a semantic interpretation, and the separation made between the descriptive and the combinatoric part of an interpretation. We expect to have more to say in a later paper about the details of the representation of combinatoric information and its interaction with descriptive structures.


### *Acknowledgements*

Our work on the syntactic/semantic side of the system has not been done *in vacuo* - our colleagues in this research effort include Ed Barton, Ron Brachman, Phil Cohen, David Israel, Hector Levesque, Candy Sidner, and Bill Woods. We hope this paper reflects well on our joint effort.

The authors wish to thank Lyn Bates, Ron Brachman, David Israel, Aravind Joshi, Candy Sidner, Brian Smith and Bill Woods for their helpful comments on earlier versions of this paper. Our special thanks go to Susan Chase, who bolstered our somewhat flagging spirits with a magnificent celebration feast, somewhere between near-deterministic parsing and combinatoric aspects of semantic interpretations.

This research was supported by the Advanced Research Projects Agency of the Department of Defense, and was monitored by ONR under Contract No. N00014-77-C-0378.


## 1. A brief introduction to KL-ONE

KL-ONE is a uniform language for the explicit representation of conceptual information based on the idea of *structured inheritance networks* [Brachman, 1978, 1979]. Several of its prominent features are of particular importance in PSI-KLONE - its semantically clean inheritance of structured descriptions, taxonomic classification of generic knowledge, intensional structures for functional roles (including the possibility of multiple fillers), and procedural attachment (with automatic invocation).

The principal representational elements of KL-ONE are *Concepts*, of which there are two major types - Generic and Individual. Generic Concepts are arranged in an inheritance structure, expressing long-term generic knowledge as a taxonomy. A single Generic Concept is a description template, from which individual descriptions (in the form of Individual Concepts) are formed. A Generic Concept can specialize one or more other Generic Concepts (its *superConcepts*), to which it is attached by *inheritance Cables*. These Cables form the backbone of the network and carry structured descriptions from a Concept to its subConcepts.

KL-ONE Concepts are highly structured objects. A subConcept inherits a *structured* definition from its parent and can modify it in a number of structurally consistent ways. The main elements of the structure are *Roles*, which express relationships between a Concept and other closely associated Concepts (i.e., its properties, parts, etc.). Roles themselves have structure, including descriptions of potential fillers,[11] modality information, and names.[12] There are basically two kinds of Roles in KL-ONE: *RoleSets* and *IRoles*. RoleSets have potentially many fillers and may carry a restriction on the number of possible fillers (e.g., the **officer** Role[13] of a particular COMPANY would be filled once for each person who is an officer of that company). A RoleSet on a Generic Concept represents what is known in general about the fillers of that Role. A RoleSet on an Individual Concept stands for the particular set of fillers of that Role for that individual (e.g., the officers of a particular company). IRoles (for 'Instance Roles') appear only on Individual Concepts, and are used to represent particular bindings of Roles to Individual Concepts (e.g., the **president** of a particular COMPANY). (There would be one IRole for each officer position in a particular company, regardless of the actual number of people playing those Roles.)

There are several inter-Role relationships in KL-ONE, which relate the Roles of a Concept to those of a superConcept. Such relationships are carried in the inheritance Cables mentioned earlier. They include:

- restriction (of filler description and/or number); e.g., that a particular kind of COMPANY will have exactly three **officers**, all of whom must be over 45
- differentiation (of a Role into subRoles); e.g., differentiating the **officers** of a COMPANY into **president**, **vice-president**, etc. This is a relationship between RoleSets in which the more specific Roles inherit all properties of the parent Role except for the number restriction (since that applies to the set and not the fillers);
- particularization (of a RoleSet for an Individual Concept); e.g., the **officers** of BBN are all COLLEGE-GRADUATEs; this is the relationship between a RoleSet of an Individual Concept and a RoleSet of a parent Generic Concept.
- satisfaction (binding of a particular filler description into a particular Role in an Individual Concept); e.g., the **president** of BBN is STEVE-LEVY; this is the relationship between an IRole and its parent RoleSet.

---

[11] These limitations on the form of particular fillers are called "Value Restrictions" (V/R's). If more than one V/R is applicable at a given Role, the restrictions are taken conjunctively.

[12] Names are not used by the system in any way. They are merely conveniences for the user.

[13] In the text that follows, Roles will be indicated as boldfaced names and Concepts will be indicated by all upper case expressions.

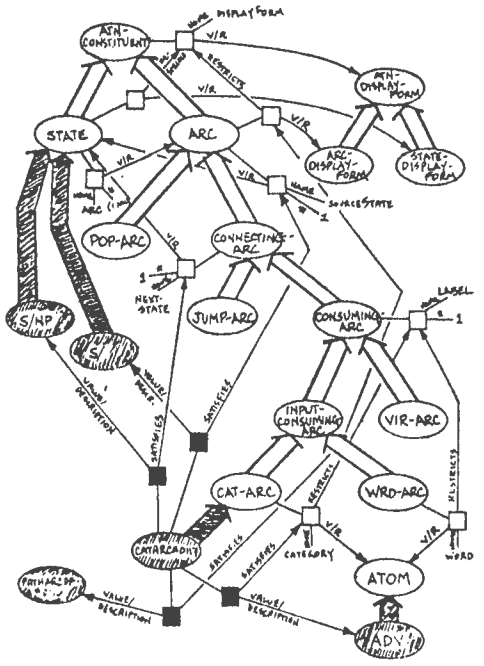



Figure 3-1: A piece of a KL-ONE taxonomy

Figure 3-1 illustrates the use of Cables and the structure of Concepts in a piece of the KL-ONE taxonomy describing an ATN grammar. Concepts are presented as ellipses (Individual Concepts are shaded), Roles as small squares (IRoles are filled in), and Cables as double-lined arrows. The most general Concept, ATN-CONSTITUENT, has two subConcepts - STATE and ARC. These each inherit the general properties of ATN constituents, namely, each is known to have a **displayForm** associated with it. The subnetwork below ARC expresses the classification of the various types of arcs in the ATN and how their conceptual structures vary. For example, a CONNECTING-ARC has a **nextState** (the state in which the transition leaves the parsing process), while for POP-ARCs the term is not meaningful (i.e., there is no **nextState** Role). Links that connect the Roles of more specific Concepts with corresponding Roles in their parent Concepts are considered to travel through the appropriate Cables. Finally, the structure of an Individual Concept is illustrated by CATARC#0117. Each IRole expresses the filling of a Role inherited from the hierarchy above -- because CATARC#0117 is a CAT-ARC, it has a **category**; because it is also a CONNECTING-ARC, it has a **nextState**, etc.

There is one important feature of KL-ONE that is worth pointing out, although it is not yet used in the current natural language system. The language carefully distinguishes between purely descriptional structure and assertions about coreference, existence, etc. All of the structure mentioned above (Concepts, Roles, and Cables) is *definitional*. All assertions are made relative to a Context (another type of KL-ONE object) and thus do not affect the (descriptive) taxonomy of generic knowledge. We anticipate that Contexts will be of use in reasoning about hypotheticals, beliefs, and wants.

The final feature of KL-ONE relevant to our discussion is the ability to attach procedures and data to structures in the network. As mentioned previously, KL-ONE is used in several places in our language understanding system - these include the syntactic taxonomy used to constrain parsing and to index semantic interpretation rules, and the structures used in the syntactic/discourse interface to express the literal semantic content of an utterance. The parser uses KL-ONE to describe those syntactically correct structures for which there are known interpretation rules. Interpretation *per se* is achieved using attached procedures and data, with semantic projection rules expressed as data attached to Roles of the syntactic Concepts.

## BIBLIOGRAPHY

Brachman, R. A Structural Paradigm for Representing Knowledge. Technical Report 3605. Bolt Beranek & Newman Inc. Cambridge MA. 1978.

Brachman, R. On the Epistemological Status of Semantic Networks. In *Associative Networks*, N.V.Findler (ed.). New York: Academic Press, 1979.

Burton, R. Semantic Grammar: An engineering technique for constructing natural language understanding systems. Technical Report 3453. Bolt Beranek & Newman Inc., Cambridge MA, 1976.

Jackendoff, R. *X-bar Syntax: A study of phrase structure*. Cambridge MA: MIT Press, 1977.

Marcus, M. A Theory of Syntactic Recognition for Natural Language. Ph.D. thesis, MIT, 1977. (Also published by MIT Press, 1979)

Sidner, C. Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse. Technical Report 537. MIT AI Lab. Cambridge MA, 1979.

Van Lehn, K. Determining the Scope of English Quantifiers. Technical Report 483. MIT AI Lab. Cambridge MA, 1978.

Webber, B. A Formal Approach to Discourse Anaphora. Technical Report 3761. Bolt Beranek & Newman Inc., Cambridge MA, 1978.

Woods, W. Theoretical Studies in Natural lagnauge Understanding: QPR 6. Technical Report 4161. Bolt Beranek & Newman Inc., Cambridge MA, 1979a.

Woods, W. Theoretical Studies in Natural Language Understanding: Annual report. Technical Report 4332. Bolt Beranek & Newman Inc., Cambridge MA, 1979b.

Woods, W. Cascaded ATN Grammars. *Amer J. Computational Linguistics*, 6(1), Jan-Mar. 1980, pp.1-12.

Woods, W., Kaplan, R. & Nash-Webber, B. The Lunar Sciences Natural Language Information System: Final report. Technical Report 2378. Bolt Beranek & Newman Inc., Cambridge MA, 1972.

# THE ROLE OF DISCOURSE STRUCTURE IN LANGUAGE PRODUCTION

David D. McDonald[1]
MIT Artificial Intelligence Laboratory
Cambridge, Massachusetts 02139

**Abstract**

A technique is described whereby descriptive discourse structures can be used directly to plan texts, manage details, and insure grammaticality. Constituent structures are interpreted as plans of action, executed by a controller that walks the tree, interpreting categories and features as latent actions or as constraints on further decisions. Under this interpretation, linguistic descriptions become programs for achieving rhetorical effects in the output text. One can then use production as an experimental laboratory to test the effectiveness of a given linguistic analysis.

## The Need for a Planning Language

Purposeful language production is a planning process no different, in many respects, from guiding robots or stacking blocks into towers. Goals must be defined, means and ends analyzed, and, most important of all, there must be a *planning language*: a means of representing texts[2] in progress, linguistic and rhetorical actions that have been scheduled but not yet carried out, and pending goals at all levels of refinement.

The central problem of language production is to translate a speaker's goals from their original internal form to a linguistic one. In non-trivial cases, the process requires deliberating between alternatives and advance planning; otherwise a decision made for one goal will often make other goals impossible. A speaker cannot perform this reasoning in terms of alternate finished texts: the space of possible texts is unmanageably large; and, more to the point, the detail of a finished text is rarely relevant to the decision-making (unless perhaps the speaker is a poet) and only serves to obscure those facts that are relevant. A planning language gives the speaker the ability to reason at the right level of detail, to postpone decisions that are of lesser importance or are determined by others, and to have a concise record of what actions have been taken, what are planned, and what remain possible (i.e. grammatical) in that context.

**The proper planning language for natural language production is linguistic structure[3] itself: syntactic and morphological structures for planning sentence-level details, and discourse structures for planning large-scale texts and rhetorical relations at all levels.**

This assertion is made, of course, in the context of a very specific proposal for how production is to be done and how the linguistic structure is to be interpreted. This proposal is the subject of my dissertation [McDonald 1980]. I have developed a theory of the linguistic component of the process, based on the psycholinguistic hypothesis that our (human) fluency as speakers stems from the use of an incremental, indelible[4] process that produces utterances in time linear with the number of elements in the source message. The dissertation

1. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research contract N00014-75-C-0643.

2. For practical reasons, my research has involved only the productions of written texts rather than acoustic signals. Also only one natural language, English, has been used.

3. By "linguistic structure" I mean specifically a surface-level, immediate constituent structure tree of the usual sort. The trees in my dissertation were annotated by category, had functional labels for constituent positions, included features (as in systemic grammar [Halliday 1970]), and used traces [Chomsky 1973]. None of these are absolute requirements however.

4. I.e. "written with indelible ink"—once a decision has been made it cannot be taken back or revised, only refined.

presents the theory and the experiments with the LISP program that embodies it, with emphasis on syntactic-level design decisions.

In the present paper, I want to focus on the application of the proposal to discourse structure. I do not have a theory of discourse structure to present, rather I will show how a constituent-oriented theory would be applied to control discourse production.

### Linguistic structures as planned actions

The process begins with an abstract description of what is to be said (a "message"). This description may be of any size or level of detail, provided that it can be relationally decomposed into a well-ordered hierarchy of elements. This hierarchy is very important: As we will see, the description is going to be its own production program—its well-ordered hierarchy will be what dictates its order of execution.

The knowledge base of the process is a "production dictionary", written specially for each new speaker and problem domain.[5] The dictionary has an "entry" for every relation (or relation type) that could appear in a description. This entry is a schematic description of the possible realizations that the relation could have and the conditions: pragmatic, intentional, and grammatical, that will dictate which realization to choose in a specific situation. The entry is where the internal vocabulary and representational calculus of the speaker is translated into a linguistic vocabulary and representation. Realizations are specifications of linguistic structures: phrases, words, rhetorical effects; typically they are translations of the relation proper, that incorporate the arguments to the relation intact for realization at a later point.

To see how a description can use the dictionary to become "its own production program", let us look at a simple example. Assume that what we want to say is described by the classic expression:

$$\forall(x)\ \ man(x) \Rightarrow mortal(x)$$

This has a clear, well-ordered decomposition: first the quantification, then the implication, then the two predications. That then will be the order of the realization decisions: first the quantification, then the implication, and so on.

We want the production process to be incremental: no one entry should be responsible for more than it has the actual expertise to decide. This means that the quantification should make its decision in terms of picking a context for the implication, and the implication in terms of a context for the two predicates.

The contexts are given in the planning language, i.e. the linguistic structures selected by earlier realizations. They are expressed as constituent structure trees with the yet-to-be-realized relations embedded at their leaves. (Message relations ("elements") can remain in their original representation since their dictionary entries will act as interpreters.) When an element is realized, it is replaced in the tree by the syntactic phrase (or word, or subelement) that was chosen for it.

There are very specific dependencies between decisions. By making the decisions in a specific order, i.e. top-down through the message and top-down and left-to-right through the incrementally growing constituent tree, we can guarantee the production *in one pass* of a grammatical text that satisfies the speaker's description (The organization of that source description must also meet certain criteria, see [McDonald 1980].) We enforce the ordering by vesting control of all processing in a simple dispatching routine ("the controller") that walks the constituent structure along the desired path. When it reaches a message relation, it dispatches to the appropriate entry; when it reaches a word, it has it printed out; and when it reaches a linguistic descriptor (a category name, a feature, or a labeled position), it runs an attached procedure, refining the context or directly printing a function word.

---

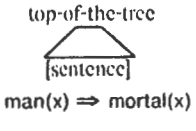
5. The computer program is designed to be a separate "linguistic component" that is then specialized to new domains by writing a dictionary and a set of simple interface functions. So far, the program has been used with five different "speakers" using as many different representations including the predicate calculus, pattern-matching assertions, and KL ONE [Brachman 1979].

Most realization decision will require multiple actions to implement, yet can only have one action performed at a time (i.e. the morphemes of the text must be printed in order). The "pending" actions thus must be stored until their time comes. Since what an entry decides is to use are instances of linguistic constructions, fixed phrases, or words, the form of the storage should be a linguistic description.

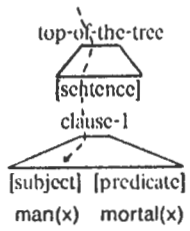
**By interpreting the usual descriptive linguistic structures as the specifications of constraints and as a source of "default" grammatical actions triggered by the passage of the controller, we can use the structures as a representation for pending decisions.**

Let us follow this process for the example formula. We start out with no context (i.e. no prior decisions and therefore no constraints). As the first action, the controller dispatches to the entry for the quantification. That entry makes some tests on the logical structure and linguistic potential of its argument (the implication) and then chooses a realization. Let us say that it decides that the implication can be given as a statement and that the quantification *per se* should be given as the determiner of the variable (i.e. "*all men*").

The quantification entry cannot implement these actions directly because they involve realization decisions that haven't happened yet. Instead, it must make a record of its decision—create a linguistic context that will force the implementation of its decisions at the appropriate time. To do this, it (1) installs the implication in a "sentence" context, and (2) it creates a "premature instance" of the variable with its determiner decision preempted to be "express-quantification". (The preempted decision will take effect much later the first time the variable is realized. We will pick up its trail then.)




Since its context is marked as requiring a sentence, the entry for the implication is now constrained to pick a clause (i.e. it can't choose "*mortal men*"). Let us say that it decides to translate the consequent as a statement about the antecedent. (The actual entries used are included in [McDonald 1980].) It implements this decision by placing the two predications in a context that identifies them as the subject and predicate of a clause, i.e.




The point of a label like "clause" or "subject" is not description for description's sake. These labels are a succinct representation of pending grammatical actions and constraints: the presence of a clause node in a sentence slot will initiate agreement once the controller reaches the main verb; the subject label will indicate where to find the element to agree with. The clause "node" defines a region with specific grammatical properties and provides a context-binding mechanism. Having "man(x)" in a subject slot will constrain its entry to choose a noun phrase realization. (The slot label actively filters out the ungrammatical alternatives as the entry is interpreted.)

The construction of the noun phrase itself is broken down within the entry into a set of near independent decisions organized by linguistic category: the referent of the phrase is the variable "x"; its "head" is taken from the name of the predicate; its "determiner" is fixed, in this case, by the earlier decision of the quantifier but would otherwise have been decided on the basis of tests for generic/specific. This internal organization of the entries is a linguistic structure just like the constituent structure and may have attached procedures and transformations on the same basis.

By building an explicit, abstract structure to shadow the produced text, we not only gain an efficient representation of pending decisions, but we can design transformations, monitors, and decision heuristics that apply on the basis of the structures themselves and do not need to know anything about the contents of the particular message. If we are then careful about using well-motivated linguistic structures, we will recoup effective general procedures from our investment.

### Unplanned coherence

The primary things that make a discourse more than just a sequence of sentences are matters of content: a common topic, common references, a progression of temporal and causal chains, or a rhetorical "point" or moral. Also necessary however are the so-called "coherence phenomena" at a linguistic level: using pronouns and definite noun phrases, not repeating facts that have been recently mentioned, or using ellipsis to condense texts with a common linguistic structure.

Many coherence phenomena can be introduced into a text "automatically"—without the speaker actively planning to use them—by associating them with discourse-level structures. Below are the first few paragraphs from a long (if boring) text written by my program by reading out the relations in a semantic net. The only aspects of this text that were specifically planned were the order of the paragraphs, and the order of the facts within them (e.g. whether there was to be a summary, or whether one fact was to be attached to another by a relative clause). Every other effect was controlled by a general purpose rule.

> ***Phrase*** *is the top of the net. Its* ***interp*** *role must be a* ***concept*** *, and its* ***modifier*** *role and its* ***head*** *role must be* ***phrases*** *. Its subconcepts are* ***pp*** *,* ***np*** *,* ***adjunct*** *,* ***indobjclause*** *, and* ***word*** *.*
>
> ***Pp*** *has the roles:* ***pobj*** *,* ***prep*** *,* ***interp*** *, and* ***ppobj*** *.* ***Pobj*** *must be a* ***np*** *,* ***prep*** *a* ***prep*** *,* ***interp*** *a* ***relation*** *, and* ***ppobj*** *a* ***pp*** *.* ***Pp*** *'s subconcepts are* ***ofpersonpp*** *, and* ***insubjectpp*** *,*
>
> ***Ofperson*** *has a* ***pobj*** *role which must be a* ***humanp*** *, and a* ***prep*** *role which must be an* ***of*** *.*
>
> ***Insubjectpp*** *'s pobj role must be a* ***subjectnp*** *,*
> *its* ***preprole*** *an* ***in*** *, and its* ***interp*** *role a* ***subject*** *.*
> ***Np*** *is another subconcept of* ***phrase*** *...*

The instances of subject merging and gapping were controlled by local monitors associated with the conjunctions. The pronominalization was controlled by a record of mentioned items that was sensitive (1) to the paragraph structure (items in earlier paragraphs[6] were considered too "old" to pronominalize) and (2) to each paragraph's "focus",[7] defined here as the network node whose properties were being described. Focus also triggered several order transformations whose aim was to position the focused item in an early, "given" location in the construction, e.g. saying "*its* ***interp*** *role*" rather than "*the* ***interp*** *role of* ***phrase*** ".

A further unplanned effect was the omission of "given" information, i.e. if a fact had appeared once it was not be repeated. Thus in the second paragraph, it said "***Pobj*** *must be a* ***np*** ", rather than "*The* ***pobj*** *role must...* " because the fact that "pobj" was a role had just been given in the previous sentence. Similarly, the originally planned first sentence of that paragraph: "***Pp*** is a subconcept of ***phrase*** " was left out because it had just been given in the last sentence of the previous paragraph. Sensitivity to what information is already "given" is part of the controller's entry interpreter.

The problem with unplanned coherence effects are that they are sometimes insensitive to the range of alternatives that are possible—their triggering conditions are usually myopic. Consider an example from yet another domain, the annotation of games of tic-tac-toe.[8] Suppose we have two moves as shown below and wish to

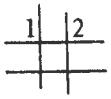
---

6. An exception is the "topic sentence" of the paragraph. These are presumed to be more salient than the bodies, and items mentioned in them are kept in an alternate chronological record. This is how the subsequent reference to "*another subconcept of phrase*" was motivated.

7. Focus in the sense of being what the paragraph is "about". See [Sidner 1979] and [Grosz 1977].

8. Unlike the earlier logic and "network-as-object" domains, the dictionary for the tic-tac-toe domain is still being developed. Not much more than the examples given in this paper have actually been implemented thus far. The model for this domain is the very fine work on discourse production done by Anthony Davey [Davey 1974].

describe them as two instances of the predicate "take a corner".




If we join the two clauses with a conjunction, then we have at least the following alternative phrasings:

(1) *Both you and I took a corner.*

(2) *You took a corner and so did I.*

(3) *You took a corner and I did too.*

(4) *You took a corner and I took another one.*

(5) *You took one corner and I another.*

The problem comes when you consider that the order of those sentences is the same as the order in which the controller will pass through each of their triggering monitors. If those monitors are set to trigger every time they are applicable, then we will always get sentence one, since its monitor will be reached first, and it will always apply, preempting the others.

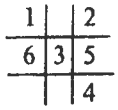
There are, of course, "kludgy" ways to get around this problem: e.g. defining a system of five enabling features, and allowing only one feature from the system to be "alive" at a time, varying the choice on some pseudo-random basis.

A better way (but one we do not understand well enough to implement) would be to determine what rhetorical effects are best served by each construction and to then label the context according to the effect the speaker want to achieve. That would give us very specific triggers where formerly we had only the most general one—"make the text more coherent".

A large part of motivating a speaker to make such specific and varied distinctions in intention is having them employ a rich discourse structure. In the remainder of this paper, I want to look at some examples of a complex discourse/planning structure that motivate distinctions in text length and content. While not nearly as subtle as those required above, these distinctions will be of the same type.

**Planning a Discourse**

Let us imagine that we are to describe the game below:




The first thing that we must determine is the level of description to use. Obviously, this will affect the lexical choices open to us; more interestingly however, it will also greatly affect our options for discourse structures. I will look at three different levels in turn: each will permit successively richer structures, and finer control over the details.

A description just in terms of squares-taken can hardly motivate any discourse structure at all, because each move is conceptually the same as the next. Since both six separate sentences or a six-long conjunct are stylistically unacceptable, a content-free default of conjoining pairs is used. The conjuncts will then motivate some default coherence strategies.[9]

---

source description: (and 1 3) (and 5 9) (and 6 4)

Resulting text:

> *You took a corner and I took another. You took the center and I took the corner below mine and opposite yours. You took the middle square between my corners and I took the middle square opposite it.*

---

9. In the figure below and those that follow, moves are referred to by the number of the square taken, where squares are numbered starting with "1" top to bottom and left to right.

If we now move up to the tactical level of "threats" and "blocks", we find a much richer semantic structure tying the moves together, and it becomes possible to plot out a rhetorical structure that will emphasize the semantic one. We begin by laying out the tactical description of the game:

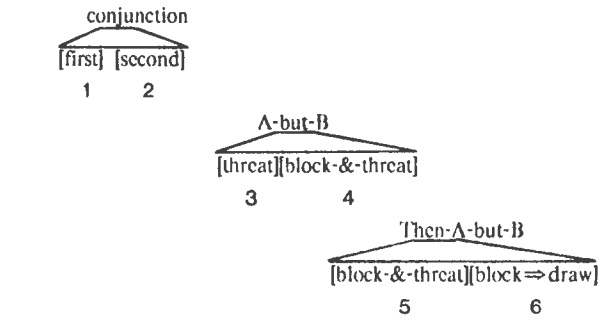
1. nothing
2. nothing
3. threat
4. block & threat
5. block & threat
6. block ⇒ draw

The next step it to "parse" this description and work out a pattern of rhetorical connections, e.g.

[ 1 and 2 ]
[ 3 but 4 ]
[ Then 5 but 6 ]

At this point we should consider what the entry for a "move" should be. Semantically, a move is a locus of relations at many different descriptive levels: no one level is primary, and combinations of levels can be effective (i.e. "*I threatened you by taking the center*").

We can control a "multi-facet" entry like this through a combination of explicit directives and contextually controlled defaults. Most of the directives will come from the discourse structure that we build by combining the tactical description and the rhetorical parsing into a constituent structure[10] with the moves as the leaves.




You may recall that the earlier text made extensive use of "corners" in its descriptions of squares taken. If a fact like that can be seen at "planning time", it can be turned into a discourse directive to use constructions that emphasizes the corner description though deletion (see the first sentence of the output text below). This is one example of a rhetorical effect that can serve to pick out a specific construction.

The default for the move entry--what it does when the move is in a constituent position that is not marked for a particular relation—will be to give the square-taken interpretation. Further let us say that when the relation is specified, the entry will add the relation of the next, more concrete level of description as a default. The result is given below: (The choice of constructions for "block and threaten" is controlled largely as synonymous alternatives.)

*You took one corner and I another. You threatened me by taking the center, but I blocked you and threatened you in turn by taking the corner below my corner and opposite yours. Then you blocked me with a threat from the middle square between my corners, but I blocked it by taking the opposite middle square, resulting in a drawn game because there were no more lines to take.*

10. Note, part of the meaning these "rhetorical categories" is a procedure triggered by the controller that causes the conjunctions "but", "then", and so on to be printed directly without ever occupying a constituent position in the tree.

At an even higher level of abstraction, consider the frame below, constructed from a study of the tactical description and reasoning about alternatives.[11]

---

missed-opportunity-37

player-who-did-it "you"
summary-missed-action "fork"
summary-actual-outcome "draw"
summary-possible-outcome "win"

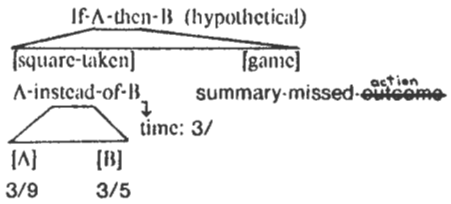
bad-move 3/5
possible-move 3/9
lead-in-moves (1/1, 2/3,)
actual-following-moves (4/9, 5/6, 6/4)
possible-following-moves (4/5, 5/7, 6/{4,8})

| | | |
|---|---|---|
| 1 | | 2 |
| 6 | 3 | 5 |
| | | 4 |

---

By developing an entry for a "missed-opportunity", we can start the production process directly with this frame. The entry will be fundamentally just like any other entry: recording alternative realizations and the conditions that select between them. The difference will be that its output is a specification for a discourse rather than for a syntactic phrase, with an output vocabulary in terms of the facets of the frame and different rhetorical specifications.

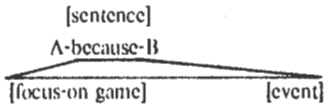
The clustering and labeling of moves in terms of the semantically charged facets makes it possible to summarize them, or to treat them as units in the discourse that can be commonly marked, say, as hypotheticals, as in this possible realization:




which, if selected, would eventually lead to the text:

*If, on your second move, you had taken the opposite corner instead of the center, you could have forked me and won the game.*

Another alternative would be:




*"The game was a draw, because you missed an opportunity for a fork."*

Because this alternative is constrained to be a single sentence, the entries for the two facets are inhibited from adding any "default" elaborating clauses because that would strain the heuristics defining good sentence size.

Realizations at this discourse level are as amenable to intentionally directed transformations as at any other level, and can be organized into families of alternative orderings, alternate focuses, or alternate assumption about what information is deducible and should be omitted; though it is certainly not clear at this juncture what the correct organizing rules and heuristics for this kind of reasoning will be.

**However, by studying discourse heuristics from the point of view of language production, we have a straight-forward way to test our ideas, namely to run the process, produce the texts, read them, and determine if the choice of rhetorical devices met our goals.**

The direct control over the process parameters that a study of production affords us will allow it to make powerful contributions to our understanding of all aspects of natural language.

---

11. Moves are given here as <number of move>/<number of square>. The example game is repeated for convenience.

## References


Brachman, R.J. (1979) "On the Epistemological Status of Semantic Networks", in Findler ed. *Associative Networks: Representation and Use of Knowledge by Computers*, Academic Press, New York.

Chomsky, N. (1973) "Conditions on Transformations" in Anderson and Kiparsky, eds., *A Feschrift for Morris Halle*, Holt, Rinehart, and Winston, New York.

Davey, A. (1974) The Formalization of Discourse Production, Ph.D. Dissertation, Edinburgh University.

Grosz, B. (1977) The Representation and Use of Focus in Dialogue Understanding, Ph.D. Dissertation, University of California at Berkley.

Halliday, M.A.K. (1970) Functional Diversity in Language as seen from a consideration of Modality and Mood in English, *Foundations of Language*, 6, 322-361.

McDonald, D.D. (1980) Natural Language Production as a Process of Decision-making Under Constraints, Ph.D. Dissertation, MIT.

Sidner, C. (1979) Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse, Ph.D. Dissertation, MIT.

# NATURAL LANGUAGE QUERIES FOR A LINGUISTIC DATA BASE USING PROLOG

Richard Kittredge

Department of Linguistics
University of Montreal
Montreal, Que. H3C 3J7

## ABSTRACT

METAVERB is a small user-designed system for consulting a data set for the linguistic study of English verb syntax. A question-answer control component, written in PROLOG, a flexible language for logic programming, links the reply to a query to the success in deriving the semantic structure of the query as a theorem from the data set, using the techniques of resolution theorem proving. English queries are analyzed into their semantic representations simultaneously with syntactic parsing in 'metamorphosis grammar', a super-Q option within PROLOG. The problem of analyzing metalinguistic queries, which may both use and mention the same verb, is also considered.

## 1. INTRODUCTION

The programming language PROLOG (Colmerauer et al.,1972,1979) has been designed to facilitate programming in first order logic using the techniques of resolution theorem proving (see also van Emden, 1977). Its applications until now have included formal integration, medical diagnosis, speech recognition, and certain data base query systems. Colmerauer's addition of 'metamorphosis grammars' to PROLOG (Colmerauer,1975) has simplified the treatment of strings and tree structures, and hence made PROLOG more accessible to linguists interested in natural language analysis.

This paper describes METAVERB, a functioning natural language query system which answers questions about the syntactic properties of English verbs, such as "Which verbs is NPN an object type of?" or "Is each object type of 'accuse' an object type of a verb whose subject type is THS?". Although the query language is quite limited and particular to the domain of syntactic research in linguistics, this system illustrates a number of interesting possibilities in the use of PROLOG, for specifying the semantics of a subset of natural language, and metamorphosis grammars, for controlling the compositional build-up of semantic representations during the process of syntactic parsing.

---

*Virtually every aspect of this system has been influenced by the work of the Groupe d'Intelligence artificielle of the University of Marseille-Luminy. In addition to references to published work, I should like to acknowledge the collaboration of Alain Colmerauer during the design and testing of an early version of this system, particularly with regard to the implementation of his proposals for the treatment of linguistic quantifier expressions.

After a brief description of the data set in §2, we consider the semantic representation given to the basic query types in §3. We then review certain relevant aspects of PROLOG and illustrate the control and semantic sections in §4. In §5 we examine the syntactic parsing of queries, which is coupled with the construction of the desired semantic representations using metamorphosis grammars. Finally, in §6, we consider the problem specific to METAVERB of parsing sentences from the metalanguage of English syntactic description.

## 2. THE DATA BASE

The query system is designed to answer questions about the syntactic properties of verbs, based on a data set for roughly 800 English verbs, including most verbs which take a wide variety of prepositional and sentential complements. A syntactic study by Kiss & Kittredge (1976) provided in tabular form the information on each verb's potential to take any of six possible subject forms and any of the 40 possible object strings. For the purpose of the METAVERB experiments, each verb's potential was given in a tree structure having the form of a positive PROLOG literal in which the verb name, list of possible subject forms and list of possible object forms are the three branches. For example, the verb "accuse" has a data representation:

+VERB(ACCUSE, N.NIL, N.NPN.NPVGO.NIL).

where "." serves as a list concatenator. The three non-NIL elements of the object list correspond to the three possible oject strings for this verb:

Noun (phrase) e.g. ...accuse Fred.
Noun+Preposition+Noun
e.g. ...accuse Fred of treason.
N+P+Verb+inG+Object
e.g. ...accuse the boy of creating a disturbance.

At the moment, relatively little is known about the dependencies between syntactic properties, or the possible groupings of verbs according to partial similarities of their properties. To advance this kind of research a query system is required with maximum flexibility to compare property lists, permit definitions of new composite properties, etc.

It might be a relatively simple matter to tailor-make an artificial query language which would permit efficient access for a large number of kinds of query. The decision to use natural language is due to several factors including the following:

(a) Considerable insight into linguistic semantics is gained from the exercise of specifying precisely the functional structure of each query word, and from observing the interactions with other words.
(b) Natural language is undoubtedly the most flexible query language. The pitfalls which present themselves during an attempt to program natural language are of interest for the theoretical foundations of syntax and semantics in linguistics.
(c) The queries required for this data set involve distinguishing the use and mention of certain verbs which may show up in both functions. Since little is known of the linguistic properties of English metalanguage, such a system provides one possible controllable situation in which metalanguage properties, in addition to the data set properties, may be studied for their own interest.

## 3. THE SEMANTIC STRUCTURE OF QUERIES

In the traditional distinction between yes/no questions and wh- questions, linguists have tended to represent a question such as (1a) semantically as (1b), and a question such as (2a) as (2b):

(1a) Did Max come?
(1b) I ask that (you tell me whether(
(Max came)OR NOT(Max came) )

(2a) Who came?
(2b) I ask that (you tell me whether(
(Max came) OR
(Fred came) OR
:
:
)

where the potentially infinite disjunction in (2b) is formed from substituting in the sentential formula "x came" the name of each individual in the universe of discourse which obeys the selection restriction for "came". Because of the occurrence of "I" and "you" in the semantic paraphrases (1b) and (2b), a true semantic analysis would require a model of both the interrogator and the system in much more detail than we wish to provide.

We therefore adopt the following conventions in accordance with Colmerauer (1977):

Q1. We can reply in the affirmative to a yes/no query provided that we can prove the corresponding declarative sentence (cf. the first part of the disjunction in (1b))
Q2. To a wh- question containing "which P", where P is representable as a predicate of one argument, we may reply "n" if n is a proper name such that P(n) is true and the expression obtained by replacing "which P" by "n" in the declarative form of the query is true.

As an example of Q2, consider the query "Which verb is NPN an object type of?". We may reply "accuse" (among other answers) if we can show that verb(ACCUSE) and "NPN is an object type of ACCUSE", or rather its semantic representation, can be proved as theorems.

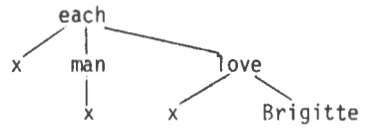
### 3.1 Quantifier expressions

Colmerauer's proposals for quantifiers are in fact much more precise and far-reaching than indicated above. His treatment of quantifying expressions (including articles) such as le,un, chaque and aucun in French assumes that each such expression acts as a kind of semantic functor, creating a complex quantifier expression out of a variable, x, and two sentential functions in which x occurs free. Consider the simple sentence "Each man loves Brigitte", whose translation in standard predicate logic is:

$$(\forall x)(man(x) \rightarrow love(x,Brigitte)).$$

Note that in this translation the variable x appears in two simpler sentential functions, man(x) and love(x,Brigitte). The resultant complex quantifier expression is represented as:

(3)




or, linearly:

(4) each(x,man(x),love(x,Brigitte))

For each quantifier, the corresponding complex expression can be converted by simple rules into a form more easily evaluated by PROLOG. In particular, (4) becomes:

(5) not(exist(x,and(man(x),not(love(x,Brig)))))

What is significant about this treatment is the fact that when all quantifiers and articles are given a lexical representation calling them predicates of four arguments, e.g.

(6) each(x, s1, s2, s3)

variable — sentential formulas — resultant complex quantifier expression

it is possible to maintain full control over the construction of semantic representations during unification of variables. This is particularly important when sentences with several quantifiers must be given the proper hierarchical semantic structure. Colmerauer's hypotheses for these construction principles are incorporated into the parsing rules of §5.

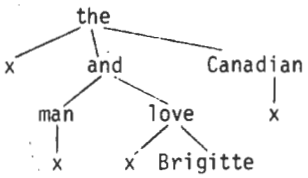
### 3.2. Relative clauses

Restrictive relative clauses serve to delimit the set of possible referents of a noun phrase, over and above the conditions on reference specified by the noun itself and any prenominal modi-

fiers. This fact is reflected in the representation of a relativized noun phrase as the conjunction of two sentential functions, both containing a free occurrence of the same variable. For example, (7) is represented as (8), where the underlined portion of (7) corresponds to the portion of the tree within the dotted line.

(7) the man who loves Brigitte is Canadian

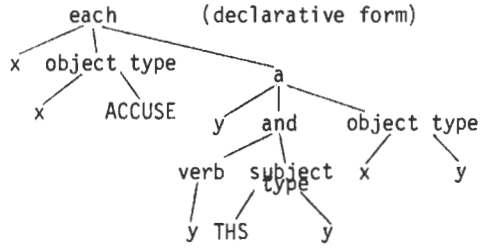
(8)




Given these conventions on representing quantifier expressions and relative clauses, it is now possible to give a representation of a more complex query:

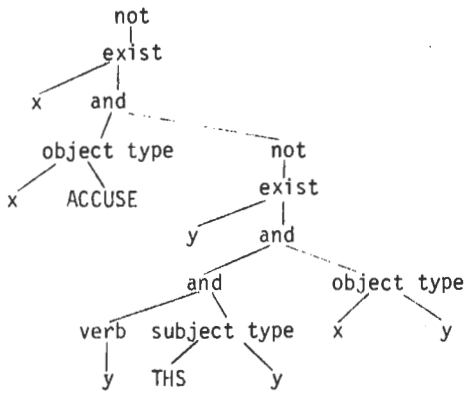
(9) Is each object type of ACCUSE an object type of a verb whose subject type is THS?

(10)




In the form evaluated by METAVERB this becomes:

(11)




3.3 Wh- questions

The word which (or what) also falls under the uniform treatment of quantific expressions outlined above. In the query:

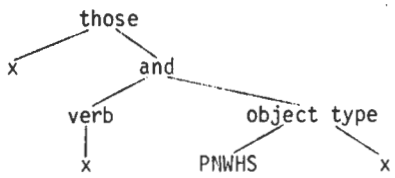
(12) Which verbs is PNWHS an object type of?

what is requested is essentially:

(13) (The set of) those x such that
x is a verb and
PNWHS is an object type of x

We set up a semantic predicate "those", similar to Colmerauer's "ces" giving a two-branched semantic tree for (12) as:

(14)




## 4. PROLOG AND THE CONTROL COMPONENT

### 4.1 Some relevant features of PROLOG

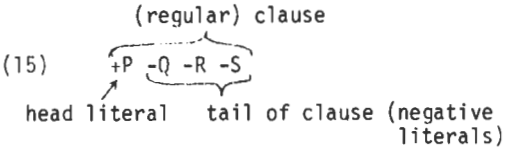
We summarize here some of the basic features of PROLOG given in the introduction to Colmerauer, Kanoui & Van Caneghem (1979).

PROLOG is a very high level programming language based on first order logic. The only operation in this language is the unification of two logical terms, which are represented as trees.

A program in PROLOG can be viewed in both a declarative and a procedural perspective. Declaratively, it corresponds to a set of regular clauses in first-order logic. These clauses express the relations between the various subparts of a proof. Procedurally, the program, with the help of a simple control language, describes how the theorem prover constructs a proof.

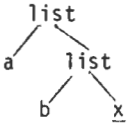
On the syntactic level a program is a set of regular clauses, each of which consists of an ordered sequence of literals. The head literal (or positive literal) is followed by zero or more negative literals, which make up the tail of the clause. For example:

(15)




A literal is an atomic formula of logic (predicat) or its negation. A predicate expresses a relation among terms, which are the objects manipulated by PROLOG. In the tree

(16)




(also represented as list(a,list(b,x)) )

the variable x is underlined to distinguish it from the constants a and b and the binary function "list". The name of a variable is local to the clause in which it appears. The following simple program calculates the concatenation of two lists:

+concat(list($x$,$\ell_1$),$\_2$,list($x$,$\ell_3$))-concat($\ell_1$,$\ell_2$,$\ell_3$)
+concat(nil,$\ell$,$\ell$)

Here "nil" is an atom.

An instance of a term or clause is obtained by uniformly substituting terms for variables. The declarative semantics defines those predicats which can be proven true. +P', the head of an instance +P' -Q' -R' -S' of the clause +P -Q -R -S is said to be true if all the predicates in the tail of the clause, Q',R',and S' are true. This definition does not make use of the order of the clauses or the order of the literals within a clause.

A bundle of clauses is a set of ordered clauses, all having the same head predicate.

The procedural semantics interprets each bundle as a procedure definition, each clause representing one possible definition of the procedure. The literals of the tails of the clauses correspond to procedure calls.

Resolution consists of executing a sequence of procedure calls. Unification is the operation which searches for the most general common instance of two terms. In other words it is the search for those substitutions which particularize variables the least while still making the two trees equal. To execute a procedure call P, the interpreter searches, in the bundle corresponding to the definition of P, for the first clause for which the head literal can be unified with P. If this unification succeeds, the unification of the tail of the instantiated clauses is attempted (unless the tail is empty), resolving each literal in order. If the tail is empty, we attempt to resolve the next literal in the waiting list. If the unification fails, or if no clause is found with a head which can be unified against the predicate to be resolved, then backtracking occurs. Upon backtracking, the interpreter returns to the last choice point, undoing all substitutions made after that point and then starts off in the new direction, trying to resolve the same predicate with another clause.

Certain predicates are interpreted directly and do not pass through the resolution mechanism. Some, such as plus(x,y,z), defined for integers, could be written in PROLOG. Others, such as those which read or write a character, can only be defined procedurally.

4.2 The Question-Answer control component

We give here only the essential parts of the control section, which can be written as five clause bundles:

```
+GO -AJOUT(+(AGAIN).NIL) -LIGNE -LIGNE -COMEON.

+COMEON -SENTENCE(*L) -TESTEND(*L).
+COMEON -LIGNE -LIGNE -AGAIN -COMEON
    +TESTEND(GOODBYE.*X) -LIGNE -SORM("SO LONG")
        -/ -SUPP(+(AGAIN).NIL).
    +TESTEND(*L) -ANALYZE(*L,*S) -LIGNE -SORT(*S)
        -LIGNE -LIGNE -ANSWER(*S).

+ANSWER(THOSE(*X,*S)) -/ -TRUE(*S) -SORT(*X)
    -SORM("!").
+ANSWER(*S) -TRUE(S) -/ -SORM("YES!").
+ANSWER(*S) -/ -SORM("NO!").

+ANALYZE(*L,*S) -SYN(QUEST(*S).NIL,*L) -/.
+ANALYZE(*L,*S) -LIGNE -SORM("IT'S HIGH TIME
    YOU DID SOMETHING ABOUT YOUR SPELLING!")
    -LIGNE -IMPASSE.
```

The relational symbols AJOUT,LIGNE,SORM,SUPP,SORT, SYN, and IMPASSE are all predefined in PROLOG. All others in these clauses are defined within METAVERB. When the goal statement -GO! is entered the first clause becomes applicable. Evaluation of the first negative literal causes the clause +AGAIN to be added to the system, allowing for repetition of queries. Two lines are skipped and COMEON is evaluated. When a query has been entered, this triggers the evaluation of SENTENCE followed by TESTEND. The first of these calls up a bundle of clauses, not given here, which read the input characters and convert that string to a list of words. TESTEND first checks whether the input is a closing statement "goodbye..." in which case a reply is printed and the repeat clause +AGAIN is suppressed. Otherwise, TESTEND tries to analyze the word list *L into the question structure *S, print the structure (which is the value of) *S and provide an answer to *S. ANALYZE(*L,*S) succeeds if the tree structure QUEST(*S) can be constructed using the metamorphosis grammar rules (called by the predefined relational symbol SYN) out of the word list *L. Failing that, a message is printed. If the tree *S in the last negative literal of the fifth clause is of the form THOSE(*X,*S) we use the sixth clause and try to show that *S is true for each such *X, and if successful we print the value of *X followed by "!". If the *S tree of the fifth clause was not of the form for a wh- question, we can answer "YES!" provided that *S can be shown true. Otherwise the answer is "NO!".

4.3. The semantic component

All queries handled by the METAVERB system as described here can be analyzed into semantic trees using the seven predicates those,exist,and, not, verb, subject type, and object type. The semantic component, then, specifies the semantics of these predicates, as well as that for TRUE:

```
+TRUE(*S)  -*S.

+EXIST(*X,*S) -*S.

+AND(*S1,*S2) -*S1 -*S2.

+NOT(*S) -*S -/ -IMPASSE.
+NOT(*S).
```

```
+VERB(*X) -VERB(*X,*L1,*L2).

+SUBJECT TYPE(*X,*Y) -VERB(*Y,*L1,*L2)
    -DANS(*X,*L1).

+OBJECTTYPE(*X,*Y) -VERB(*Y,*L1,*L2)
    -DANS(*X,*L2).
```

A formula *S is said to be true if *S can be proved from the data base. There exists an *X such that *S provided *S can be proved. We can prove AND(*S1,*S2) provided we can prove both *S1 and *S2. To show NOT(*S) we specifically try first to prove *S, and if successful bring the system to a halt. Failing to find any contradiction, we may conclude NOT(*S). To show that *X is a verb it suffices to find *X as the first branch of a VERB tree in the data base. To show that *X is the subject (object) type of *Y, it suffices to show that *Y is in a VERB tree of the data base and that *X is in the list of subject (object) types, given as the second (third) branch of that tree. The relation DANS, which is easy to define declaratively, is predefined.

## 5. PARSING QUERIES IN METAMORPHOSIS GRAMMAR (MG)

Metamorphosis grammars were designed by Colmerauer(1975) to facilitate the syntactic and semantic analysis of natural languages and other sets of complex strings within the general framework of PROLOG. Rules which manipulate strings of trees are expressed directly, with the possibility of representing subparts of trees by means of variables. In this respect MGs bear a certain resemblance to Colmerauer's Q-system grammars (1971).

MG rules, as they are used in METAVERB, are written like context-free production rules, except that non-terminal symbols may be complex, having the general form of tree structures, as elsewhere in PROLOG. Non-terminal elements are preceded by the symbol ":" and terminals, by "#". For example the two rules:

```
:NCOMP(*X.*L,*S1,*S) == :NCOMP(*L,*S1,*S2) #OF
                        :NP(*X,*S2,*S).
:NCOMP(NIL,*U,*U) ==.
```

state that a noun complement may, in the general case, consist of a noun complement followed by the string "OF" followed by a noun phrase. Or, :NCOMP(...) may be the empty string. The variables within the non-terminal trees are arranged so that when values are assigned during unification, there is control over the recursive process by which pieces of the semantic representation are built up, corresponding to the recursive analysis of syntactic constituents.

### 5.1 Example

The process of syntactic-semantic analysis can be seen by following a simple query:

(17)Is NPN an object type of ACCUSE?

through the following MG rules (numbering serves only for reference in this discussion):

```
① :QUEST(*S) == :YESNO(*S).
② :YESNO(*S4) == :BE(*NB) :NP(*NB-*X,*S3,*S4)
     :ART(*NB-*X,*S1,*S2,*S) :NC(*NB-*X.*L,*S1)
     :NCOMP(*L,*S1,*S3) #?.
③ :NP(SIN-*X,*Z,*Z) == :NPROP(*X).
④ :NP(*X,*S2,*S4) == :DET(*X,*S1,*S2,*S3)
     :NC(*X.*L,*S1) :NCOMP(*L,*S3,*S4)
⑤ :NCOMP(*X.*L,*S1,*S) == :NCOMP(*L,*S1,*S2)
     #OF :NP(*X,*S2,*S)
⑥ :NCOMP(NIL,*U,*U) ==.
⑦ :NPROP(*X) == #*X -OBJTYP(*X).
⑧ :NPROP(*X) == #*X -VERB(*X,*L1,*L2).
⑨ :ART(SIN-*X,*S1,*S2,EXIST(*X,AND(*S1,*S2)))
     == :A.
⑩ :A == #AN.
⑪ :BE(SIN) == #IS.
⑫ :NC(SIN-*X.*NB-*Y.NIL,OBJECTTYPE(*X,*Y))
     == #OBJECT #TYPE.
```

Rules 7 and 8 show an important difference from the others. A negative PROLOG literal appears to the right of the MG rule proper, indicating that the rule applies only if the literal can be proved.

The question-answer control component, discussed in §4.2 above, is activated upon receipt of a query terminated by "?". After some initial steps, we are in the position of trying to show +SYN(QUEST(*S).NIL,*L), as seen in the next-to-last rule of that section. The predefined relation SYN calls the metamorphosis grammars, trying to unify variables in such a way that the sentence list *L can be generated from the tree QUEST(*S). We can develop the left-hand member of rule ① in the form of the right-hand member of rule ② by setting *S of rule ① equal to *S4 of rule ②. The first non-terminal of the right of rule ② can be resolved against the input terminal "IS" by applying rule ⑪. Hence the value of *NB in ② becomes SIN. In order for :NP(SIN-*X,*S3,*S4) to match against the proper name NPN, which is the next element of the input sentence string we must apply ③ and ⑦, which returns the value NPN for *X in rule ②'s string representation. As a result of the application of rule ③ we also have *S3 set equal to *S4, by virtue of the fact that both are set equal to *Z of rule ③. As we continue in this way the values in the variables of the right side of rule ② are progressively instantiated in such a way that we build up a semantic representation. In order to show more precisely how this occurs, we represent in figure 1, on the following page, the direction in which values are assigned to variables for the remainder of the input string, using the instantiations assigned up to this point. We describe only some of the major events in this assignment: (a) When :ART(...) is resolved against #AN by rules ⑨ and ⑩, the value of *S in the top line becomes EXIST(*X,AND(*S1,*S2));(b) When :NC(...) is matched against #OBJECT #TYPE, the value of *S1 becomes OBJECTTYPE(NPN,*X); (c) The value of this last *X becomes equal to ACCUSE by virtue of the match using rules ③ and ⑧;

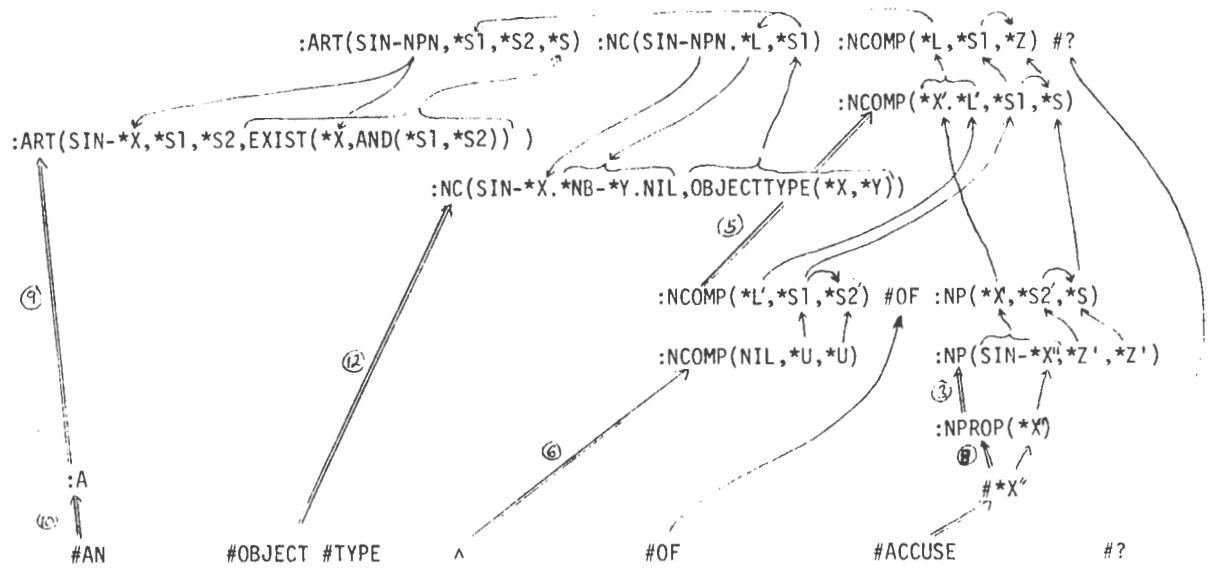



figure 1. Transfer of value assignment during unification of variables

(d) *S1 is put equal to *Z by virtue of certain equalities assigned during the development of :NCOMP(...); (e) *L is set equal to SIN-ACCUSE. It is the value of *S1 in the top line which is transferred, via *Z,*S3 and *S4 to become the value of the variable *S in :QUEST(*S). The final result of the resolution process for this question is :QUEST(OBJECTTYPE(NPN,ACCUSE)). The tree structure containing EXIST(...) is not used in this case.

## 6. PARSING METALINGUISTIC QUERIES

The metalanguage of English ($M_E$) is a subset of the language which constitutes a sublanguage in the sense that it is closed under the transformational operations which can be set up for describing English grammar. The subset of English which is appropriate for describing the syntactic potential of verbs is a tiny subpart of $M_E$, which in fact also constitutes a sublanguage. Although we have not explored the properties of this smaller sublanguage in any systematic way, it is clear that it has a number of special properties which simplify the parsing grammar and the semantic representations. One example is the lack of any tense variation out of the simple present. Another is the restriction on the size of the lexicon used in metalinguistic statements, and the tighter selection restrictions on the verbs (and other predicate words) when used metalinguistically.

Certain pitfalls, however, are also present in metalinguistic parsing, which are relatively uncommon in typical texts. Even in the restricted context of the METAVERB system, we may wish to use a verb which is also mentioned in the study of verb properties:

(18) Does HAVE have a property which no other verb has?

Without resorting to special marks to differentiate use and mention (such as the use of capitals for clarity above), it is virtually always possible to distinguish the two functions for both syntactic and semantic reasons. It is a fortunate property of metalanguage that the shift of levels throws mentioned expressions into the category of proper noun on the higher level, regardless of their category on the object language level. During parsing only by assigning the category NPROP to the first occurrence and V to the second will a successful parse be produced.

In a system such as METAVERB where the semantics of each query expression is tightly controlled, there are also semantic reasons for not confusing use and mention of verbs. The semantics of HAVE as a meta-verb will be defined only to allow words functioning as two-place predicates as its syntactic direct object. Thus any local syntactic ambiguity, such as where the second "have" of (18) is interpreted as direct object of the first, will be weeded out semantically at an early stage.

Some of the problems which have shown up in trying to extend the query language have sharpened our awareness of the semantics of natural language. The sentence (18), for example raises the question of quantifying over properties which have been defined in the system. In the absence of a second-order logic programming language, we must be content to handle such situations in a rather ad-hoc manner.

** This work was supported in part by a CAFIR grant from the Quebec Ministry of Education.

BIBLIOGRAPHY

Colmerauer,A.(1975) "Les grammaires de métamorphose" Groupe d'Intelligence artificielle, University of Marseille-Luminy

Colmerauer,A.(1977) "Un sous-ensemble intéressant du français" Groupe d'Intelligence artificielle University of Marseille-Luminy

Colmerauer,A., Kanoui,H.,Paséro,R. & Roussel,P. (1972) "Un système de communication homme-machine en français" Groupe d'Intelligence artificielle, University of Marseille-Luminy

Colmerauer,A., Kanoui,H., Van Caneghem,M. (1979) "Etude et réalisation d'un système PROLOG" Groupe d'Intelligence artificielle, University of Marseille-Luminy

Kiss,K. & Kittredge,R. (1976) "Verb Tables for English" Contrastive Syntax Project, University of Montreal

van Emden,M.H. (1977) "Programming with Resolution Logic" in Elcock & Michie (eds) Machine Intelligence 8

# TOWARDS SYNTHETIC IMAGES IN SCENE ANALYSIS

Brian V. Funt*
Computer Science Department
State University of New York at Buffalo
Amherst, New York 14226

## Abstract

In the course of interpreting an image of a scene, a machine vision program can evaluate its progress, obtain new clues about the correct interpretation, and be led to new hypotheses by comparing its input image with a synthetic image. This is a hypothesis, and to test it a vision system which uses this type of comparison for feedback is being implemented. The system constructs its synthetic image on the basis of its current understanding of the scene's contents and lighting conditions. Then to evaluate the similarity of the actual and synthesized images, the system compares features extracted from each image. This paper discusses the motivation for the system, gives an overview of its design, and reports on its current status.

## I. Introduction

Research in Machine Vision over the past fifteen years has uncovered many hurdles which are only gradually being overcome. The essential difficulty is that a machine vision program must derive invariant features of a three-dimensional scene, such as object shapes and positions, from highly variable features of the intensity data which are (i) only two-dimensional, (ii) dependent on the lighting conditions and the relative positions of the objects, and (iii) affected by both noise in the camera and noise due to such things as dust specks in the scene. Some of these problems are illustrated by the three scenes in Figure 1. The two-dimensional image of the cube changes from scene (a) to scene (b) as the orientation of the cube changes; the image of the cylinder is affected in (b) by the cube occluding it, and in (c) by the cube casting its shadow on it. Also in (c) changing the light-source position will create a different shadow.

A number of techniques have been developed to deal with these complexities. Horn [1975], Barrow and Tenenbaum [1978], and Marr [1979] have shown how a careful analysis of scene illumination, surface reflectance properties, and the surface characteristics of common objects, can lead to a surprising amount of information about the orientation of surfaces in the scene as well as reasonable estimates of their relative distance. Other researchers (Shirai [1975], Hanson and Riseman [1978]) have built into their programs knowledge of what can be expected in typical scenes, and this knowledge helps guide the scene interpretation process. Baumgart [1974] implemented a three-dimensional geometrical modeling system which was to be used for the type of feedback vision described in this paper. Although he developed many interesting image synthesis techniques, methods for extracting polygon descriptions of two-dimensional images, and a polygon matching algorithm, he did not implement a complete vision system based on comparing actual and synthetic images.

### II.1 Overall System Design

In order to provide an overview of the central ideas in the vision system currently being implemented, I will

*Author's new address: Computing Science, Simon Fraser Univ., Burnaby, British Columbia V5A 1S6.

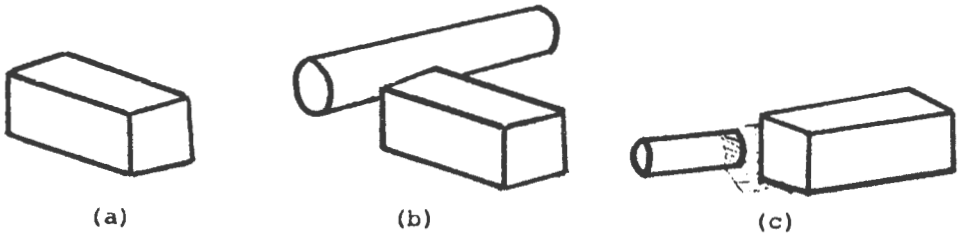



Figure 1

System Overview

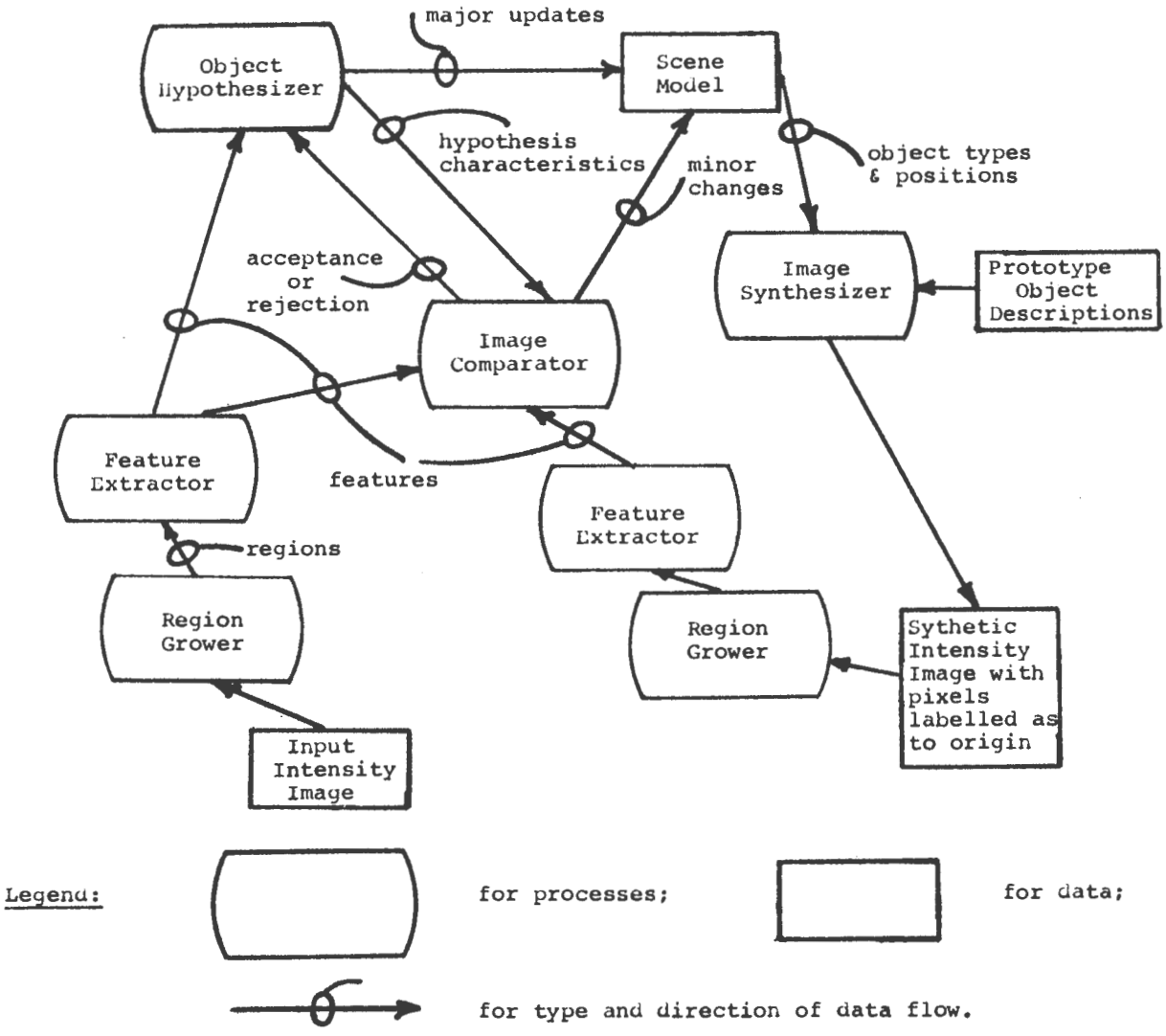



Figure 2

discuss its analysis of the scenes in Figure 1 (the actual input is an intensity array, not a line-drawing). The general aim of the system is to construct a three-dimensional model (useful representation) of the given scene. This means that it must recognize the objects and determine their relative positions and orientations. To do this it must have knowledge of the objects it is to recognize and use some assumptions about the general environment. The following assumptions are made in the system:

(1) all objects are resting on a flat horizontal table
(2) all object surfaces have the same reflectivity
(3) the dominant lighting is with parallel rays from a known direction
(4) there is a uniform low level background illumination
(5) the camera is at a sufficient distance from the objects so that the image is a relatively orthogonal projection of the scene
(6) the system knows the dimensions of the objects it is to recognize

This last assumption means that the system does not recognize the class of rectangular parallelpipeds, but rather the single 2-by-3-by-6 box. This assumption is needed to help determine the distance of an object from the camera. Although for boxes it may seem a bit strong, it is not an entirely unrealistic assumption since humans, it would seem, use knowledge about the standard sizes of such objects as telephones and typewriters.

II.2 Feature Extraction

The first processing stage (see Figure 2) segments the input intensity image into regions using techniques similar to those of Brice and Fennema [1970]. These regions are then input to a special set of parallel feature-extraction algorithms (Funt [1980]) which handle the regions as patches of area rather than just processing their boundaries. The feature-extraction algorithms determine such properties of a region as its area, its center-of-area, its neighboring regions, its corners (if any), its degree of bilateral symmetry at each of 36 orientations about its center-of-area, and its degree of rotational symmetry also at each of 36 different orientations. In addition the feature-extraction algorithms can find the degree of similarity of two two-dimensional shapes under any combination of translation, rotation, and scaling. Rather than finding all features of all regions in the image in advance, they are computed on request by the object-hypothesizer or feature-comparator.

II.3 Object Hypothesis

The object-hypothesizer uses the features available from the feature-extractor to conjecture which objects are in the scene and to produce a rough estimate of their positions and orientations. This is the least developed part of the system because it is not central to testing the suggestion that synthetic images might be useful in scene analysis, and there has also been a good deal of work on the problems of recognition and scene labelling (Roberts [1965], Nevatia and Binford [1977], Waltz [1972], Guzman [1968], Falk [1972], Mackworth [1973]). But there is a difference here because the object-hypothesizer need only make reasonable guesses as to the objects and their positions. The guesses it makes will be either rejected, confirmed or improved upon by the image-comparator at a later stage in the processing.

The hypothesizer makes hypotheses about only one object at a time starting with any unoccluded objects first. It is organized as a set of "specialists"--one

for each object in its repetoire. Each specialist decides whether a subset of the regions under consideration could possibly have arisen as a result of the presence of its object in the scene. The box-specialist, for example, looks for evidence such as:

(1) three mutually abutting regions
(2) approximate symmetry under a rotation of 180° of the overall shape (symmetry is always estimated about a shape's center-of-area) of the three regions considered as one large region.
(3) approximate symmetry under a rotation of 180° of each of the three regions taken independently.

When satisfied that a set of regions may have resulted from the box, the box-specialist then estimates the box's orientation. For this it uses the angle that a horizontal line would make with the bottom edge of one of the lower regions of the box. By measuring the area of one of the box sides in the image, compensating for the effect of the known rotation, and comparing the resulting area with the side's area stored in the box model, an estimate of the box's distance is obtained.

II.4 Image Synthesis

Since it is possible that some other object might have caused the same set of image features, the hypothesis, a box in this example, must be tested further. In order to evaluate it, the system generates a raster graphics image which contains a box of the same size and position as the one it has hypothesized to be in the actual scene. An existing computer graphics system (Herman [1979]) has been used for the image synthesis. Modifications are being made so that each pixel in the synthetic image will also carry a label identifying the object or object faces which the pixel represents. The pixel intensities and labels can be accessed by the feature-extractor and then used by the image-comparator. It compares the synthetic and actual images of the scene, and accepts the box-hypothesis if the two are sufficiently similar.

II.5 Image Comparison

Horn and Brachman [1978] had considerable success with statistical matching of images on a more constrained domain, but in general two images will not match very well on a pixel-by-pixel basis. In addition, we want the system to be able to determine how to modify its current hypothesis in order to improve the match. For this we need more information from image comparison than a single statistical measure of success or failure. The solution is to compare the images at the level of the features found by the feature-extractor. These features are more meaningful for comparison than the actually pixel intensities because they result from global properties of the object's position and orientation as well as the illumination and imaging process. A region missing in the synthetic image can be a sign that the wrong object has been hypothesized; a region of slightly the wrong size, but the correct shape, on the other hand, is more likely to indicate that the object has simply been placed at the wrong depth. The system's strategy, therefore, is to use the same feature-extraction algorithms on the synthetic image as on the actual image, and then compare the corresponding sets of features. Discrepancies discovered in the comparison lead to a further analysis of their probable cause, an improved hypothesis, and a new synthetic image.

The first task of the image-comparator is to make sure that the current hypothesis doesn't invalidate any of the previously accepted hypotheses. This

would occur, for example, if a new object was hypothesized at a depth which placed it in front of an already-accounted-for object. Since the pixels are labelled as to their origin, the image-comparator can detect when a newly hypothesized object occludes an established object, thus invalidating either the current or previous hypothesis. When this happens, the current hypothesis is thrown out and the hypothesizer is called and informed of the reason.

From this point on the image-comparator can concentrate solely on those portions of the two images which are affected by the current hypothesis. It first counts the number of regions the feature-extractor finds in that part of the synthetic image with pixel labels representing the newly hypothesized object. The hypothesizer, when it calls the comparator, passes it a list of the regions it thinks that it has accounted for in the input image as well as a designation of which part of the object each region represents. Ideally, the number of regions found in the synthetic image will correspond to the number the hypothesizer expects it has accounted for, and it will be possible to correlate the regions from the two images with one another. However, even in cases where the hypothesis is correct this may not always be the case. The synthetic image is produced by an imperfect graphics system which has only an approximate model of the surface reflectivity and lighting conditions and does not take into account such factors as self-illumination, while the input image is produced by an imperfect camera which introduces noise and distortion.

The first stage in finding the correspondence between the individual regions is to compare the overall shape of the object as it appears in each image. The overall shape of the object in one of the images is defined by the union of the regions which make it up. The similarity tester finds the translation, rotation and scaling required to overlay the two shapes as closely as possible. If there is a relatively high degree of similarity in the overall shape then the comparator proceeds to the next stage in finding correlates for the individual regions; otherwise, it can use the lack of similarity to adjust the hypothesis. In particular, if one shape needs to be scaled in order to correspond to the other then this is strong evidence for a translation in depth of the hypothesized object. On the other hand, if one of the shapes needs only to be moved in order for it to correspond to the other then this is evidence for a translation parallel to the image plane. A complete lack of similarity in the overall shapes is evidence that the hypothesis is totally incorrect. If a translation is called for, the image-comparator updates the current world model and calls the image synthesizer to generate a new synthetic image (see Figure 2).

Once the image-comparator receives a synthetic image in which the overall shapes are in the same relative positions and are of the same size, it begins establishing the correlation between the individual regions in the two images. It takes the largest of all the regions, L, and sees which regions from the other image would lie under it (minor overlap is ignored) if the two images were superimposed. Then L's shape is compared with the combined shape of the regions it covers. If there is a reasonably good match without any translation, rotation or scaling then the relationship between these regions from the two images is recorded, and an attempt is made to find a corresponding set of regions for

the next largest region. This process is repeated until all the regions have been exhausted. There is one further circumstance which must be taken care of and that is when L only partially covers one of the regions, R, from the other image. In this case the regions R covers in L's image are found and combined with L. If a satisfactory mapping between the regions of the two images cannot be found then the comparator calls the object hypothesizer and rejects the current hypothesis.

After correlating the regions, the image-comparator considers whether a rotation of the hypothesized object is required. Since it is assumed that the objects are resting on a flat table, any rotation which keeps the same face of the object in contact with the table must be about a vertical axis. The hypothesizer postulates both the object and the face it rests on, so if the object cannot be made to match using rotation about only a vertical axis then the hypothesis is rejected. If there is a good match between the correlated regions of the last step then no rotation is necessary and the comparator accepts the current hypothesis. A poor match between some of them, on the other hand, may be resolved by a small rotation. A rotation will change the shapes in the image because they are two-dimensional projections of surfaces in three-dimensions.

The comparator estimates the direction and angle of rotation by comparing the sizes of correlated regions. For simple objects such as boxes and cylinders, a clockwise rotation (as viewed from above) will cause regions from the right of the object to grow and regions from the left to shrink. The more different the sizes of the related regions, the further the object will have to be rotated. Whenever the need for a rotation is established the world model is updated and a new image is synthesized.

When the image-comparator is satisfied and accepts the current hypothesis, it calls the object hypothesizer which attempts to make a hypothesis about one of the other objects in the scene. When the hypothesizer is unable to generate any new postulates, execution terminates.

## III. Concluding Remarks

I expect the benefits of this approach--image feature extraction, object hypothesis, image synthesis, feature comparison, hypothesis adjustment --will arise most clearly on the more complicated scenes of Figure 1 (b) and (c). In them the two objects interact, in (b) by occlusion and (c) by shadowing. After the box is confirmed, there will be sufficient features to hypothesize the presence of the cylinder, but not enough features to confirm the hypothesis directly. The technique of synthesizing an image of the hypothesized scene is particularly powerful in this case, because in the synthetic image (assuming the hypothesis is correct) the same portions of the cylinder will be occluded or in shadow as in the actual image; therefore, the features derived from the actual image will match those derived from the synthetic image even though they would not have matched those of a standard cylinder in isolation.

To improve the performance of the system, parallelism can be used in both feature extraction and image synthesis. In fact, one of the advantages to the system of using synthetic images is that they are a geometrical representation of its hypotheses to which parallel processing can be applied. A parallel processor consisting of at least 1000 individual processors each connected to its immediate neighbors via communication links has been assumed. For the time being this processor is simulated on available

sequential hardware. To process an image, it is first mapped onto the entire parallel processor in such a way that each individual processor represents one small portion of the image, and neighboring processors represent neighboring image regions. By mapping small image areas onto some processors and large areas onto others, the parallel processor as a whole "sees" the image in varying degrees of detail in a manner analogous to the way people see less detail at the edge of their field of view. If the mapping from image to parallel processor is done correctly, then rotation and scaling of two-dimensional shapes can be performed by simple neighborhood communication between individual processors. This is the same basic parallel processing structure as used in WHISPER (Funt [1980]), and some of the same feature-extraction algorithms have been used directly (e.g. center-of-area finding, similarity test), while others have required improvement (e.g. the symmetry test, vertex finding).

The implementation of the system is well underway, and some preliminary experimental results are expected soon. At this time many of the feature-extraction algorithms are running, the image synthesis problem has been largely overcome by interfacing the system's three-dimensional world model to an existing image synthesis system (Herman [1979]), and the object-hypothesis, image-comparison and hypothesis adjustment procedures are partially coded for the case of an isolated object.

Acknowledgements

Roland Berg, Alan Mackworth, Tony Maida, Dennis Martin, Jeff Posdamer, Stuart Shapiro, Mann-May Yau, and Han Yong You have all contributed to this work. The financial support of the University of Buffalo Foundation is also gratefully acknowledged.

References

Barrow, H.G., and Tenenbaum, J.M., [1978] Recovering Intrinsic Scene Characteristics from Images, SRI International TN-157.

Baumgart, G.B., [1974] Geometric Modeling for Computer Vision, PhD Thesis, Stanford University AI Memo 249.

Brice, C.R., and Fennema, C.L., [1970] "Scene Analysis Using Regions", Artificial Intelligence, vol. 1, pp205-226.

Falk, G. [1972] "Interpretation of Imperfect Line Data as a Three Dimensional Scene", Artificial Intelligence, vol. 3, pp.101-144.

Funt, B.V., [1980] "Problem-Solving with Diagrammatic Representations", Artificial Intelligence, (in press).

Guzman, A., [1968] "Decomposition of a visual scene into three-dimensional bodies", AFIPS Fall Joint Computer Conf., vol. 33, pp.291-304.

Hanson, A.R., and Riseman, E.M., [1978] "VISIONS: A Computer System for Interpreting Scenes", Computer Vision Systems, (ed.) Hanson and Riseman, Academic Press.

Herman, G.T., [1979] Representation of 3-D Surfaces by a Large Number of Simple Surface Elements, SUNY at Buffalo Medical Image Processing Group Technical Report 26.

Horn, B.D., [1975] "Obtaining Shape from Shading Information", The Psychology of Computer Vision. (ed.) Patrick Winston, McGraw-Hill Co.

Horn, B.K., and Brachman [1978] "Using Synthetic Images with Surface Models", Communications of the ACM, vol. 21 #11, pp.914-924.

Mackworth, A. [1973] "Interpreting Pictures of Polyhedral Scenes", Artificial Intelligence, vol. 4, pp.121-137.

Marr, D., [1979] "Visual Information Processing: The Structure and Creation of Visual Representations", Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, August 1979.

Nevatia, R. and Binford, T.O., [1977] "Description and Recognition of Curved Objects", Artificial Intelligence, vol. 8, pp.77-98.

Roberts, L.G., [1965] "Machine Perception of Three-Dimensional Solids", Optical and Electro-Optical Information Processing, (ed.) Tippet et al., MIT Press.

Shirai, Y., [1975] "Analysing Intensity Arrays Using Knowledge About Scenes", The Psychology of Computer Vision, (ed.) Patrick Winston, McGraw-Hill.

Waltz, D., [1972] Generating Semantic Descriptions from Drawings of Scenes with Shadows, PhD Thesis, MIT AI TR-271.

# Mediation Between Central and Peripheral Processing: Useful Knowledge Structures

Roger Browse
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5

## Abstract

This paper outlines a computational vision research project aimed at the development of techniques for the mediation between central and peripheral processes. The key ingredients are structural relations between image and scene domain hierarchies, and a representation which emphasizes the dependencies that exist within the knowledge. These constructs may be used to select areas of the image to process in greater detail on the basis of the progress of interpretation; the nature of the task which is motivating the visual system; and the contents of both peripheral and foveal vision.

## 1. Introduction

Information extraction is an essential component of intelligent behaviour. This extraction often involves the selection of sequences of intense, localized processing within contexts of less detailed global processing. The locomotion of an organism through its environment provides locations from which detailed information may be obtained. Of course the extent to which information may be received from locations nearby may vary, but it is always constrained by the capabilities of the organism.[1] Similarily the saccadic eye movements of the visual system, together with the acuity structure of the retina, provides a sequence of select locations in the visual environment which may be intensely processed within the context of acuity-limited peripheral vision.[2] Visual attention may also be viewed as the selection of locations for more detailed processing, both within the information available in a single fixation of the eyes, and within the knowledge which is involved in the visual processing.[3] To extend the idea to its limit, thought may be a sequence of availabilities of information within the mind.[4]

Visual attention and thought have no directly observable manifestations as do locomotion and eye movements, and hence the patterns of their operation must be inferred from psychological experimentation and from introspection and will thus always remain speculative.

The interesting questions which arise are:

1. Is there some uniform computational process which can mediate between detailed, local information extraction and less detailed, global information extraction?
2. What is the role of such a process

---

1. Rowat (1979) has implemented a computer model which demonstrates the intricacies of these processes.
2. see Rayner (1978) for a summary.
3. see Kahneman (1973) for an explanation of these different forms of visual attention.
4. see Bartlett (1932) for a discussion of the ballistic nature of thought.

in intelligence?

This paper describes part of a research project which is aimed at the development of answers to these questions. The context is a computational vision system which interprets line-drawings of human-like body forms.[1] Of particular concern is the interaction between interpretation and the selection of locations to process with foveal acuity.

## 2. A Computational Model

A fixation within the image is represented computationally as the availability of information at different levels of a detail hierarchy: at any given time, a small location will have the actual lines of the drawing available, and the periphery has available only information about the density of line in each unit area. In addition, a data structure is available which encodes knowledge about the hierarchical relations among body parts, and represents the possible configurations that comprise different postures. Thus a very clear distinction is maintained between image and scene domains (Clowes, 1971). The key to the operation of the system is the relation between these two hierarchical structures. Primitive image features can be detected at any of the detail levels, and these features act as cues for the existence of scene domain elements at a corresponding level of the body form knowledge hierarchy. For example, a foveally located line vertex (the most detailed level) may act as a cue for the connection of a finger to the hand, while a vertex at some coarser level may cue the connection of an arm to the body.

Previous computational vision systems have exploited the power of operating at more than one level of detail in an image (Kelly, 1971; Shirai, 1973) but the present system differs through the introduction of structural dependencies between model knowledge hierarchies and these levels of available image information. Such structural dependencies permit the interaction of processing at these levels in a way which is (at least) analogous to the interaction of peripheral and foveal vision.

Computational vision systems often utilize the relations found within the scene domain (see Havens, 1978; Mulder, 1979). Usually, however, higher-level scene domain elements are confirmed by a structure which relies on low-level input information. Issues of top-down vrs. bottom-up processing are then addressed within a context similar to that of grammatical analysis in which the existence of any non-terminal is supported by evidence which is ultimately traceable to a single level of input representation (the terminal string). Processing top-down or bottom-up, the resulting support evidence is the same. Addressed within Psychology, issues of the direction of processing assume the availability of support for high level scene domain elements independent of low level information (Kinchla and Wolfe, 1979; Mermelstein, Banks and Prinzmetal, 1980).

The present system has been designed to be responsive to several influences in the selection of locations to process within an image: the ongoing interpretation and the critical

---

1. The drawings are similar to those used to illustrate the movement notation of Eshkol and Wachmann (1958).

ambiguities which arise; the nature of the task which is motivating the visual process; and the contents of both foveal and peripheral vision. As a result of these requirements, a declarative structure has been chosen which will make explicit the details of the hierarchical relationships among components. The remainder of this paper will describe that knowledge structure and point out some of the more interesting processing capabilities which it provides.

## 3. Representation

The body form knowledge is centered around schemata-like structures called CONCEPTS,[1] which may be related to one another via SUBCONCEPT relations, and which may be related to entities in the image through INSTANCE relations. For example:

> The CONCEPT hand may have SUBCONCEPTS left hand and right hand. One structure in the image may be related by an INSTANCE pointer to the left hand CONCEPT, while another may be an INSTANCE of the CONCEPT hand.

From the example it is shown that incomplete knowledge can result in image entities being specified as INSTANCES of more abstract CONCEPTS. The binding of image entities to the CONCEPT structure is a critical aspect of the interpretation process, and requires a richer form of representation than the simple INSTANCE pointer (see Stefik, 1979). This discussion, however, will center on the internal structure of the CONCEPTS, which has been fashioned in a way similar to that of Stanton (1968,1971).

Each CONCEPT has a NAME,ATTRIBUTES and DESCRIPTIONS. DESCRIPTIONS are named lists of CONCEPTS which characterize the described CONCEPT. For example:

> The CONCEPT arm has the part-of DESCRIPTION giving the list (upper-arm lower-arm hand).

CONCEPTS may have ATTRIBUTES. For example:

> The CONCEPT arm may have ATTRIBUTES such as posture, orientation, etc.

Only entities in the image may take on actual ATTRIBUTE values. The CONCEPT does, however, specify the range of values that ATTRIBUTE can take on, and also specifies other ATTRIBUTE values which are necessary in order to compute the value. For example:

> The development of a value for the ATTRIBUTE posture for the CONCEPT arm (whose values may be either straight or bent) may require values for the ATTRIBUTES orientation of both upper-arm and lower-arm.

Thus the development of ATTRIBUTE values for a CONCEPT may depend on the ATTRIBUTE values of CONCEPTS found in its DESCRIPTIONS.

The existence of a CONCEPT cannot be assumed just because of the existence of the CONCEPTS which are given in one of its DESCRIPTIONS, there are DESCRIPTION RELATIONS which must hold among CONCEPTS, or more specifically, among the ATTRIBUTES of the CONCEPTS. For example:

---

1. Some words used in the description of the knowledge representation are the same as English words. When the meaning within the system is intended, the word appears in capitals. Names of CONCEPTS and ATTRIBUTES appear underlined.

The DESCRIPTION RELATION connect must exist between the ATTRIBUTE distal-end of the CONCEPT upper-arm and the ATTRIBUTE proximal-end of the CONCEPT lower-arm before the existence of these two CONCEPTS can confirm the existence of the more general CONCEPT arm.

This outline of the knowledge structure may be adequate to point out the intricate chain of dependencies among ATTRIBUTES of CONCEPTS. By laying bare these relationships, they may be exploited in the decisions to be made in the processing of images representative of the CONCEPTS. To complete this capability, correspondences have to be established between image domain and scene domain elements.

The advantage of having available all of the dependencies among ATTRIBUTES relies upon the assumption that a vision system is better off spending its energies deciding upon useful locations to process rather than processing in a somewhat arbitrary order and determining the usefulness of the information later. This mode of operation seems most reasonable for human vision because saccades consume up to 250 ms (Yarbus, 1967), and attentional shifts 50 ms (Eriksen and Hoffman, 1974). At least in the case of saccades, there is a serious loss of visual capability during the shift (Latour, 1962). In addition, there is compelling evidence that humans fixate very rapidly on the most important aspects of a scene (Loftus and Mackworth, 1978).

4. Using the Knowledge Base to Select Locations

By examining the knowledge involved in the visual task, intelligent decisions can be made as to candidate processing locations. For example:

> The discovery of an L-vertex at some coarse level of detail may signal either an elbow or a knee. To process in the proximal direction may result in the discovery of a vertex which could either be the connection of the arm or the leg to the body, and hence the ambiguity would not be resolved. More differences are expected in the distal direction toward either the hand or foot, so processing would move in the direction of the distal portion of the vertex.

If the task presented to the vision system can be expressed in terms of ATTRIBUTES to be evaluated, then the dependencies among them can be used to mark the entire set of ATTRIBUTES (at all levels) which are critical to the task. For example:

> If the task at hand is to determine if the body is standing upright, (i.e., if the posture ATTRIBUTE has the value standing-upright), then the method by which this value can be obtained is examined, and it is found that the values of the ATTRIBUTES orientation of the CONCEPTS middle-body and lower-body are necessary. These ATTRIBUTES are marked, as are the requirements for their evaluation, etc. Subsequent decisions about the choice of processing locations will take into account the expectation of finding information relevant to these specially marked ATTRIBUTES.

In the more general case, the existence of a

declarative, analyzable ATTRIBUTE dependency structure provides the basis for the application of an inclusive hierarchy of support for any particular CONCEPT node.

1. A CONCEPT may be only "suggested" through the existence of the CONCEPTS found in one of its DESCRIPTIONS (within some reasonable image area).
2. A CONCEPT may be "confirmed" through the validity of its DESCRIPTION RELATIONS.
3. A CONCEPT may be "understood" through the evaluation of its ATTRIBUTES.

Each of these processes relies on the availability of different ATTRIBUTE values at different levels in the knowledge structure. The different levels of support are dependent on one another. For example, one CONCEPT may have to be confirmed before it can be used to suggest another, and some ATTRIBUTES will have to be evaluated before other CONCEPTS can be confirmed, etc.

The distinction drawn between (1) and (2) above is a computational counterpart of the distinction between attended and non-attended feature analysis in the Feature Integration Theory of visual attention (Treisman and Gelade, 1980). While the identification of features may take place simultaneously over an image, the assemblage or integration of those features into larger perceptual units requires the application of attention which operates sequentially over smaller areas of the image.

## References


Bartlett, F.C.
1932 Remembering Cambridge University Press.

Clowes, M.B.
1971 On Seeing Things. Artificial Intelligence, 2 pp. 79-112.

Eriksen, C.W. and Hoffman, J.E.
1974 Temporal and Spatial Characteristics of Selective Encoding from Visual Displays. Perception and Psychophysics, 12 PP.201-4.

Eshkol,N. and Wachmann,A.
1958 A Movement Notation, Weidenfield and Nicolson, London

Havens, W.S.
1978 A Procedural Model of Recognition for Machine Perception. Ph.D. Thesis, Dpt. of Computer Science, U. of British Columbia.

Kahneman, D.
1973 Attention and Effort. Prentice-Hall, Englewood Cliffs, N.J.

Kelly, M.D.
1971 Edge Detection by Computer using Planning. in Machine Intelligence 6 B. Meltzer and D. Michie (eds.) pp. 379-410.

Kinchla, R.A. and Wolfe, J.M.
1979 The Order of Visual Processing: Top-down, Bottom-up, or Middle-out. Perception and Psychophysics, 25 pp.225-231.

Latour, P.
1962 Vision Thresholds During Eye Movements. Vision Research, 2 pp. 261-262.

Loftus, G.R. and Mackworth, N.H.
1978 Cognitive Determinants of Fixation Location During Picture Viewing. Journal of Experimental Psychology, Human Perception and Performance, 4 pp. 565-572.

Mermelstein, R., Banks, W.P. and Prinzmetal, W.
1980 in press Perception and Psychophysics.

Mulder, J.A.
1979 Representation and Control in a Program that Understands Line Sketches of Houses. M.Sc. Thesis, Department of Computer Science, U. of British Columbia.

Rayner, K.
1978 Eye Movements in Reading and Information Processing. Psychological Bulletin,85,3, pp. 618-660.

Rowat, P.F.
1979 Representing Spatial Experience and Solving Spatial Problems in a Simulated Robot Environment, Ph.D. Thesis, Department of Computer Science, U. of British Columbia.

Shirai, Y.
1973 A Context Sensitive Line Finder for Recognition of Polyhedra. Artifical Intelligence,4 pp. 95-199.

Stanton, R.B.
1969 Graphical Communication and Computer Graphics. Proc. 4th. Australian Computer Conference. pp. 279-286.

Stanton, R.B.
1971 RAMOS: A Description Based Language.

Stefik, M.
1979 An Examination of a Frame-Sturctured Representation System. Proc. IJCAI-79, pp. 845-852.

Treisman, A.M. and Gelade, G.
1980 A Feature-Integration Theory of Attention. Cognitive Psychology, Feb. 1980.

Yarbus, A.L.
1967 Eye Movements and Vision. (B. Haigh trans.) New York: Pelnum Press.

# SCHEMATA-BASED UNDERSTANDING

# OF

# HAND-DRAWN SKETCH MAPS

William S. Havens

Computer Sciences Department
University of Wisconsin
Madison, Wisconsin
53706

Alan K. Mackworth

Computer Science Department
University of British Columbia
Vancouver, British Columbia
V6T-1W5

## Abstract

This paper describes current research in applying schemata-based recognition methods to the understanding of hand-drawn sketch maps. In this system, schemata are employed as representations for models of the cartographic objects and systems of objects possible in sketch maps. The resulting hierarchical network is then searched using a combination of both data-driven and model-driven methods. Low-level models are invoked by primary cues computed directly from the input image. Once invoked, schema models apply object-specific procedural methods to complete their recognition. Completed schema instances are then used as abstract cues to invoke other models higher in the schema hierarchy. A multiprocessing control regime is utilized to permit a number of schemata to apply their recognition procedures concurrently.

## 1. Introduction

In order to cope with the enormous complexity of visual information, computer vision systems must employ extensive model-specific knowledge of the visual world. A major problem in model-driven vision systems is the invocation of appropriate models to interpret a given image. Typically, data-driven methods are employed to generate low-level image cues to select likely models as hypotheses. It has been pointed out that this method is ineffective. Low-level cues are highly ambiguous matching to many inappropriate high-level models.

As a solution to this problem, we are currently integrating model-driven and data-drive recognition in schemata representations by employing a recursive hierarchy of cues and models. Schema models are invoked both by primary cues computed directly from the image and by abstract cues created recursively as the result of recognition. The successful recognition of a schema instance at one level in the hierarchy yields a context-sensitive cue to invoke schema models at higher levels.

Sketch maps have been chosen for this research for the following reasons:

1) We believe that the conventional semantics of of cartography accurately reflects geographic features in real aerial and satellite imagery.
2) The use of vector graphic input data greatly reduces the amount of low-level processing required while still capturing the essential difficulties of geographic image analysis. The research is therefore able to focus on issues of cue generation and model invocation.
3) The enhanced abilities of this approach can be easily compared to a previous sketch map system, MAPSEE, [Mackworth, 1977a] employing a constraint network representation and a network consistency search method [Mackworth, 1975]. By testing both systems on the same input maps, we should obtain a quantitative measure of the expected improvement of schemata over constraint network methods.

## 2. Model-Driven Recognition

Computer vision can be characterized as the task of mapping a two-dimensional sensory image into an abstract symbolic description of the three-dimensional scene represented by that image [Clowes, 1971]. This process necessarily involves the interpretation of sensory signals that are voluminous in their quantity and simultaneously highly ambiguous in their possible meanings. In order to cope with this complexity, com-

puter vision systems must employ both model-driven and data-driven recognition methods. Model-driven recognition utilizes knowledge of the objects and their abstract relationships possible in the visual world. Conversely, data-driven methods exploit spectral knowledge about the signal source and physical knowledge about the processes of image formation and surface recovery. Indeed, the representation and coordinated application of knowledge is the central problem in computer vision [Reddy, 1978].

We are exploring a recognition paradigm for computer vision that integrates top-down, model-driven recognition with bottom-up, data-driven methods in hierarchical schemata-based knowledge representations [Havens, 1978a]. A major problem in model-driven vision systems is the invocation of appropriate models to interpret a given image. In most current systems, data-driven methods are employed to generate low-level image cues to select likely models as hypotheses. Cues can be regions of statistically homogeneous properties or edges inferred from characteristic changes in image intensity. It has been pointed out that this methodology is ineffective [Barrow & Tenenbaum, 1975]. Region and edge-finding algorithms have no knowledge of the real objects to which they may belong. As a result, low-level cues are highly ambiguous matching too many inappropriate high-level models.

As a solution to this problem, we argue that high-level object models must be invoked by appropriate high-level cues. The discovery of such abstract cues is, of course, recursively the recognition problem, thereby necessitating the use of a recursive hierarchy of cues and models. Schema models must be invoked both by primary cues computed directly from the image and by abstract cues created recursively as the result of recognition. The successful recognition of a schema instance at one level yields a context-sensitive cue to invoke schema models at the next higher level.

To realize this recognition paradigm, we are employing a multiprocessing programming language methodology that supports the concurrent execution of top-down and bottom-up search processes in hierarchical knowledge representations [Havens, 1978b].

## 3. Schemata Representations

Recent research has focused on the application of schemata [Bartlett, 1932] as a representation of knowledge [Minsky, 1975] [Bobrow & Winograd, 1977] [Rumelhart & Ortony, 1976]. Schemata have been used or proposed in a number of computer vision systems [Freuder, 1976] [Hanson & Riseman, 1978b] [Brady, 1978]. Schemata are object centered representations which represent complex concepts as specific compositions of simpler schemata thereby forming hierarchical knowledge structures. By exploiting composition, a finite number of schema stereotypes can be used to represent an arbitrary number of object instances. Schemata may contain both active and passive knowledge. Passive knowledge represents descriptive models of stereotypical objects. Active knowledge is represented as procedures attached to schema models to guide the recognition process for instances of those schema stereotypes [Winograd, 1975].

The recognition process in schemata-based systems can be characterized as a search of the schema hierarchy to find a best match of the information present in the input image to the knowledge represented in the knowledge-base. Havens [1976] has shown that this search can be neither a strict top-down nor bottom-up search. Instead, recognition must be an integration of top-down, hypothesis-driven search and bottom-up, data-driven methods [Rumelhart & Ortony, 1976]. Schemata represent models providing expectation and guidance for top-down search. At the same time, features discovered in the image provide cues for the bottom-up selection of particular schemata as likely hypotheses.

We are investigating the integration of model-driven and data-driven recognition by employing both a model hierarchy and a cue hierarchy within a schemata knowledge representation. The interactions between model-driven and data-

driven processes in computer vision are poorly understood [Brady, 1978]. This research is concerned with characterizing that interaction.

A preliminary schemata knowledge representation for sketch maps is illustrated in Figure 3. The nodes in this tree represent schema stereotype models of various cartographic objects and systems possible in sketch map scenes. The arcs represent composition with nodes higher in the hierarchy being composed of connected nodes lower in the hierarchy. The interpretation of the arcs, however, depends on whether a top-down or bottom-up search method is being applied. Using top-down search, the arcs are possible subgoal paths. To recognize a Road-System, for instance, this schema can selectively call the Town, Road and Bridge schemata as subgoals. Using bottom-up search, on the other hand, the arcs represent cue paths to select possible supergoals. As an example, if the Bridge schema has satisfied its expectations for a bridge instance in the input image, it must invoke plausible higher schemata as supergoals. In this case, both Road-System and River-System are very likely to be found in a sketch map scene containing a bridge.

For this sketch map system, the image is represented as plot vectors taken directly from a vector graphics tablet. Conceptually, the data consists of connected image points called Links and blank space called Patches. We have employed a simple recursive quadrant-splitting region finder to yield a conservative first segmentation. For this domain, regions are thought to be poor cues. Instead, a line finder which attempts to connect plot vectors into chains is used to provide primary cues. This algorithm is again chosen to be conservative, forming chains only where the distance between links is small and there is no ambiguity as to chain direction.

## 4. Cycles of Perception

Unfortunately, to completely segment a complex image requires the use of model-specific information about the scenes interpretation, yet that information is only fully available after the segmentation has been performed. In order to avoid this "chicken and egg problem" [Mackworth, 1977b] [Havens, 1976], an integration of low and high-level processing must be achieved. Mackworth [1978] has advocated a "cycle of perception" theory for computer vision to avoid this problem (see Figure 1). Kanade [1977] defines a similar cyclic model. An initial conservative segmentation is used to generate primary cues that invoke appropriate object models. Once invoked, these models can guide a context-sensitive resegmentation of the image, thereby providing new more powerful cues to repeat the cycle.

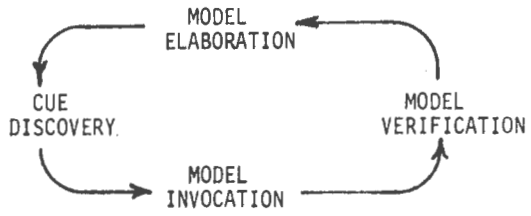



Figure 1

We argue that this cycle of perception can as well be characterized as a recursive process. When all the expectations of a particular model have been satisfied, the instantiated model becomes an abstract cue to recursively invoke appropriate models higher in the knowledge hierarchy (see Figure 2). Instead of relying only on primitive context-free cues, the recognition of schema instances at intermediate levels in the hierarchy can provide context-sensitive cues for the next level.

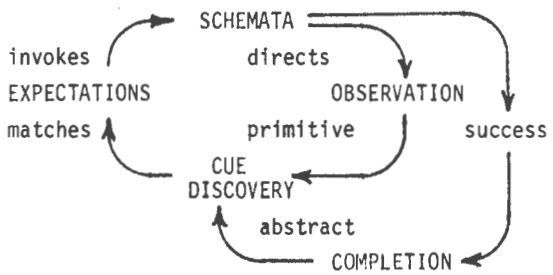



Figure 2

## 5. Programming Methodology

The development of programming methodology for such tasks as computer vision is an active area of research [Bobrow & Raphael, 1974]. Recent work has focused on the development of schemata-based programming languages such as KRL [Bobrow & Winograd, 1977] and MAYA [Havens, 1978b]. Such languages define data structures for representing and accessing schemata and for constructing schemata networks. A method of copying stereotype schemata to provide specific schema instances is also essential. Since schemata may contain both descriptive and procedural knowledge, a mechanism must also be included for allowing attachment of procedures to data within schemata [Winograd, 1975].

The procedures associated with each schema are considered model-specific methods for guiding the search process for that schema. Procedural methods can be used in both top-down and bottom-up search of the schema hierarchy. Both require multiprocessing capabilities. Top-down, subgoal search can be realized by using generators [Sussman & McDermott, 1972] as independent processes that can be recalled on failure to repeatedly attempt alternative solutions to their subgoals.

Bottom-up search requires that a number of models be allowed to be active hypotheses simultaneously. Therefore, each procedural method associated with an active schema must be realized as an independent process. Each such process is allowed to guide the recognition process for its schema stereotype. The coordination of multiple competing processes in goal-directed systems is poorly understood [Brady, 1978]. KRL defines a hierarchy of scheduler processes but leaves the specification of these schedulers to the programmer.

To the contrary, MAYA defines four control primitives for implementing bottom-up, data-driven recognition in schemata networks. The first primitive, PROCESS, creates a new process associated with some schema and begins its execution. This process may attempt to satisfy its schema's model by employing subgoal search or by invoking low-level iconic processes to generate primary cues. If the search is unproductive, the process can suspend itself, using SUSPEND, to simple n-tuple patterns representing the unfulfilled expectations of the schema model. When such information is discovered later by a lower process, the process can be restarted, using RESUME, by a successful pattern match to its pattern. A number of schemata can, therefore, conduct their recognition in pseudo-parallel being activated by the discovery of cues or information matching their model's expectations, applying their methods, suspending themselves when information is not available, and being resumed later by the discovery of additional matching cues of information. See Figure 4.

This iterative cycle continues for each active schema until some schema succeeds in satisfying its model's expectations. If the schema is intermediate in the schema hierarchy, then the completed schema instance is an abstract cue. The control primitive COMPLETE allows this schema to perform two essential control operations. A pattern match determines which higher schemata processes are waiting for the information provided by this completed instance. The completed process is suspended and the matched higher-level processes are resumed, in turn, to continue their own methods.

## 6. Conclusion

This research is concerned with extending the use of model-driven recognition methods in computer vision. By employing a recursive hierarchy of cues and models, represented as schemata, the acknowledged difficulties of invoking models by low-level cues are avoided. By using a schema-based multiprocessing programming environment, a number of models can simultaneously be active hypotheses applying their object-specific methods concurrently. Finally, by testing these techniques on the idealized domain of cartographic sketch maps, both qualitative and quantitative measures of their performance can be obtained.

## References


Barrow, H.G. & Tenenbaum, J.M. (1975) Representation and Use of Knowledge in Vision, Tech. Note 108, Artificial Intelligence Center, SRI International, Menlo Park, Ca.

Bartlett, F.C. (1932), Remembering, Cambridge University Press, England.

Bobrow, D.G. & Raphael, B. (1974) New Programming Languages for Artificial Intelligence Research, Comp. Surveys 6, #3, Sept. 1974, pp. 153-174.

Bobrow, D.G. and Winograd, T. (1977). An Overview of KRL: A Knowledge Representation Language, Cognitive Science, Vol. 1, #1, January 1977.

Brady, J.M. & Wielinga, B.J. (1978) Reading the Writing on the Wall, in Hanson & Riseman [1978a], pp. 283-302.

Clowes, M.B. (1971) On Seeing Things, Artificial Intelligence 2, pp. 79-116.

Freuder, E. (1976) A Computer System for Visual Recognition Using Active Knowledge, Ph.D. Thesis, AI-TR-345, MIT AI Lab, Cambridge, Mass.

Hanson, A.R. & Riseman, E.M. (1978a) (eds.) Computer Vision Systems, Academic Press, New York.

Hanson, A.R. & Riseman, E.M. (1978b) VISIONS: A Computer System for Interpreting Scenes, in Hanson & Riseman [1978a].

Havens, W.S. (1976) Can Frames Solve the Chicken and Egg Problem?, Proc. 1st National Conference, CSCSI, Vancouver, Canada, August 1976.

Havens, W.S. (1978a) A Procedural Model of Recognition for Machine Perception, TR-78-3, Ph.D. Thesis, Department of Computer Science, University of British Columbia, Vancouver, Canada.

Havens, W.S. (1978b) MAYA Language Reference Manual, TM-13, Computer Science Department, University of British Columbia, Vancouver, Canada.

Kanade, T. (1977) Region Segmentation: Signal vs. Semantics, Proc. 3rd IJCPR.

Mackworth, A.K. (1975) Consistency in Networks of Relations, Artificial Intelligence 8, #1, pp. 99-118.

Mackworth, A.K. (1977a) On Reading Sketch Maps, Proc. 5IJCI, MIT, Cambridge, Mass, August 1977, pp. 598-606.

Mackworth, A.K. (1977b) How to See a Simple World, in Machine Intelligence 8, E.W. Elcock and D. Michie (eds.), Halsted Press, New York.

Mackworth, A.K. (1978) Vision Research Strategy: Black Magic, Metaphors, Mechanisms, Mini Worlds, and Maps, in Hanson and Riseman [1978].

Minsky, M. (1975) A Framework for Representing Knowledge, in The Psychology of Computer Vision, P. Winston (ed.), McGraw-Hill, New York.

Reddy, R. (1978) Pragmatic Aspects of Machine Vision, in Hanson and Riseman [1978a], pp. 89-98.

Rumelhart, D.E. & Ortony, A. (1976) The Representation of Knowledge in Memory, TR-55, CHIP, Dept. of Psychology, Univ. of Ca. at San Diego, La Jolla, Ca.

Sussman, G.J. & McDermott, D. (1972) Why Conniving Is Better Than Planning, AIM-255A, A.I. Lab, MIT, Cambridge, Mass.

Winograd, T. (1975) Frame Representations and the Procedural Declarative Controversy, in Representation and Understanding, D.G. Bobrow & A. Collins (eds)., Academic Press, New York, pp. 185-210.

MAPSEE2 COMPOSITION HIERARCHY

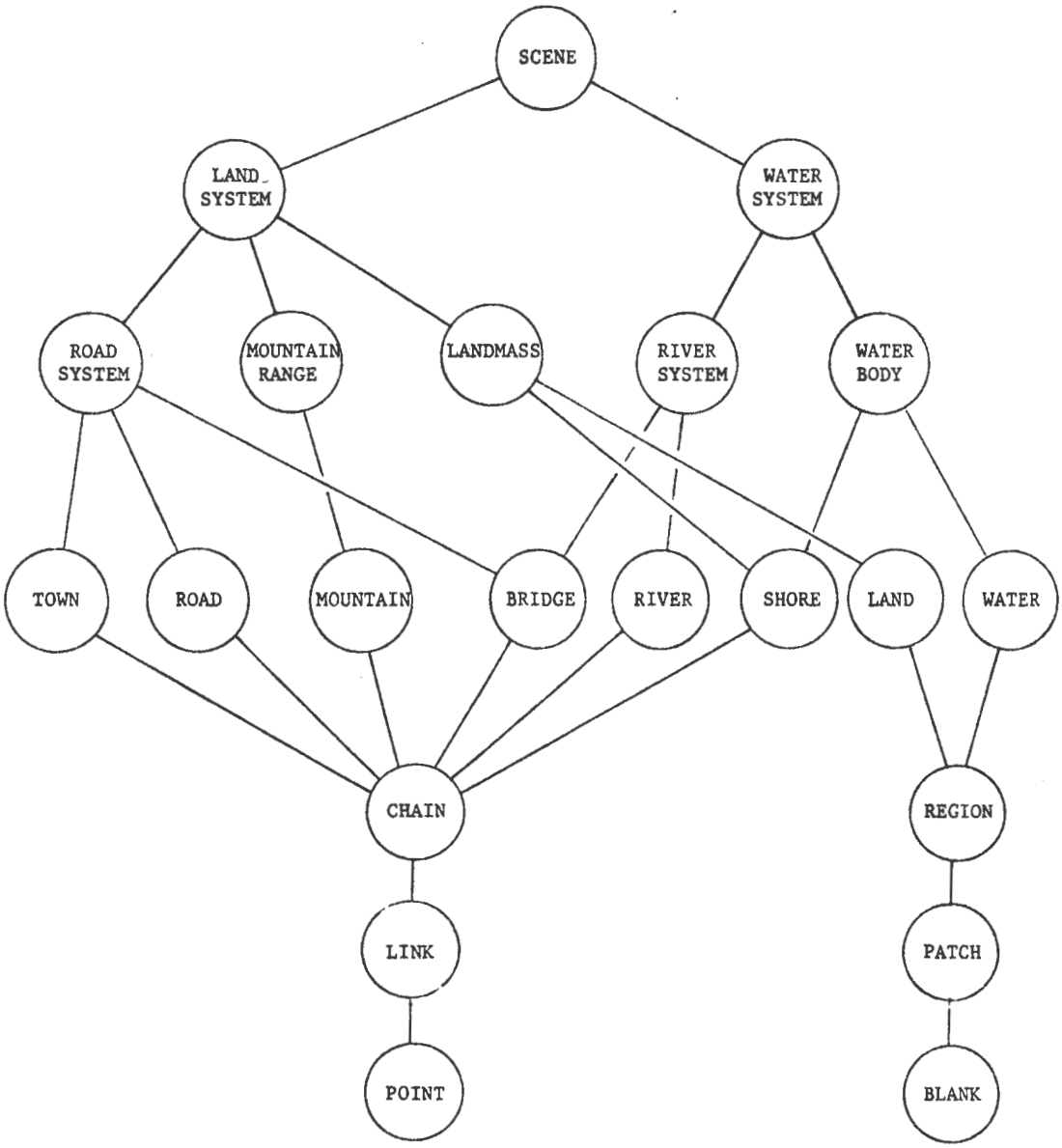



Figure 3

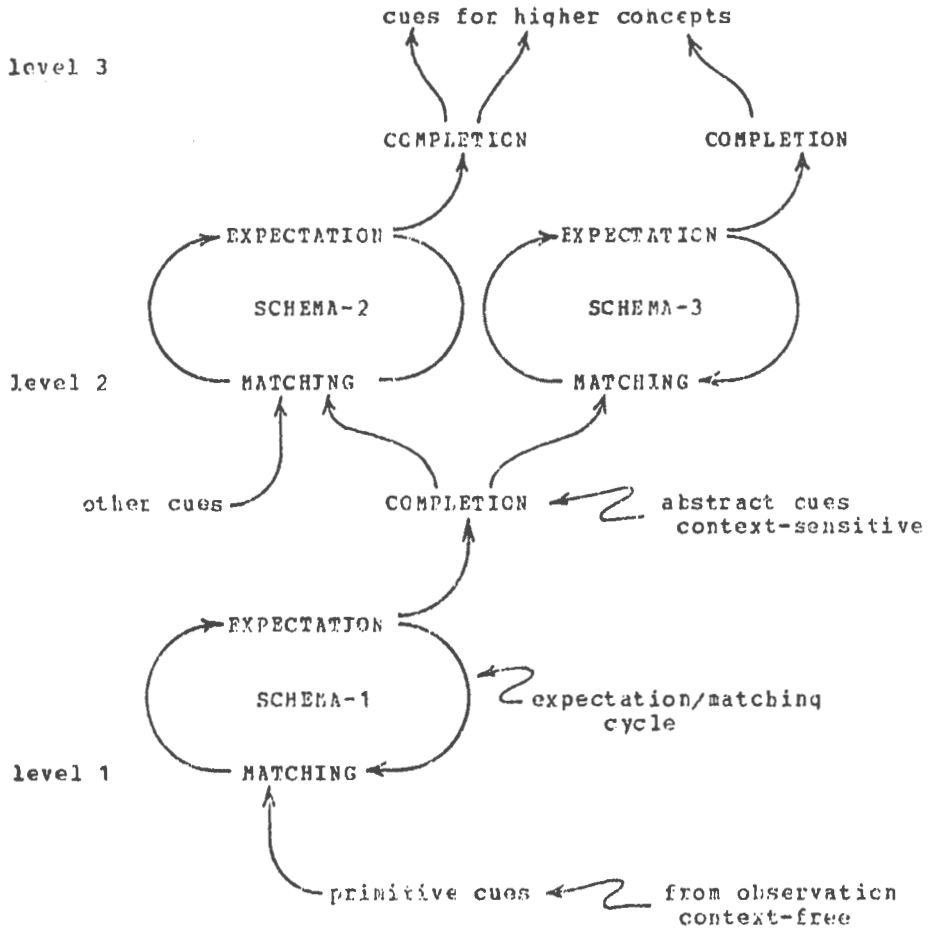

cues for higher concepts
level 3
COMPLETION
COMPLETION
EXPECTATION
SCHEMA-2
MATCHING
EXPECTATION
SCHEMA-3
MATCHING
level 2
other cues
COMPLETION
abstract cues
context-sensitive
EXPECTATION
SCHEMA-1
MATCHING
expectation/matching
cycle
level 1
primitive cues
from observation
context-free

Figure 4

# Push and Pop on Pictures: Generalizing the Augmented Transition Network Formalism to Capture the Structure and Meaning of Images

Heinz Breu and Alan K. Mackworth

Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada V6T 1W5

## Abstract

This paper presents a generalization of the Augmented Transition Network formalism to allow the writing of picture grammars. The generalized ATN is used to write a picture grammar for a subclass of Heraldic Shields. This application serves to illustrate such (usually linguistic) terms as "anomaly", "ambiguity" and "paraphrase" as applied to pictures. The paper also suggests two obstacles to progress in picture grammars and concludes that the proposed system overcomes them.

## 1. Introduction

### 1.1 Overview

The linguistic approach to image understanding pushed the analogy between sentences and images to the point of designing picture grammars as generalizations of string grammars (Miller and Shaw 1968, Fu 1974, Ledley 1964, Evans 1969). Such attempts were successful in highly circumscribed two-dimensional scene domains, but the approach has not made much further progress. Two reasons for the lack of progress are, first, the failure to allow for the expression and exploitation of graphical relationships and, second, the weak expressive power of conventional grammars as a programming language in which to write effective recognition procedures.

We propose a substantial generalization of the Augmented Transition Network formalism for string grammars (Woods, 1970) to allow the writing of picture grammars. This modification required the solution of the following problems:

- the ATN must look at suitable picture primitives rather than words
- the ATN must work with many graphical relations not just concatenation and composition
- the notion of "get next token" must be generalized for pictures
- the notion of "end of sring" must be extended to pictures

The solutions to these problems are presented in detail. They required the generalization both of the ATN formalism for grammars and its associated parser.

The modified ATN was used to write a picture grammar for a subclass of Heraldic Shields. This picture grammar takes, as input, an image of the shield represented as an array of pixels with colour values. It then outputs a linguistic description of the shield called an Heraldic Blazon.

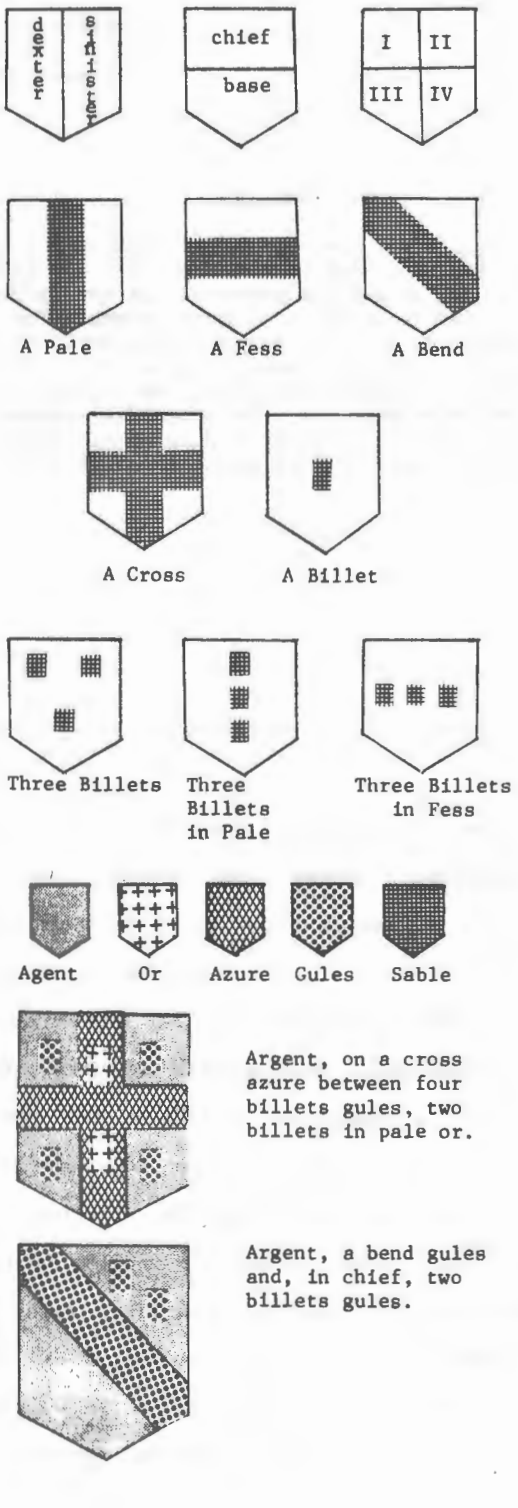
### 1.2 Heraldic Shields

We will restrict ourselves to the class of shields that can be described by the class of

blazons generated by the grammar of (Baker, 1977). The grammar for this class is presented below.

```
<shield> ::= <field>.
<field> ::= <colours>,<charges> |
            per pale: on the dexter <field>; and
                      on the sinister <field> |
            per fess: in chief <field>;
                      and in base <field> |
            quarterly: I <field>; II <field>;
                       III <field>; IV <field> |
            quarterly: I and IV <field>;
                       II <field>; III <field> |
            quarterly: I and IV <field>;
                       II <field> and III <field>
<colours> ::= <tincture> |
              <partition> <tincture> and
                          <tincture>
<tincture> ::= argent | or | azure |
               gules | sable
<partition> ::= per pale | per fess | quarterly
<charges> ::= <centred charge> |
              <minor charges> |
              on <centred charge>,
                 <minor charges> |
              <palewise ordinary> and,
                   <fesswise side>,
                   <minor charges> |
              <palewise ordinary> charged with
                   <minor charges> and,
                   <fesswise side>,
                   <minor charges> |
              <fesswise ordinary> and,
                   <palewise side>,
                   <minor charges> |
              <fesswise ordinary> charged with
                   <minor charges> and,
                   <palewise side>,
                   <minor charges> |
<centred charge> ::= <ordinary> |
                     <ordinary> between
                      <minor charges>
<ordinary> ::= <palewise ordinary> |
               <fesswise ordinary> | <cross>
<palewise ordinary> ::= a pale |
                        a pale <colours>
<fesswise ordinary> ::= a fess |
                        a fess <colours> |
                        a bend |
                        a bend <tincture>
<cross> ::= a cross | a cross <tincture>
<fesswise side> ::= on the dexter |
                    on the sinister
<palewise side> ::= in chief | in base
<minor charges> ::= a billet <tincture> |
                    <number> billets <direction>
                                     <tincture>
<number> ::= two | three | four
<direction> ::= in pale | in fess | <empty>
<empty> ::=
```

The terminology of heraldic blazons is bound to be unfamiliar to most readers. To aid in the reading of this paper, a description of these terms is provided in the following figure.



## 2. Generalizing the ATN

The ATN is generally regarded as an "off-the-shelf" tool for writing string grammars. The nature of the words in the dictionary and on the WRD arcs are left to the grammar writer. (The WRD arc is an extension to Wood's original proposal that requires the presence of a particular word next in the string.) Similarly, the nature of the function MORPH is also left to the user. (MORPH is the morpher that, given a derived word not in the dictionary, finds its root form in the dictionary and adds the required inflectional features.)

Our picture grammar ATN is to be treated this way also. We found however, that discussion of the ATN concepts is facilitated by the use of an example application. With this end in mind, let us first examine the heraldic shields domain.

### 2.1 Domain Specific Concepts

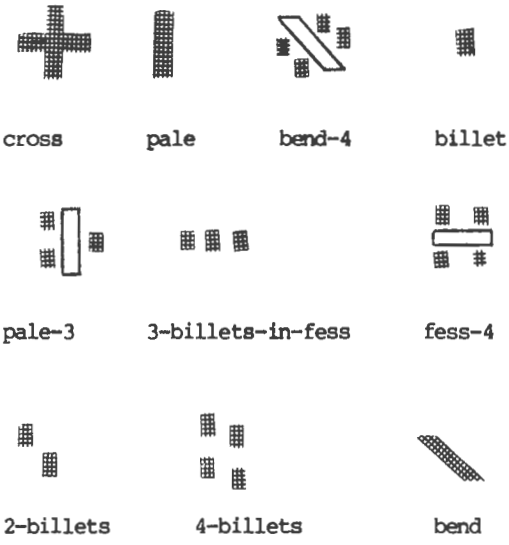
We would like to solve the following problems:

- define the picture primitives
- identify the picture primitives
- choose the relations needed to describe picture structure
- write a grammar and define the associated semantics

These problems will now be examined in more detail.

Primitives.

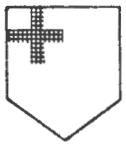
A subset of the primitives defined is shown below. The names underneath are the "words" on the WRD arcs of the grammar.




There are 24 primitives in all. The unshaded objects above are for illustration only and are not part of the primitives.

Each primitive is qualified by its coords which uniquely determines a primitive's size, shape, and coordinates. For example, (BEND COORDS) where COORDS = (Q-I SHIELDS), means only the primitive:

Note that "coords" is used in a slightly non-standard sense. This word here refers to the region in which the primitive is enclosed. For example, (CROSS (Q-I SHIELD)) is the shaded region below.



A billet in the middle of this region is indicated by (BILLET COORDS).

Identification

These primitives are detected by the WRD arcs of the grammar. The WRD arc requires the COORDS as an argument. A simple pattern matcher checks the occurrence of the primitive and returns its colour. For example, suppose the word is BEND and the coords are (Q-I SHIELD). The following pattern would then be generated:

Note that only the boundary of the primitive is generated as the pattern. This is to allow charged charges (e.g. a billet on a fess). The inside of the pattern is an "I don't care" region. The boundary must be a uniform colour (which is returned), and the pixels adjacent to it on the outside must be a different colour.

A possible matcher is shown below

```
(DEFUN MATCH (OBJECT COORDS)
  (COND ((APPLY1 '(LAMBDA (PATTERN)
                    (DIFFERENT (RIM PATTERN)
                               PATTERN))
                  (GENERATE OBJECT COORDS))
         (ERASE OBJECT COORDS))
        (T NIL)))
```

where

GENERATE returns a list of x,y coordinates making up the pattern i.e. $((X_1\ Y_1)\ (X_2\ Y_2)\ \ldots\ (X_n\ Y_n))$

RIM returns a list of x,y coordinates making up the rim of the pattern (i.e. the outside boundary)

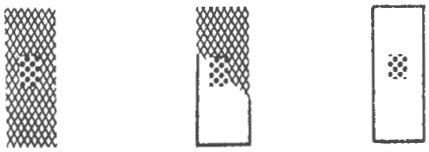
DIFFERENT returns T if PATTERN is a unique colour in the picture and (RIM PATTERN) consists of any colours different from those of PATTERN

ERASE SETQ2's every point of the object to a deletion figure (the object is all points within PATTERN of PATTERN's colour) and returns the colour of the object

MATCH therefore returns the colour of the object or NIL if the conditions are not met

PATTERN is a list of x,y coords

Upon recognition, the object is deleted. This is best done with a "bug" (or several, if need be) placed on the object, which "eats" away the coloured object. For example,



Note that the differently coloured billet in the centre is left untouched.

In our system, this deletion should be done using SETQ2. This is an undoable function in LISP/MTS and is chosen here to allow backtracking in the grammar. This notion of deletion corresponds to that of string advance in the regular ATN parser.

Relations

The relation among words in string grammars is that of concatenation. Another way of thinking of a sentence is as a series of slots into which the words fit. Along these lines, we consider a picture as a collection of regions into which the prmitives fit. These regions are the ones used in describing the shields with blazons. Namely,

| | |
|---|---|
| DEXTER | Q-I |
| SINISTER | Q-II |
| CHIEF | Q-III |
| BASE | Q-IV |
| CROSS | FESS |
| PALE | BEND |

Colours, as well, are considered to be relations. This holds not so much for primitives, as for entire regions. This is tested by looking at several points in the region where charges could not be. All the points must be the colour being tested for.

## 2.2 Generalization of ATN Concepts

The ATN parser was designed for string grammars, consequently some modifications were necessary to enable use with picture grammars.

String Advance

The area of interest is restricted on PUSH arcs by sending (via SENDR) the desired COORDS to the net of the subpicture being PUSHed for. For example, suppose we were looking for a cross in quadrant I of the shield and that we were presently at the top level. We would

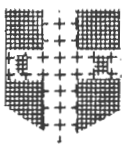
(SENDR COORDS (Q-I (GETR COORDS)))

to the CENTRED-CHARGE/ net. Here COORDS is the register containing the current region and Q-I is a function which returns the region which is the first quadrant of the region defined by its argument.

The notion of "advancing the input picture" is realized by the following actions:

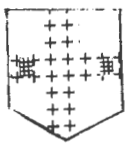
(1) Upon recognition of a primitive, that primitive is deleted from the picture (replaced by deletion characters). This occurs on WRD arcs.

(2) Upon detection of the colour of a region, that colour is deleted for that region. This occurs on TST arcs. For example,



detection of

(SABLE SHIELD)



Note that the two billets in fess sable are not deleted due to the nature of the bug method of deletion.

End of String

The notion of, "a string grammar parse is successful when the entire string is accepted", is generalized to that of, "a picture grammar parse is successful when the entire picture is accepted". This condition is detected in our system by a function (EMPTY COORDS) which tests if all the objects within the region specified by COORDS have been deleted. "End of Picture" is determined by calling (EMPTY SHIELD/).

WRD arcs

These are now used to detect picture primitives, rather than words. This arc now takes COORDS as an argument. This is necessary to completely specify the primitive as explained earlier in the section on primitives. An internal variable, PROPERTY, is set to (MATCH OBJECT COORDS). This could be anything the user desires, in this case, the colour of the object. Following the conventions of (Reiter, 1978), the WRD arc definition appears below:

(WRD <word> [coords] [test] [action]*)

If "test" evaluates to non-NIL, and if the primitive "word" is found at "coords" by MATCH (user defined function), perform the sequence of actions. The last action must be (TO <state>). * is bound to the tuple (word PROPERTY), where PROPERTY is returned by (MATCH word coords). WRD "advances the input picture".

MEM arcs

As for WRD, only MEM allows a list of primitives (in only one region, COORDS) all of which are checked. The format for MEM arcs is:

(MEM <words> [coords] [test] [action]*)

where <words> is a list of possible

alternatives.

TST arcs

These are used to test relations. In the SHIELDS example, they are used to test for the colours of regions. As such, TST still "advances the input picture".

PARSE

Since no sentence is being parsed, the function PARSE need not accept a sentence as an argument. The new format for calling PARSE is:

(PARSE <state>)

All string advance calls have been eliminated from the parser, as have the CAT arc and the calls to MORPH.

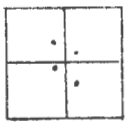
## 3. The Heraldic Shields Grammar

### 3.1 Semantic Tests

A few words need to be said about the semantic tests used. At present, these tests are only simulated but their designs are presented in the following discussion.

Q-TEST
PALE-TEST
FESS-TEST

These tests return T if the region specified by the argument is divided quarterly, vertically (per pale), or horizontally (per fess) respectively. These tests would work as follows:



Tests are made at the points (.) indicated above. These points are chosen so as to avoid charges. The numerals in the tests below are to be interpreted as

<NUMERAL> = colour of <NUMERAL> quadrant

Q-TEST (I ≠ II) AND (II ≠ IV) AND
(I ≠ III) AND (III ≠ IV)
PALE-TEST (I = III) OR (II = IV)
FESS-TEST (I = II) OR (III = IV)

Q-COLOURED
P-COLOURED
F-COLOURED

These are tests to determine if a given region is coloured by quarterly, palewise or fesswise division respectively. The tests themselves are the same as those above, but the tests are made on all the unshaded points below:

Points of one quadrant must be of uniform colour.

### 3.2 EMPTY

This test determines if the region specified by its argument has had all objects deleted, which is the case if the entire region is made up of deletion characters. It is used on POP arcs to ensure that a given region has been correctly parsed before going on to the next region. For example, we would like to know that quadrant I has been correctly parsed before attempting to process quadrant II. It is also used on some PUSH arcs to prevent looking for something in an empty region.

Note also the use of SENDR in restriction of

the area of interest by narrowing down of the COORDS.

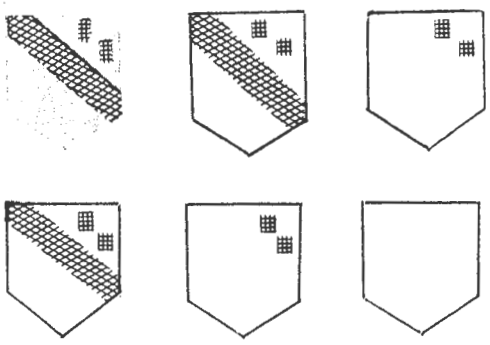
### 3.3 Successful Parses

A simulation of the proposed system has been implemented. This includes:

a) a modified ATN parser
b) the grammar
c) simulated semantic tests

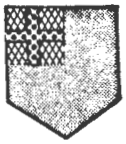
That is, the entire system has been implemented on the AMDAHL/470 at UBC, except the semantic tests, which are not automatic, but require interaction with a human operator. This implementation correctly translates shields into blazons for those shields described by Baker's heraldic blazon grammar. For example, when given the two example shields of Figure 1, the program yielded exactly those blazons corresponding to the shields in Figure 1. Shields not describable by that grammar are said to be anomalies. Anomalies are correctly rejected by our system.

The figures below show how the picture is processed.



### 3.4 Ambiguities

An interesting point that arose in this work is the existence of ambiguous shields. A shield is said to be ambiguous if it can be described by more than one blazon. An example of such a shield along with two possible blazons is shown below.



(1) Per pale: on the dexter per fess: in chief azure, a cross gules; and in base argent; and on the sinister argent

(2) Per fess: in chief per pale: on the dexter azure, a cross gules; and on the sinister argent; and in base argent.

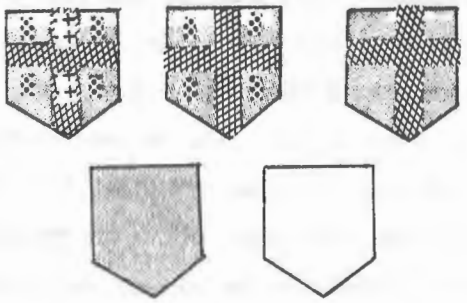
Note that the above shield could also be described as divided quarterly, but this would not be a valid parse in our system due to the semantic Q-TEST. The ambiguity is resolved in our system by arc ordering. The ATN is thought of as being a parallel process, but this is, of course, not the case in any sequential machine implementation. In our case, the arcs are tried in the order in which they appear. Hence, if we put the arc for PER PALE division first, then it will be tried first. This is equivalent to saying that if a shield can be divided both PER PALE and PER FESS, then divide it PER PALE. Blazon (1) is therefore the correct description as generated by our parser.

## 4. Directions for Further Work

Due to the nature of our pattern matcher, multi-coloured primitives are not permitted (e.g. a two tone bend). The matcher could be extended to allow such objects, which are permitted in Baker's grammar.

The present system's recognition of region

colours is unsatisfactory. It must avoid all charges that might be on the region in its detection and must be careful to delete around them. A re-ordering of nodes and arcs could have the system look first for charges, then for the colours of the region. Deletion would now mean extending the colours over the deleted charge. The semantic test EMPTY would require modification to consider a region of pure colour as "empty". As well, the grammar could no longer rely on knowledge of the region before checking for charges (e.g. at present, if a cross is detected, charges on the cross are looked for. The new design would find the charges on the cross first (and then realize that they are on a cross later, then look for the cross). An example of how the new system would process shields is diagrammed below.



Since our pattern matcher effectively generates each primitive, we could, by modifying this matcher into a generator, generate shields by running the grammar "in reverse". That is, we (or some program) could decide on precisely what arcs to take (i.e. the parse is pre-selected).

This, together with our notion of shield ambiguity, gives us a method of paraphrasing blazons. If we consider our arc ordering as being "standard", then our parse can be considered to be a "standard blazon". In particular then, we have a method for the standardization of non-standard blazons:

- use the non-standard blazon to generate a shield
- parse the shield
- the output will be the standardized blazon

## 5. Conclusions

This paper has shown that ATN's are modifiable to allow the writing of picture grammars. We have demonstrated this for a picture grammar for heraldic shields exhibiting context-sensitive properties, such as identity of quadrants.

We have found that the syntactic method (in our grammar) yields useful information (blazons) about the structure of heraldic shields. Anomalous shields are rejected and a notion of "standard blazon" eliminates ambiguity.

We feel that generation of shields is possible by running the grammar "in reverse", and that heraldic blazons may be paraphrased by considering shields to be the semantics of the blazon.

We see that relational richness is provided in our ATN by our use of "layered" regions. That is, we look for things on crosses, etc. An alternative way of thinking of this, perhaps more indicative of its power, is as "restricted" regions. That is, when we PUSH for (Q-I coords), we are actually focusing our attention on this restricted area of coords.

Finally, we conclude that our system has inherited the ease-of-programming associated with

string grammar ATNs.

## REFERENCES


Baker, J. Grammar for Heraldic Blazons. In UBC CPSC 215 Notes, 1977.

Evans, T.G. Descriptive pattern analysis techniques. In (Grasselli 1969), 79-96

Fu, K.S. Syntactic Methods in Pattern Recognition. Academic Press, New York, 1974.

Graselli, A.(Ed.) Automatic Interpretation and Classification of Images. NATO Advanced Study Institute, Pisa, 1968. Ledley, R.S. High-speed automatic analysis of biomedical pictures. Science 146, 9 (1964), 216-223

Miller, W.F. and Shaw, A.C. Linguistic methods in picture processing--a survey. Proc. AFIPS Fall (1968) Joint Computer Conference, 279-290.

Reiter, R. The Woods Augmented Transition Network parser. Technical Note 78-3, Department of CPSC, UBC, 1978.

Woods, W.A. Transition network grammars for natural language analysis. Comm. ACM 13, 10 (1970), 591-606

# Automatic registration of Landsat images using features selected from digital terrain models

James J. Little
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada V6T 1W5

## Abstract

Before two Landsat MSS images can be compared, they must be registered by being brought into correspondence with some reference datum. The reference can be one of the images, a synthetic image, a map, or other symbolic representation of the area imaged. A novel method is presented for determining the transformation to align an image to a digital terrain model, a structure which represents the topography of an area. Parameters of an affine transformation are computed from the correspondence between features of terrain extracted from the digital terrain model, and brightness discontinuities found in the Landsat image.

## 1. Introduction

A Landsat MSS image measures scene radiance in each of four spectral bands, at a nominal ground resolution of 80 x 80 meters. The position and attitude of the satellite is known with limited precision. After bulk processing, the estimated ground location of a pixel may differ from its true position by as much as 10 km. Further processing is required to make the coordinate systems of multiple images comparable. Registration can benefit from the availability of accurate digital terrain models. A digital terrain model (DTM) is accurately located in a geographic coordinate system. A Landsat image registered to a digital terrain model can be directly compared with other sources of geographic information, and other images.

Horn and Bachman (1978) used synthetic images generated from digital terrain models to register Landsat images. Their work assumes that the transformation between the synthetic image and the Landsat image can be described in terms of rotation, translation and scale change. A correlation of the real and synthetic images is used as measure of goodness of fit to guide the adjustment of the parameters of the transformation. Horn and Bachman's method is based upon an areal correlation which can be computationally expensive. The authors avoid some of this expense by first using low resolution images to produce rough estimates of the registration parameters. The full resolution of the data is used to compute the final corrections to these estimates.

Work by Horn and Woodham (1978) has shown that an affine transformation is sufficient to register small subsections of a Landsat image to a synthetic image, or, in our case, a DTM. In the technique presented here, the Landsat image and the DTM are each characterized by a set of curvilinear features. A correspondence between the elements of the two sets of features is established which satisfies both geometric (shape) constraints and topological (adjacency) constraints. The matching between elements

provides the input to a least-squares estimator for the parameters of the affine transform. To test the method, a 100x100 pixel subsection of a Landsat image (figure 1) is registered to a digital terrain model. The Landsat image was acquired on September 14,1976 (frame ID 11514-17153). The digital terrain model was digitized from a 1/50000 series contour map, NTS sheet 82 F/9, St. Mary Lake, centered on latitude 49 degrees, 37.5 minutes and longitude 114 degrees, 15 minutes. This area is southwest of Cranbrook, British Columbia.

Fig.1 Landsat image subsection (100 x 100)

## 2. Extracting features of terrain

The terrain representation used is the Triangulated Irregular Network (Peucker et al., 1978) which represents the terrain surface as a mesh of contiguous, non-overlapping triangular facets. The structure of terrain can be represented by the network of ridges and channels, or divides and streams. The ridges are convex linear surface features which, theoretically, connect passes (saddle points) to peaks (relative maxima). In practice we find that the set of ridges on a surface also includes convex linear features which connect to the main ridges that do join passes to peaks. Channels are concave linear features similarly defined. These elements of surface structure are explicitly represented in the model as the edges of triangular facets. A synthetic image generated from the test area DTM is shown in figure 2.

Fig.2 Synthetic image from the DTM

After determining the sun position corresponding to the Landsat image, it is possible to select those ridges which will appear in the image as linear brightness discontinuities. A simple model of surface reflectance is used (Horn and Bachman, 1978). The slope of each surface facet is derived and the brightness of the surface determined using the assumed reflectance function. Those ridges are selected which are bounded, on one side, by a

self-shadowed facet (one which receives no direct illumination), and, on the other side, by a facet whose predicted brightness is relatively high. The ridge sections are merged into curves when they are adjacent and are consistent in direction. Only those curves are output which represent, on average, a strong brightness discontinuity. Figure 3 shows the features extracted from the DTM.


Fig.3 Features extracted from the DTM

## 3. Extracting Features from the Landsat Image

In the Landsat image, these ridges will appear as boundaries where a transition occurs between a bright and dark region. Desirable boundaries are those formed by mountain ridges oriented perpendicular to the azimuthal direction of solar illumination. Shadow boundaries may also be found, but, since the direction of the incident illumination is known, they can be distinguished from the transition features formed by ridges. Shadows are dark on the side of the edge nearer the light source.

### 3.1 Filtering

The Landsat image is convolved with a 5x5 Sobel operator (Iannino and Shapiro, 1979), which is composed of two orthogonal components. The ratio of the outputs of these provides an estimate of the direction of the boundary element passing through the pixels tested. By checking this direction against the azimuthal angle of the sun position, it is possible to reject any brightness discontinuities caused by shadows.

The 5x5 filter gives high values not only at discontinuities, but also at pixels offset from the discontinuities. This produces secondary lines lying parallel to the original. In order to eliminate these as early as possible a scheme of Nevatia and Babu (1979) is used. An edge element is judged to exist at a pixel if

a) the magnitude of the filter output is above a threshold
b) its magnitude is higher than that of its two neighbors in the direction normal to the estimated edge direction, and
c) the edge directions of these neighboring pixels are within 45 degrees of the direction at the central pixel.

If any of these conditions do not hold then no edge element is present. The effect of this process is to suppress the 'echo' elements at an early stage, eliminating the need for later curve thinning procedures.

### 3.2 Line Growing

The output of the filter is used in the construction of the linear features, which

implements the method of Bajcsy and Tarakoli (1976). A histogram of the values of the filter output is derived. This histogram is used to direct the process so that lines are 'grown' from those points which had the highest output from the filtering step. A cumulative distribution function is derived from the histogram. At each step in the line growing process, the filter threshold is is relaxed so that five percent more pixels are above it. Initially the threshold is set at the 95 percent level.

At each stage in the line construction process, the threshold is set at the proper level and all points are tested in the order in which they are stored in the image. The threshold is lowered a level, and the process repeated, until the minimum level is reached. Lines are constructed incrementally in this first stage; at first a line consists of a single point. When another adjacent point lies above the threshold, and cannot be joined to any existing line, it is joined to the single point and forms a two-point line. To ensure that the lines found have more than a certain minimum curvature, points are added to an existing line only if they are adjacent to the endpoints of the line and the segment connecting the new point to the endpoint lies within 45 degrees of the direction of the nearest segment in the line.

The result of the first stage is a set of lines each consisting of a set of connected pixels. In the next stage, these lines are merged into larger connected lines when two conditions hold: first, the lines must be adjacent at their endpoints, and, second, they must each be compatible with the orientation of the line segment at the end of the other line. To aid in this process, a piecewise linear approximation is derived for each of the curves.

### 3.3 Approximations to Lines

A piecewise linear approximation to a digital line (Ramer,1972) approximates a line to a given precision by a set of linear segments connecting points on the line. In its construction, the first and last points in the line are connected by a straight line segment and those extreme points are found which lie farthest in perpendicular distance from the line segment (figure 4, A and B). These extreme points are included in the approximation if their distances from the segment are above the specified threshold. The line is then subdivided into the three sets of points to the left, between and right of the selected points. The three subsets of the line are processed recursively in a similar fashion. If the point farthest from the segment in a particular subset is within the threshold distance, then processing of that subset of the line is stopped, and only the endpoints of the line segment retained. The process of finding such an approximation is termed 'generalization'.

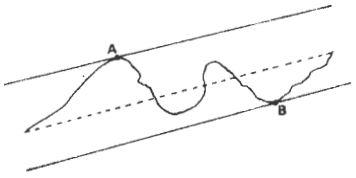



Fig.4 A curve and its extreme points

Most of the lines contain many colinear points, so the line approximation process reduces the number of points in the lines substantially. Using the approximations, directional decisions involving the orientation of line segments are less affected by any perturbations at the end of curves caused by quantization. The output of the feature-detector is the set of lines in generalized form, which are longer than a specified minimum length(figure 5).


Fig.5 Landsat features

## 4. Matching

The original scheme proposed matching the edges from the Landsat to those in the DTM by using the intersection of linear features when these formed vertices of degree more than two. Nodes of similar degree were to be paired, starting with those of highest degree. The orientation of the edges incident upon these nodes would be compared to determine the correctness of the matching. However, because it was difficult to find nodes of degree higher than two in the Landsat image, the design of the matching process had to be modified. The matching is instead driven by the structure of the curves themselves and the spatial relationships among them, rather than their intersections.

### 4.1 Previous Work

Our method is an extension of the techniques used for image registration developed at SRI (Bolles et al.,1979). In their method, the transformation from the test image to the reference is modelled as a function of the camera parameters, such as focal length, X,Y, Z, heading, pitch and roll. An essential part of the SRI method is that there is a good 'a priori' estimate of the camera parameters and of the errors in these parameters. These estimates are used to predict the extent of the region in the image which is to be searched for an element from the reference image. The predicted search region for an element is termed its 'uncertainty region'. Once an element is located within its search region, it is possible to reduce the search regions for other elements; the pairing of reference element and image element provides new information, which is used to improve the camera parameters and reduce the errors. Both linear and point features are hand-selected from the reference image for registration. The SRI system utilizes the notion of local support for the verification of linear features. For example, highways are composed of several parallel lanes; in detection of a highway the system searches for

locally offset lanes to confirm the matching of others. The matching of elements provides information for the correspondence refinement process which solves the nonlinear camera parameter estimation problem.

## 4.2 The Method

Our registration method proceeds in several stages, following a similar scheme. The input consists of the features found in the Landsat images, which we will term the "l-edges", and the ridges selected from the DTM. The elements of both feature sets are the result of 'generalizing' the appropriate curves using the same approximation technique and the same threshold. If a curve is represented in its generalized form by a straight line segment, then it is 'simple'. Any curve whose generalized representation requires interior points, other than its endpoints, is said to be 'structured'. During all stages of the registration process, features in both images are ordered by the amount of structure in their generalized representation.

The transformation applied to register features from Landsat to terrain model coordinates is an affine transform, an operator of 6 parameters. Finding the tranformation parameters requires pairing at least three points from each image. These are commonly supplied manually by selecting ground control points from both images. If more than three are supplied, a least-squares estimate of the transform can be computed. An 'a priori' estimate of the coefficients of the affine transform can be derived from the parameters of the satellite's orbit (Horn and Woodham, 1978). This transform least constrains the translation component, so the practical strategy is to determine the necessary translation. The nominal position of the Landsat data indicates the center of a search region for a terrain element, and the known errors determine an 'uncertainty region'.

The ridges are considered in the order of their structural complexity; it is assumed that the more strongly an element differs from a straight line the less likely it is to be incorrectly matched. The goal of the matching process is to pair a sufficient number of features from the Landsat image and the terrain model to compute the transform parameters correctly. Note, however, that we compute the transformation directly rather than estimate camera parameters.

Once three feature pairings have been established, the affine transform can be derived. Exhaustive examination of all such triples is clearly too expensive. Knowledge of the constraints imposed in the problem, especially in the image formation process, must limit the search space.

## 4.3 Construction of the Matching

The matching starts by selecting a ridge and ranking the l-edges in its search region by the strength of the match found. The comparison procedure determines a translation vector which will match the appropriate points of the ridge and the transformed l-edge. Each l-edge is

transformed according to the 'a priori' transform estimate, and compared with the candidate ridge. To assess a pairing of features, it proceeds as follows:

> a) If neither feature has structure, then the l-edge is tested to see if its endpoints lie within a band about the test ridge. If so, the measure of goodness of the match is the cosine of the angle between the two curves.
> b) If the ridge feature is structured, then the l-edge is compared with each element in the ridge at the second level, in a similar fashion.
> c) If both have structure, then the angles of the 'bends' in the curves are compared. If they are sufficiently similar, a matching is constructed which identifies the points at the 'bends'. Otherwise the best match of substructures of the two features is returned.

Any comparisons which yield a high value, near 1.0, are said to succeed.

### 4.4 Support for a Matching

A successful match specifies a translation vector. Each subsequent pairing must be consistent with the previous pairings, that is, the translation required to construct the pairing must be similar to those previous. However, experimentation with the feature sets has shown that this is not enough. To eliminate incorrect matchings, we must also use the local spatial structure of both the ridges and the l-edges to guide the matching.

When an initial pairing of features is made, nearby ridges are examined and a tally is kept of the number of nearby ridges which can be paired with l-edges in a matching consistent with that under construction. The pairing of ridge and l-edge is chosen which has the highest tally (i.e. which can be best locally extended). This strategy can be understood as a generalization of the scheme of determining local support for linear features employed in the SRI system.

The matching is extended to include three mutually consistent pairings of features. With the six values from the matching, an affine transform can be determined. Each pairing of a ridge and an l-edge provides a point-to-point match for the parameter determination. Detection of the termination of a boundary in the Landsat image is unreliable, so the endpoints of an edge are not entirely satisfactory choices for matching in all cases. However, when an l-edge is matched to one of the arms of a structured ridge, or vice versa, the appropriate endpoint can be directly matched to the bend point. In other cases the center of the segment (the average of its endpoints) is used.

This first estimate of the transform computed from the three pairings is tested for self-consistency by using the new transform to predict the overlap of the transformed l-edges and their matching ridges. If they overlap, the new transform is used to predict the the location of the remaining l-edges in the terrain model. The number of l-edges which overlap existing ridges is used as the measure of the quality of the matching. If the enough features can be matched in this way, the set of pairings is used to form an extended matching, from which a least-squares estimate of the affine transform parameters is computed. This transform, in turn, is used to predict the location of the l-edges in the DTM. If the matching grows, a new least-squares estimate of the transform is

computed. This iterative process terminates when the number of matched features does not increase.

## 5. Discussion and Conclusions

### 5.1 Results

In our test case, the initial search region for a ridge feature was set at 0.75 km, or approximately 10 pixels. The registration determined from the matching found by our system resulted in an average error of less than 10 meters, much less than half a pixel. The points matched are labelled A-K in figures 3 and 5. Further experiments will examine the capacity of the method to register images when the error in the original estimate is larger.

### 5.2 Future Work

Several improvements in the method are envisioned. The relationship among features is not used in the feature-to-feature matching. In particular, feature pairings should be constrained to ensure that colinear l-edges are matched to colinear ridges. Presently, line-to-line matching is based only upon line shape. The inclusion of other information, such as the shape of the intensity profile across a line, should improve matching

### 5.3 Comments and Conclusions

The present system extends the work of Bolles et al. in several areas. First, the features used for matching are automatically generated from the surface representation, consistent with the analysis of the image to be registered. Our method generalizes the notion of support for matches by using the local spatial structure of the features. Since the system is guided by the structure of elements, it can rapidly discover distinctive matches. The technique of searching for supporting evidence eliminates false matches readily. These aspects of the method recommend it as a registration technique, considering the volume of satellite images to be processed.

## REFERENCES


Bajcsy, R., and Tavakoli, M. Computer recognition of roads from satellite pictures. IJCPR2, 1976.

Bolles, R.C. et al., "Automatic Determination of Image-to-Database Correspondences", IJCAI-79.

Horn, B.K.P. and B.L. Bachman, "Using Synthetic Images to Register Real Images with Surface Models", Comm. ACM 21, 11 (Nov. 1978), 914-924.

Horn, B.K.P. and R.J. Woodham, "Landsat MSS Coordinate Transformations", Proceedings of the Fifth Annual Symposium on Machine Processing of Remotely Sensed Data, Purdue University, 1979.

Iannino, A. and S. Shapiro, "An Iterative Generalization of the Sobel Edge Detection Operator", Proceedings of the Pattern Recognition and Image Processing Conference, IEEE, August 1979, 130-137.

Nevatia, R. and K.R. Babu, "Linear Feature Extraction and Description", IJCAI79.

Ramer, U. "An iterative procedure for the polygonal approximation of planar curves". Computer Graphics and Image Processing 1,3 (1972).

Peucker, T.K., R.J. Fowler, J.J. Little, and D.M. Mark, "The Triangulated Irregular Network", Proc. of the Digital Terrain Models Symposium, May 9-11, 1978.

# Quantification and Characterization of the Shape of a Moving Cell

M.D. Levine and Y.M. Youssef

Electrical Engineering Department
McGill University
Montreal, Quebec
Canada

## Abstract

Cell movement is a fundamental process of some importance to embryological development and to host defence mechanisms. However there is no existing method for quantifying the observable changes in nucleus and membrane shape that occur during locomotion. This paper outlines an image interpretation system capable of analyzing the structural changes in the shape of a moving cell from a sequence of pictures. It is used for analyzing a cine film to detect, quantify, and symbolically describe the dynamic changes in the cell's nucleus and membrane. The system consists of two main analysis processes: static scene analysis, to provide a numeric and symbolic description of the static cell geometry and location; and dynamic motion analysis to quantify and analyze the dynamics of the cell motion and shape changes. The different computational processes of the system cooperate through a common relational database structure using two different memories, a Short Term Memory (STM), and a Long Term Memory (LTM). The processes interact through the STM using the information stored in the LTM, until a complete description of the dynamic cell motion and shape is obtained. This type of analysis may be extended to consider more than one cell interacting with each other.

## 1. Introduction

In the study of the effect of substances which modify cell locomotion at the cell membrane level, we are interested in quantifying the observable changes in membrane shape that occur in locomotion. Advances have been made recently in the characterization of locomotory paths taken by cells in vitro and how these are affected by various substances [49]. The internal mechanisms for cell locomotion are also reasonably well understood and the role of microtubles, microfilaments and contractile proteins is receiving much attention [48]. However progress has been much slower as to how the cell monitors these external substances so that internal mechanisms might be modified. This interaction between external factors and internal processes has to occur at or within the cell membrane; yet presently we have no means of quantifying the observable changes in membrane shape that occur in locomotion. Consequently, it is difficult to study at the membrane level the effects of substances which modify cell locomotion. To achieve this objective using the techniques of digital image processing, this paper outlines a computer vision system capable of analyzing the structural changes in the shape of a moving object from a sequence of pictures. The system must be able to recognize the various image patterns, segment and interpret the desired object, and detect significant changes in the location and shape of the object. Using such a system to analyze a cine film of a moving cell, quantification and symbolic description of the cell's geometry are provided and in this way we can characterize dynamic changes in the shape of the cell nucleus and membrane.

A review of the background and related areas of study to the developed system is given in section two. A general overview and its structure is presented in section three. Finally, in section four, we conclude with the present status of the project.

## 2. Background

The research reported here is related to three different areas of study: automatic image processing of cell images, tracking of moving objects, and shape analysis and description. This section is a brief review of the significant work that has been done in each of these areas.

The early history of automatic processing of cell images can be traced to the 1950's, and is directly related to development of the so-called television microscope [57]. Most of the work which has been done in this field has dealt with static pictures of blood smears for the purpose of classification or counting [3, 19, 20, 40, 58]. More recently, this has led to the development of experimental and also practical systems whose performance in many cases equals that of the human technologist. With regard to the analysis of cell movement, most of the effort has been concentrated on tracking cell paths, rather than studying cell interaction characteristics [6,10,15, 50, 16]. However, our earlier work[32,33,70] was rather different, because it addressed the problem of analyzing a group of live cells. This system was able to quantify the cell path and compute the steady-state probabilities, from which the ultimate direction of the cell population could be predicted.

Except for the work on cell mitosis reported in [13], there is no existing system which concerns itself with the analysis of the structural changes in the cell membrane which occur during locomotion, the main concern of our current work.

Extending our consideration to the general problem of processing dynamic scenes, either by motion detection or motion analysis [38], this field has been largely restricted to the detection of locomotion changes of an object rather than the dynamic alteration of its shape [2,9,29,30,36,37, 42]. In some recent work by Nagel [45], the problem of detecting the shape of a moving vehicle

was considered. However its motion was subject to many constraints, so that changes in its shape could be predicted given a knowledge of the prevailing situation. In our case, we are considering two kinds of changes with time, one in locomotion, the other in shape, and both can change randomly from frame to frame.

The third problem our research is related to is shape discrimination. This is a central issue to pattern recognition and as such has received attention in many papers dealing with recognition of characters, waveforms, chromosomes, cells, machine parts, etc. In a recent review by Pavlidis [51], he has classified the methodologies used in shape discrimination under two catogories; whether they examine only the boundary or the whole area, and whether they describe the original pictures in terms of scalar measurements or through structural description. Most studies of shape and pattern analysis are based on global feature measurements which then constitute a feature vector used for the shape representation. However more recently there has been great interest in syntactic pattern recognition techniques [53,55] which analyze patterns by a parsing process of hierarchial decomposition. The advantage of such an approach suggests that it might be appropriate to study hierarchial shape representation in more detail as a vehicle for cell shape description. We also note that, to data, numerical descriptors have been used for shape measures; however in order to provide a readable analysis for an interested physiologist, we are endeavoring to provide a symbolic description.

## 3. SYSTEM OVERVIEW

### 3.1 General Approach:

The basic requirement of the desired system is to be able to detect and analyze the structural changes in the shape of a moving object. To achieve this, we postulate two main analysis stages: static scene analysis to provide a numeric and symbolic description of the static cell geometry and location, and dynamic motion analysis to quantify and analyze the dynamics of the cell motion and shape changes. The different computational processes of the system cooperate through a common relational database structure, such as described in Shaheen and Levine [63]. using two different memories. The Short Term Memory (STM) is designed to work as a communication channel for all of the processes. It contains a record of the instantaneous cell motion and shape changes, as well as global description of the cell behaviour. The Long Term Memory (LTM), on the other hand, is also a relational database and contains the general model of the morphology and dynamics of the cell. The different processes interact through the STM using the stored information in the LTM, until a complete description of the dynamic cell motion and shape is obtained. Figure (1) shows a block diagram of the different computational processes of the system. This section will give a brief description of the data structure of the system and the objective of each process.

### 3.2 Static Scene Analysis

The main goal of this module is to process one frame in order to provide a numeric and symbolic description of the cell geometry and location in the current frame or any specific frame. Figure (2) shows a block diagram of the sequential processes in this module, a brief description of which follows:

Preprocessing (Initialization): Initates the input device such as disc, magnetic tape, 16 mm cine film, or a real time device (may be microscope connected to a TV camera interfaced with the computer viewing live blood cells).

Segmentation of the Cell: This task analyses the histogram to define the threshold(s) which segment(s) the complete cell(s) from the background (protoplasm). It also defines the coordinates of the cell boundary points (membrane) and labels the regions which belong to the different cell parts (nucleus, and cytoplasm.) The procedure is slightly different in the first frame from the others. In the first frame the desired cell is selected interactively, whereas following this it is tracked automatically.

Polygon Representation: Based on an approach introduced by Ramer [59], this process approximates the boundary points of the cell, as well as its parts, and represents them as connected polygons. This stage has the effect of grossly reducing the data which are manipulated by the higher level processes. The main factor here which controls the whole procedure and thus the result is the approximation level "threshold". This value must be selected very carefully in order to achieve a maximum decrease of data (minimum polygon vertices) while at the same time preserving all important information regarding the original shape. A similar approach to polygon representation was used by Liu [35] to classify the age of the neutrophil cell. He found that the represented polygon not only decreased the amount of data but also reduced the noise around the boundary points resulting from the digitization of the original object.

Polygon Decomposition: This technique is used to decompose the polygon making up the cell into simpler components. It was originally introduced by Feng and Pavlidis [12] to decompose a concave polygon into simpler convex ones for character and chromosome analysis. In our case this decomposition is not as simple. We are using the decomposition technique as a vehicle for recognizing and describing the different parts of the cell and for detecting and quantifying the dynamic changes in each of its parts. Connecting the centroids of these simpler components, we may simulate a type of medial axis transform which gives the simplest representation of the geometry of the cell shape.

Feature Extraction and Measurement: This task is executed in parallel with the previous processes to compute the necessary features such as: area, centroid, perimeter, orientation, elongation, circularity, etc. The features are selected in such a way as to minimize the number needed to give a complete description of the cell's geometry and location.

Shape Description: Based on the numeric feature measured, this step provides a global numeric and symbolic description of the morphology of the cell and its different parts (nucleus, cytoplasm, and membrane). First the numeric features are converted into symbolic ones using a "mapping table". Second, comparing the numeric and symbolic features with the LTM information about the general model of the cell, a description of the static cell's geometry and location can be given. For example, a description of the cell as "bended" or "segmented" can be computed from the decomposition of the cell. Other descriptions such as "small", "circular", or "elongated", can be computed from the features giving area, circularity, and elongation. The location of the cell within the frame can be described using the coordinates of the centroid and the two farthest points (the diagonal line), or other critical points on the boundary (membrane).

### 3.3 Dynamic Motion Analysis:

The objective of this module is to quantify and analyze the dynamics of the cell motion and shape changes. A brief description follows of the function of each computational process of this module shown in Fig. (1).

Incremental Location Change Detection: To define the changes in the cell location from frame to frame (cell tracking), the displacement of the cell's centroid between two frames is the most important factor. However it is possible that the centroid exhibits some displacement without any change in the cell location because all of the cell's elements are in continuous random motion. For this reason, in detecting the change in cell location, we must take into consideration besides the centroid displacement, tne changes in the coordinates of other critical points such as the centroid of the best fitted rectangle, the centroid of the maximum containing rectangle, and the farthest points around membrane in both length and width. This process also computes the speed and direction of the cell motion between any two frames.

Incremental Shape Change Detection: This computation is used to detect and quantify the change in cell geometry from frame to frame. The dynamic change in shape can be quantified by computing the incremental changes in the features that describe the cell geometry. Some of these changes may describe global change in the cell shape, for example: "The cell changed from bended to segmented (or vice versa)". Such a description can be derived from the change in the decomposition of the cell's polygon. Furthermore, the ultimate local change in the shape of any of the cell's parts can be described using the incremental changes in the features of the subpolygon making up this part, such as: The number of sides, length of each side, the angles between the sides, and the coordinates of each vertex. The selected features, symbolic terminology, and the type of description given by the system can be easily changed according to the application. Currently we are most interested in dynamic changes in the cell membrane, and are therefore giving more attention to features which describe the boundary (angles, sides, curvature, etc.) rather than those which describe the interior of the cell.

### 3.4 Global Motion and Shape Characterization:

This stage is concerned with characterizing the cell behaviour by analyzing the processed data of the motion and shape changes. After the analysis of each frame this module examines the details derived from each image as a coherent sequence, rather than as an individual increment. It compares the results of this analysis with the LTM data from which the dynamics characteristics of cell motion and shape can be given. For example, the motion of the cell can be characterized as: "From $t_1$ to $t_2$ the cell exhibited a constant velocity $V_{12}$ in direction $\theta_{12}$, and from $t_2$ to $t_3$ its motion was random. From $t_i$ to $t_j$ there was acceleration $A_{ij}$ in direction $\theta_{ij}$. Using the same methodology the dynamic changes in the shape can be characterized: "The cell started at $t_0$ with a small area and circular shape. At $t_1$ it showed slight elongation towards the north-east. At $t_i$ a pseudopod began growing at the lower left hand corner with a short joining base line".

These dynamic motion and shape characteristics are updated after processing each frame in order that at any given time the system can give the behaviour of the cell from the first ($t_1$) to the last ($t_n$) processed frame. However the module can be operated interactively by the user to investigate the cell behaviour through a specific period of time, or to examine the dynamic changes within a specific part of the cell in more detail.

## 4. CONCLUSIONS

This research deals with the quantification and characterization of the shape changes of a moving cell, a fundamental process of some importance to many aspects of cell biology. However there is no existing method to quantify the observable changes in nucleus and membrane shape that occur in locomotion. This paper outlines an image interpretation system capable of recognizing the various image patterns, segmenting and interpreting the desired object (cell), and detecting significant changes in the location of the object as well as in its shape. Using such a system to analyze the structural changes in the shape of the constituents of a moving cell, a quantification and symbolic description of the cell's geometry could be provided, thereby characterizing changes in the shape of the cell membrane. Investigation of the characteristics of the dynamic shape change and motion of the cell might provide clues to the nature and distribution of "receptors" on or within the membrane which might be a vital link in the interaction between external factors and cell internal processes.

To date, the static shape description, the incremental location change detector, and the incremental shape change detector have been largely completed. Attention is now being focussed on the last most important stage which is responsible for providing the global description of the

cell shape changes.

It is interesting to note that this technique employing a general purpose database in conjunction with general purpose analysis processes is also applicable to other similar problems. Examples are the visual monitoring of the behaviour of rats under the influence of various protocols, or the quantification and analysis of the changes of growing plants in different soils or under the effect of different fertilizers.

REFERENCES


[1] Alt, F.L., Digital Pattern Recognition by Moments, J. Assoc. Comput, Mach. 11, 1962, pp. 240-258.

[2] Arking, A.A., Lo, R.C., and Rosenfeld, A., An Evaluation of Fourier Transform Techniques for Cloud Motion Extimation", Computer Science Technical Report TR-351, University of Maryland, Jan. 1975.

[3] Bacus, J.W., An Automated Classification of the Peripheral Blood Leukocytes by Means of Digital Image Processing, Ph.D. Dissertation, University of Illinois, Chicago,1970.

[4] Bacus, J.W., A Whitening Transformation for Two Color Blood Cell Images", Pattern Recognition, Vol. 8, 1976, pp. 53-60.

[5] Bacus, J.W., and Gose, E.E., Leukocyte Pattern Recognition, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-2, No. 4, Sept. 1972.

[6] Barer, R., J. Opt. Soc. Am.47, 545, 1957.

[7] Blum, H., A Transformation for Extracting New Description of Shape", in Symposium on Models for the Perception of Speech and Visual Form, M.I.T. Press, 1964.

[8] Blum, H., Biological Shape and Visual Science", I.J. Theor. Biol. 1973, pp. 205-287.

[9] Bristor, C.L. Frankel, M. and Kendall, E., Some Advances in Automatic Determination of Cloud Motion and Growth From Digitized ATS-1 Picture Pairs, Manuscript submitted to Weather Motions from Space ATS-1, University of Wisconsin Press, Madison, 1970.

[10] Causley, D., and Young, J.Z., in Research 8, 1953, pp. 430-434.

[11] Chow, W.K., and Aggarwal, J.K., Computer Analysis of Planar Curvilinear Moving Images, IEEE Trans. on Computers, Vol. C-26, Feb. 1977, pp. 179-185.

[12] Feng, H.Y., and Pavlidis, T., Decomposition of Polygons into Simpler Components: Feature Extraction for Syntactic Pattern Recognition, IEEE Trans. Computers, C-24, 1975, pp. 636-650.

[13] Ferrie, F.P., Levine, M.D., Zucker, S.W., Cell Tracking: A Modelling and Minimization Approach, IJCPR, Miami, Florida, Dec. 1-4, 1980.

[14] Giuliano, V.E., Jones, P.E., Kimball, G.E., Meyer, R.F., and Stein, B.A., Automatic Pattern Recognition by a Gestalt Method, Inform. Cont. 4, 1961, pp. 332-345.

[15] Greaves, J.O.B., The Bug-System: The Software Structure for the Reduction of Quantized Video Data of Moving Organisms, Proceedings of the IEEE, Vol. 63, No. 10, October 1975.

[16] Greene, F.M., and Barnes, F.S., System for Automatically Tracking White Blood Cells, Rev. Sci. Instrum., Vol. 48, No.6, June 1977.

[17] Hannah, M.J., Generalized Automated Pattern Recognition: Pattern Classification by Moment Invariants, M.Sc. Thesis, Industrial Eng. Dept. University of Missouri, Columbia, June 1971.

[18] Hawksley, P.G.W., Blackett, J.H., Meyer, E.W., and Fitzsimmons, A.E., Brit. J. Appl. Phys. 5, pp. 165-173, 1954.

[19] Ingram, M., Nargren, P.E., and Preston, K. Jr., Automatic Differentiation of White Blood Cells, in Image Processing in Biological Sciences, ed. by D.M. Ramsey, Ed. Berkeley, Calif. Univ. Press, pp. 97-117.

[20] Ingram, M., and Preston, K. Jr., Automatic Analysis of Blood Cells, Sci. Amer. Vol. 223, pp. 72-82, 1970.

[21] Izzo, N.F., and Coles, W., Electronics 35, pp. 52-57, 1962.

[22] Kaufman, L., et al., Contour Description Properties of Visual Shape, Final Report on Contract AF19(628)-5830, Sperry Rand Research Center, Sndbury, Mass., Sept. 1967.

[23] Klinger, A., Kochman, A., and Alexandridis,N., Computer Analysis of Chromosome Patterns: Feature Encoding for Flexible Decision Making, IEEE Trans. Computers, C-20, 1971, pp. 1014-1022.

[24] Kolers, P.A., The Role of Shape and Geometry in Picture Recognition, Picture Processing and Psychopictorics (B.S. Lipkin and A. Rosenfeld, Eds.) pp. 181-202, Academic Press, New York, 1970.

[25] Langridge, D.J., On the Computation of Shape, Frontiers of Pattern Recognition (S. Wafanabe, Ed.), pp. 347-365, Academic Press, New York, 1972.

[26] Ledley, R.S., Analysis of Cells, IEEE Trans. on Computers, July 1972, pp. 740-753.

[27] Ledley, R.S., Rotolo, L.S. Golab, T.J., Jacobsen, J.D., Ginsberg, M.D., and Wilson, J.B., in Optical and Electro. Optical Inf. Proc., MIT Press, Cambridge, 1965, pp. 591-613.

[28] Ledley, R.S., High Speed Automatic Analysis of Biomedical Pictures, Science, 146, 1964, pp. 216-223.

[29] Leese, J.A., Novak, C.S., and Taylor, V.R., The Determination of Cloud Pattern Motion from Geosynchronous Satellite Image Data, Pattern Recognition Vol. 2, Dec. 1970, pp. 279-292.

[30] Leese, J.A., Novak, C.S., Clark, B.B., An

Automated Technique for Obtaining Cloud Motion from Geosynchronous Satellite Data Using Cross-correlation, J. Applied Meteorology, Vol. 10, Feb. 1971, pp. 118-132.

[31] Levine, M.D., and Youssef, Y.M., An Automatic Picture Processing Method for Tracking and Quantifying the Dynamics of Blood Cell Motion, Fourth International Congress of Cybernetics and Systems, Amsterdam, the Netherlands, Aug. 21-25, 1978.

[32] Levine, M.D., and Youssef, Y.M., A Real Time Laboratory Device for Tracking and Quantifying Blood Cell Movement, TR No. 78-2R, Jan. 1978, Elec. Eng. Dept. McGill University, Montreal.

[33] Lillestrand, R.L., Techniques for Change Detection, IEEE Trans. on Computers, Vol. C-21, July 1972, pp. 654-659.

[34] Limbo, J.O, and Murphy, J.A., Estimating the Velocity of Moving Images in Television Signals, Computer Graphics and Image Processing Vol. 4, 1975, pp. 311-327.

[35] Liu, Huei-Chi Richard, Shape Description and Characterization of Continuous Change, Ph.D. dissertation, State Univ. of N.Y. at Stony Brook, 1976.

[36] Lo, R.C., and Parkih, J.A., A Study of the Application of Fourier Transforms to Cloud Movement Estimation from Satellite Photographs, TR-242, University of Maryland, 1973.

[37] Lo, R.C., and Johr, J. Applications of Enhancement and Thresholding Techniques to Fourier Transform Cloud Motion Estimates, TR-326, University of Maryland, 1974.

[38] Martin, W.N., and Aggarwal, J.K., A Survey on Dynamic Scene Analysis, Dept. of Elec. Eng., The University of Texas at Austin, Dec. 1977.

[39] Mckee, J., and Aggarwal, J.K., Finding the Edges of the Surfaces of Three-Dimensional Curved Objects by Computer, Pattern Recognition, Vol. 7, 1975, pp. 25-52.

[40] Mendelsohn, M.L., Mayall, B.H., Prewitt, J.M.S., Bostrom, R.C., and Holcomb, W.G., Digital Transformation and Computer Analysis of Microscopy Images, in Advances in Optical and Electron Microscopy, V.E., Coslett, Ed. New York: Academic Press, 1968, pp. 77-150.

[41] Montanari, U., Continuous Skeletons from Digital Images, J. Assoc. Comput. Mach. 16, 1969, pp. 534-549.

[42] Moore, G.A., Application of Computers to Quantitative Analysis of Microstructures", National Bureau of Standard Report No. 9428, 1966.

[43] Mott-Smith, J.C., Medial Axis Transformation, in Picture Processing and Psychopictorics, B.S. Lipkin and A. Rosenfeld, Eds., pp. 267-283, Academic Press, New York, 1970.

[44] Mui, J.K., Bacus, J.W., and Fu, K.S., A Scene Segmentation Technique for Microscopic Cell Images, Proceedings of the Symposium on Computer Aided Diagnosis of Medical Images, No. 11, 1976.

[45] Nagel, H.H., Formation of an Ojbect Concept by Analysis of Systematic Time Variations in the Optically Perceptible Environment, Computer Graphics and Image Processing, Vol. 7, pp. 149-194, 1978.

[46] Nagy, G., Feature Extraction on Binary Patterns, IEEE Trans. Systems Sci. Cybernet, SSC-5, 1969, pp. 272-278.

[47] Nakimoto, Y., Nakato, K., Uchikura, Y., and Nakajima, A., Improvement of Chinese Character Recognition Using Projection Profiles, in Proceedings of the First International Joint Conference on Pattern Recognition (1973), pp. 172-178.

[48] Noble, P.B., Personal Communication.

[49] Noble, P.B., and Lewis, M.G., Lymphocyte Migration and Infiltration in Melanoma, Pigment Cell, Vol. 5, pp. 171-181, 1979.

[50] Parpart, A.K., Science 113, pp. 483-484, 1951.

[51] Pavlidis, T., Survey: A Review of Algorithms for Shape Analysis, Computer Graphics and Image Processing, Vol. 7, 1978, pp. 243-258.

[52] Pavlidis, T., Computer Recognition of Figures Through Decomposition, Inform. Contr. 14, 1968, pp. 536-537.

[53] Pavlidis, T., Analysis of Set Patterns, Pattern Recognition 1, 1968, pp. 165-178.

[54] Pavlidis, T., Representation of Figures by Labelled Graphs, Pattern Recognition 4, 1972, pp. 5-17.

[55] Pavlidis, T., Structural Pattern Recognition: Primitive and Juctaposition Relations, in Frontiers of Pattern Recognition, S. Watanabe, Ed., pp. 424-454, Academic Press, New York, 1972.

[56] Philbrick, O., A Study of Shape Recognition Using the Medial Axis Transformation, Report No. 288, Air Force Cambridge Research Laboratories, Nov. 1966.

[57] Preston, K., Digital Picture Analysis in Cytology in Digital Image Analysis, Azriel Rosenfeld, Ed., New York: Springer-Verlag, in Press.

[58] Prewitt, J.M.S., and Mendelsohn, M.L., A General Approach to Image Analysis by Parameter Extraction, in Proc. Computers in Radiology, Chicago, 1966.

[59] Urs Ramer, An Iterative Procedure for the Polygonal Approximation of Plan Curves, Computer Graphics and Image Processing, Vol. 1, 1972, pp. 244-256.

[60] Rosenfeld, A., and Kak, A.C., Digital Picture Processing, Academic Press, New York, 1976.

[61] Rosenfeld, A., and Weszka, J.S., Picture Recognition and Scene Analysis, Computer 9, 1976, pp. 28-38.

[62] Rutovitz, D., Centromere Finding: Some Shape Discriptors for Small Chromosome Outlines, Machine Intelligence 5, 1970, pp. 435-462.

[63] Shaheen, S.I., and Levine, M.D., Image Segmentation and Interpretation Using a Knowledge Data Base, Second National Conference of the Canadian Society for Computational Studies of Intelligence, Toronto, Ontario, 19-21, July, 1978.

[64] Spinard, R.J., Machine Recognition of Hand Printing, Infor. Contr. 8, 1965, pp. 124-142.

[65] Ulstad, M.S., An Algorithm for Estimating Small Scale Differences Between Two Digital Images, Pattern Recognition, Vol. 5, 1973, pp. 323-333.

[66] Wied, G.L., Lipkin, L.E., and Shapiro, N.H., Science 166, 180-204, 1968.

[67] Wong, E., and Steppe, J.A., Invariant Recognition of Geometric Shapes, in Methologies of Pattern Recognition, S. Watanabe, Ed., pp. 535-546, Academic Press, New York, 1969.

[68] Young, J.Z., and Roberts, F., Nature 167, 231 (1952) Nature, 169, 963 (1952).

[69] Young, I.T., Automated Leukocyte Recognition, Ph.D. dissertation, M.I.T., Cambridge, 1969.

[70] Youssef, Y.M., An Automated Picture Processing Method for Tracking and Quantifying the Dynamics of Blood Cell Movement, Master Thesis, Elec. Eng. Dept., McGill University, June 1977.

[71] Zahn, C.T., and Roskies, R.Z., Fourier Descriptors for Plane Closes Curves, IEEE Trans. Computers, C-21, 1972, pp. 269-281.

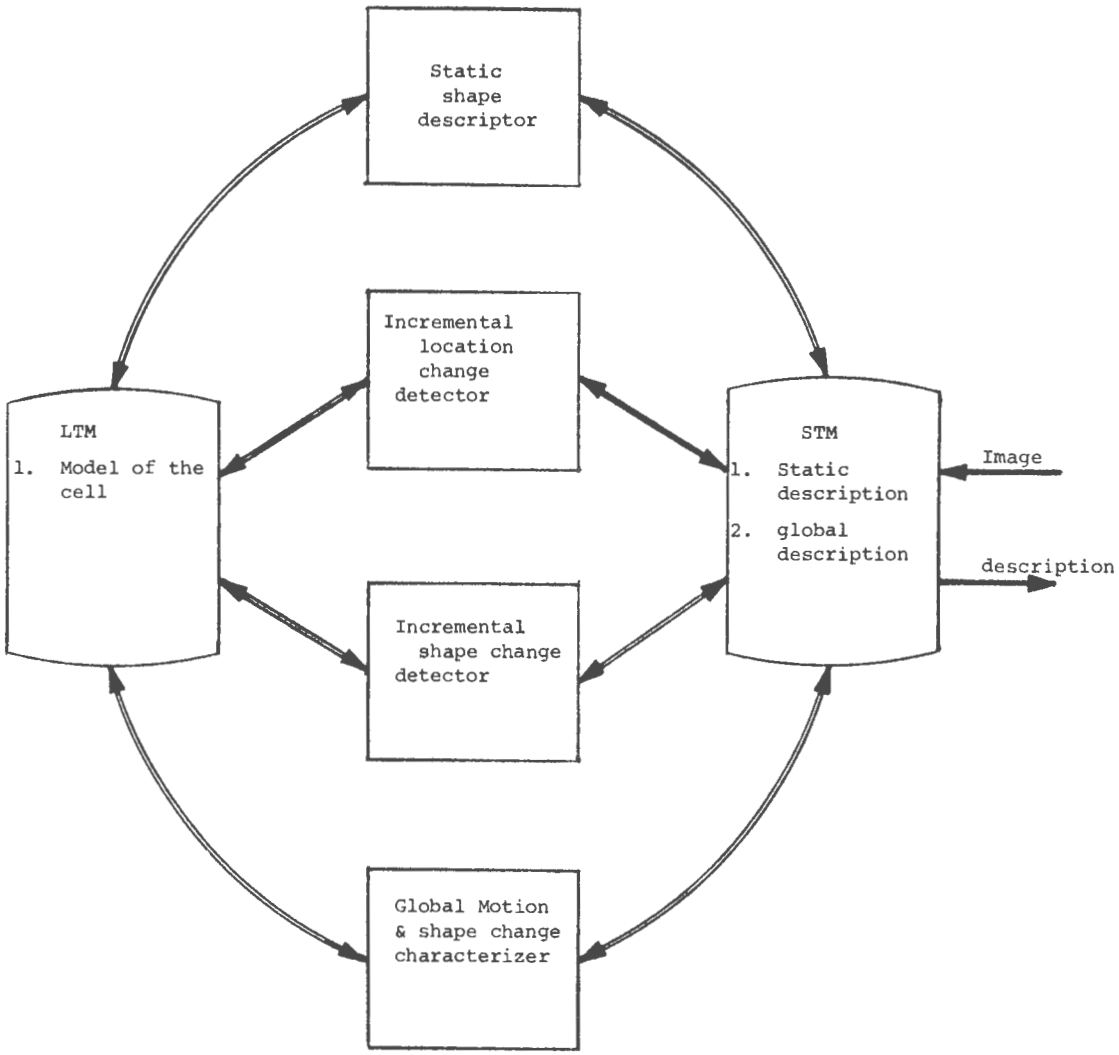



Figure 1: The main modules of the processing system for quantifying and characterizing the change in cell morphology.

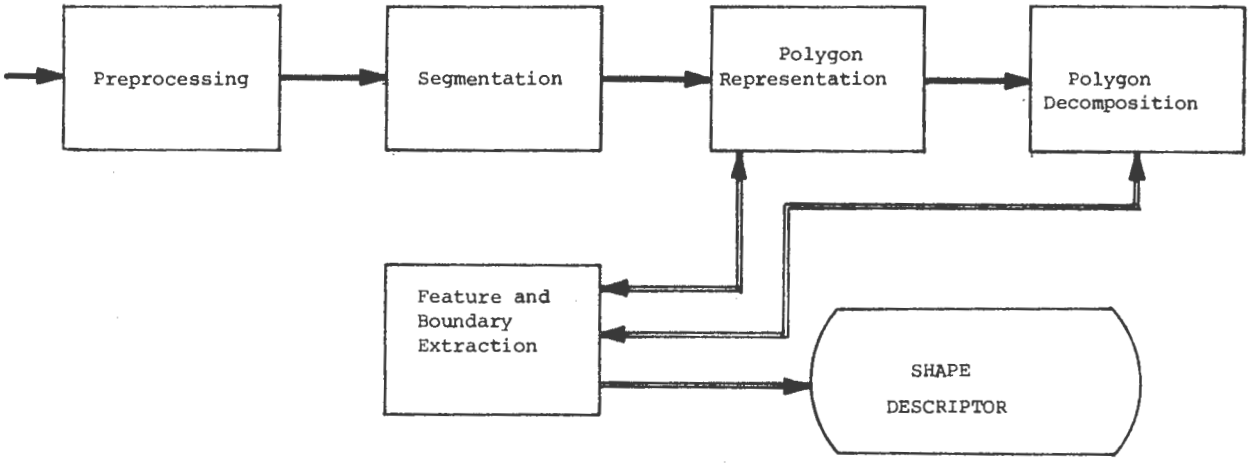



Figure 2: Block diagram of the processing steps for a particular cine frame.

# SPATIAL EXPERIENCE AND SPATIAL PROBLEMS IN A SIMULATED ROBOT-ENVIRONMENT SYSTEM

P.F.Rowat and R.S.Rosenberg
Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1W5.

Abstract

An overview of a simulated robot-environment system and the design of a robot-controller are presented. The environment is a flat tabletop with fixed and movable objects on it and the robot is a moving point that can pickup, move and drop an object. The robot uses an eye with varying resolution to view the tabletop and construct a world model of the environment. A novel approach to path-finding problems that uses the skeleton of the empty space is outlined.

## 1 Aims and motivation

This work was animated by a desire to understand the connection between perception and action. Every day we do such simple things as

- avoiding all obstacles in crossing a cluttered room
- navigating through an unfamiliar house
- making and executing a mental plan to go to the local shop or cross a campus
- moving an awkward piece of furniture around a house.

In order to explore the abilities required to exhibit such skills we have proceeded as follows:

1. To design and implement a simulated robot world which reflects to a certain extent the spatial aspects of a cluttered room or the floorplan of a house,
2. To specify a class of tasks of a spatial nature which the robot might reasonably be expected to solve in this world, and
3. To design computational processes which enable the robot to handle these tasks in a reasonably intelligent manner.

The simulated robot world is carefully designed to enforce a non-trivial treatment of the interaction between perception and action. The robot's sensory input from distant parts of the environment is either non-existent or very inexact and fuzzy, in accord with real world organisms; yet plans have to be made and actions executed.

## 2 The action cycle

The information-processing component of any organism that physically interacts with the outside world must consist of three distinct parts: sensory receptors, action effectors, and an intermediary that relates the senses and the actions. Our main interest is in a sufficient design for the intermediary, which will be referred to as the robot-controller.

Its major task, in order to improve the organism's survival chances, is to

build a world model: a model of the outside world. In information-processing terms, a world model is a data base of facts which, together with interpretive procedures, enables the prediction of future sensory input. Equivalently, it is a data structure and procedures for making predictions about the outside world. The purpose of a world model is to allow the construction of plans and thus to better achieve the organism's goals. A world model must be built to explain the sensory input received so far, using sensory inputs as the primitive items of evidence. Thus the world model of an organism is a function of the design of its receptors, and furthermore can never be assumed to be correct.

The interface between an organism and the outside world is defined by the organism's sensory receptors and action effectors, and is necessarily always sloppy.

The discussion so far is summarized in figure 1. Our robot-controller functions, at the top level, by the perpetual repetition of the action cycle, a loop containing three parts: perception, planning, and action. In our robot-controller these three processes are performed in serial order, whereas in most living organisms they are presumably performed in parallel.

## 3 System overview

The system consists of three main programs: TABLETOP, that simulates the outside world; UTAK, that simulates the robot; and PPA, the robot-controlling program.

### 3.1 TABLETOP, the environment simulator

TABLETOP simulates a frictionless tabletop with a polygonal restraining boundary or verge. There may be arbitrary polygonal shapes on the tabletop, some fixed and some movable. These shapes constitute the objects of the outside world. There are never any holes in an object. On this simulated tabletop the everyday laws of physics hold: the shape of an object remains invariant during motion, and if the path of a moving object is obstructed by another object or the verge then the moving object comes to an immediate standstill with a small gap between it and the obstruction. Figure 2 shows a sample world.

### 3.2 UTAK, the robot simulator

UTAK simulates the robot, (referred to as Utak), who is represented as a dimensionless point and is free to move wherever there is empty space. He cannot pass between two objects in contact. He can grasp an adjacent movable object, and can translate, rotate, or release such a object.

Utak senses his environment with an eye having a limited field of view and

having a variable resolution: fine in the centre or fovea, progressively coarser towards the periphery. The eye may be thought of as a TV camera, suspended at the top of a stalk sticking vertically up from Utak, with the camera pointing directly downwards at the tabletop and its field of view centered on Utak. Thus the eye gets a two-dimensional view of part of the tabletop and an image of Utak always appears at the centre of the field of view.

The retinal geometry of the eye is shown in figure 3(a). Each little square constitutes a retinal field, and covers a certain area of the task environment depending upon Utak's position. Corresponding to each retinal field there is a retinal cell, which registers a graylevel, or integer in the range 0 - 7, that depends on the ratio of object to total area in the part of the task environment covered by the retinal field. A retinal impression is the structured set of graylevels registered by all the retinal cells at one particular instant in time. Figure 3(b) shows an example of a retinal impression. Utak also has eight "tactile" receptors, one in each of the eight basic compass directions, which allow him to sense the colour of an immediately adjacent object. A tactile impression is the structured set of eight colors registered by the tactile receptors at one particular instant of time.

In sum, Utak inhabits an outside world which may be likened to a tabletop with confining verges, where he can wander around and move objects, and where his sensory contact with this world consists of a series of retinal and tactile impressions. The task for PPA, the robot-controlling program, is to make sense of the sensory input and create a world model for planning purposes.

Utak is given path finding ("Go to the north-east corner") and object moving tasks ("Push the square movable object into the next room"). The statement of a task may require considerable changes to Utak's world model. Consider, for instance, the object-moving task just mentioned. If Utak has so far seen a square movable object but has only explored what he thinks is one end of a single room, his world model before the task statement will simply consist of a room with one square movable object in it; but after "understanding" the task statement his world model will include an extra room with a doorway which connects it to the room he's currently in at a position consistent with his accumulated sensory experience to date.

### 3.3 PPA, the robot-controller

PPA, the robot-controller, is divided into three parts: ACCOM, SPLAN, and ACT. PPA operates by continually cycling

through these three parts corresponding to perception, planning, and action. ACCOM accepts a retinal impression and modifies (accommodates) the current world model in the light of this new evidence; SPLAN is the spatial planner and is responsible for always maintaining a valid plan to achieve the current task by creating a new plan or by updating an old one, while ACT simply computes from the current plan the next action to be executed.

A world model, a task, and a plan are defined at all times in PPA, whatever Utak's actual situation, including the moment before Utak "opens his eye" and receives his first retinal impression. So far, the following defaults have been used. The world model is taken to be a large empty square centred on Utak's initial position. The default task is to explore the assumed world, which means "collect evidence (i.e. sensory input) to confirm the current world model". If the default task results in the specific task of, say, "go to the north-east corner", then the current plan would consist of walk actions to the hypothesized position of the north-east corner. Other possible defaults are "sleep" or "find food".

The ACCOM program, responsible for understanding incoming sensory impressions, divides into two parts, ACC-INIT and ACC-SUB. ACC-INIT accomodates the initial default world model to the very first retinal impression while ACC-SUB carries out all subsequent accomodations of the world model to incoming sensory impressions.

The spatial planner SPLAN depends on a subsystem called SHAPE to solve path-finding and object-moving problems. SHAPE makes extensive use of the world model. The basic world model is maintained in a format of points and lines specified by means of Cartesian coordinates and will sometimes be referred to as the Cartesian world model or the Cartesian representation. SHAPE functions by projecting and re-projecting all or part of the Cartesian world model onto a digital array (the screen). Path-finding and object-moving problems are solved in simple cases from one projection on the screen; more interesting cases require several projections. A projection of the Cartesian representation onto this digital array will sometimes be referred to as an image representation.

The most important part of SHAPE is the collection of algorithms for solving path-finding and object-moving problems. These are based upon the concept of the skeleton of a two dimensional shape[Blum,1967]. The skeleton was found to be a useful tool for path-finding problems, and would be a useful heuristic for object-moving problems provided algorithms could be devised to compute it.

The skeleton of the empty space in a TABLETOP world is shown in figure 4. It turned out, for reasons given in [Rowat, 1979, 1980] that the published skeleton algorithms were unsatisfactory. However, it was possible to modify one of these to provide a satisfactory skeleton algorithm.

The skeleton of a planar shape may be described as follows. Imagine the shape to be the boundary of a dry grass prairie and imagine a fire set at all points of the boundary simultaneously. The fire advances as a wavefront with unit velocity into the interior of the shape. At certain points two or more sections of the wavefront emanating from distinct points of the boundary meet and mutually extinguish themselves. The locus of these points of extinction form a set of connected lines called the skeletal graph, while the time from the setting of the fire to the time of extinction at a point of the skeletal graph is known as the quench function. The value of the quench function gives the radius of the maximal circle that fits inside the shape at that point.

The third component of PPA is ACT, the program that computes the next action for Utak from a completed plan. This is not entirely trivial because the nature of the next action has to be a function of the confidence Utak has in the details of the world model in the vicinity of his current position and of the accuracy with which he can execute an action.

### 3.4 The parts of the organism-controller

Any complete organism-controller for Utak requires the following program steps. These can be stated here without specifying data structures or processes. All that is needed is a world model, a way to receive a retinal impression, and an action effector.

INITIALIZATION STEPS

1. Set the current world model equal to some default world model.
2. Receive the first retinal impression.
3. Analyze the retinal impression into regions and borders.
4. Interpret the regions in the retinal impression and identify the image of Utak in the retinal impression.
5. Modify the default world model to be consistent with the interpreted retinal impression.
6. Accept a task and interpret it in terms of the world model. This may require substantial modification of the world model, for instance the addition of an object if one is mentioned in the task but no object is "visible" in the current retinal impression.

PLAN

7. Construct a plan to achieve the task, using the spatial planner.

THE ACTION CYCLE

ACT

8. Test whether the task is complete. If so, STOP.
9. Decide on the next action to take, by examining the initial portions of the plan and the degrees of confidence associated with those parts of the world model close to the planned actions.
10. Execute the next action and receive the next retinal impression.

PERCEIVE

11. Interpret the new retinal impression on the basis of the current world model, and modify the world model as necessary to make it consistent with the current retinal impression.

PLAN

12. Is the plan still viable? If so

go to 8.

13. Otherwise, re-compute all or part of the plan, as in step 7, and go to 8.

The parts of the action cycle correlate with figure 5 as follows. Steps 8, 9, & 10 are carried out by ACT; Step 11, "perceive", is carried out by ACCOM, and Steps 12 & 13 are carried out by SPLAN.

A task statement as required in step 6 is presented as two parameterized world models, a starting and a goal world model. For example the world models corresponding to the task "Push the square object into the next room" will have two rooms, a square object, and a connecting doorway, the only difference between the start and goal world models being in the position of the square object. The problem in step 6 is to reconcile the current default world model with the world model implied by the task statement.

## 4 Conclusions

We have briefly introduced a simulated environment and simulated robot and have sketched the design of a robot-controller capable of executing simple tasks in this robot world.

A spatial reasoning module is an important and essential part of any robot-controller. It makes plans for action on the basis of the current collection of hypotheses about the form of the environment. We have outlined a new approach to problems of spatial reasoning based on the use of the skeleton of a two-dimensional shape. When the environment has been drawn on an array of points like a screen, an iterative algorithm requiring a constant amount of computation reduces any pathfinding problem to a graph-traversal problem. Each edge of the graph corresponds to a path between two objects, each node corresponds to a junction of three or more paths, while the number of nodes is reduced to a minimum. Thus the amount of subsequent search is reduced. There are other ways to do this that are based on a Cartesian representation of the shapes of objects, but which may require more search if the shapes in the environment have much extraneous detail.

An interesting technical problem was found in this approach, that we call the rope-tightening problem. Namely, when one reasonable obstacle-avoiding path between two points has been found, how can this path be optimized to be as short as possible?

Though there have been several robot simulation programs written before, ours is among the first to handle the movement and collision of two dimensional shapes. Of the previous two major robot simulations, [Becker & Merriam,1973] used a Cartesian representation and [Nilsson & Raphael,1967] used a digital representation of shapes. The Cartesian

represents shapes as a series of points given by Cartesian coordinates while the digital represents a shape directly as an array of points like a screen. Of the previous rigid object motion simulations, [Eastman,1973] and [Pfefferkorn,1975] used Cartesian representations while [Funt,1976] and [Baker,1973] used digital representations. Our simulation uses a combination of the Cartesian and the digital representations to simulate the motion and collision of objects on a tabletop. The TABLETOP and UTAK simulation programs, which execute actions and produce tactile and retinal impressions, have been implemented. For much more detail see the first author's thesis [Rowat,1979].

Acknowledgements

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant no. A-5552.

References

[Baker,1973]
Baker, Richard. A spatially oriented information processor which simulates the motions of rigid objects. Artificial Intelligence, 4 (Spring 1973), pp.29-40.

[Becker & Merriam,1973]
Becker,J.D., and Merriam,W. "Robot" computer problem-solving system. Report 2646, Bolt, Beranek & Newman Inc., 1973.

[Blum,1967]
Blum, H. A transformation for extracting new descriptors of shape. In: W.Walthen-Dunn (ed.), Models of the perception of speech and visual form. MIT Press, 1967, pp.362-380.

[Eastman,1973]
Eastman, Charles M. Automated space planning. Artificial Intelligence, 4(1) (1973), pp.41-64.

[Funt,1976]
Brian V. Funt. WHISPER: A computer implementation using analogues in reasoning. Technical report no. 76-09, Department of Computer Science, University of British Columbia, 1976.

[Nilsson & Raphael,1967]
Nilsson,N.J., and Raphael,B. Preliminary design of an intelligent robot. Computer and information sciences, 7(13) (1967), pp.235-259.

[Pfefferkorn,1975]
Pfefferkorn, C.E. A heuristic problem solving design system for equipment or furniture layouts. Communications of the ACM, 18(5) (May 1975), pp.286-297.

[Rowat,1979]
Rowat, P.F. Representing spatial experience and solving spatial problems in a simulated robot environment. Technical report 79-14, Department of Computer Science, University of British Columbia, 1979.

[Rowat,1980]
Rowat, P.F. A new iterative algorithm for the skeleton of a planar shape. In preparation.

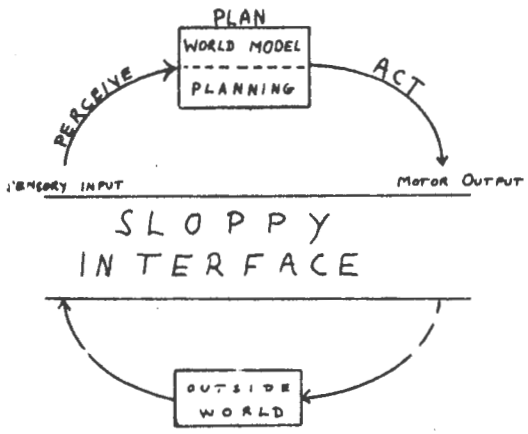



FIGURE 1. THE ACTION CYCLE

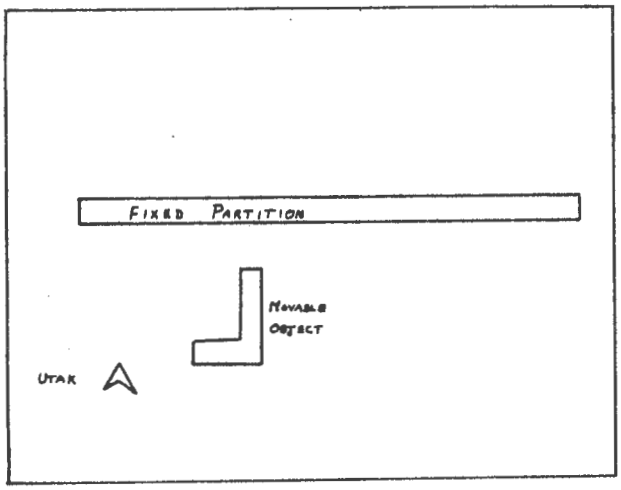



FIGURE 2 A TASK ENVIRONMENT

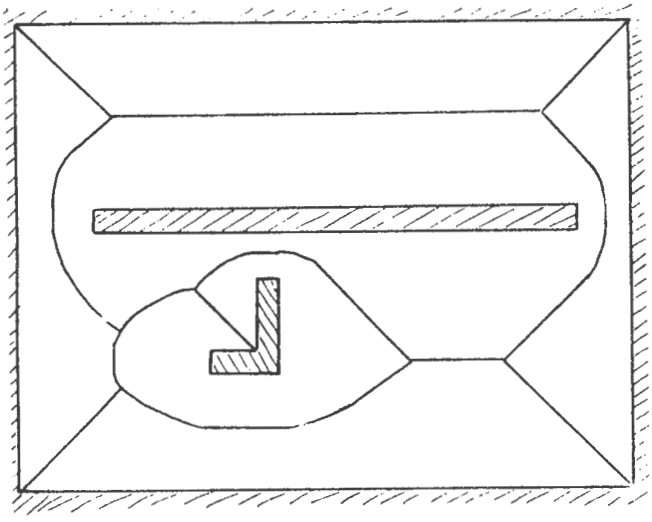


FIGURE 4. THE SKELETON OF A SHAPE.

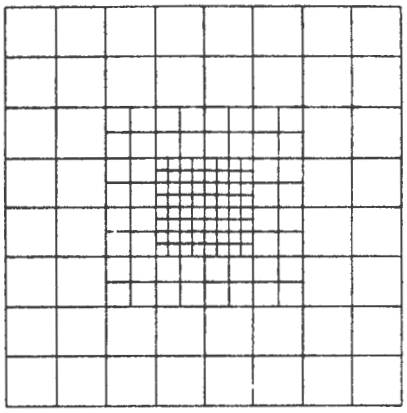
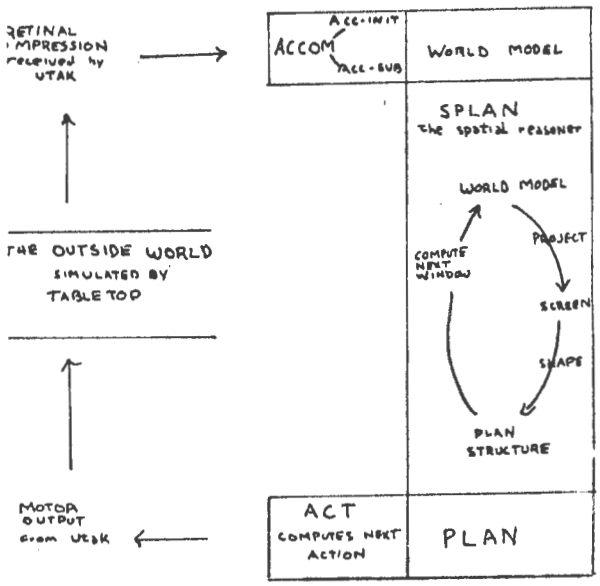


FIGURE 3(a) The retinal geometry.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 2 2 2 2 2 2 2 2 / 2 2 2 2 2 2 2 2 | | | | 2 | 2 |
| 6 | 0 | 0 0 0 0 0 0 0 0 / 0 0 0 0 0 0 0 0 | | | | 2 | 0 |
| 6 | 0 | 0 0 0 0 0 0 0 0 / 0 0 0 0 0 0 0 0 | | | | 1 | 0 |
| 6 | 0 | 0 0 0 0 0 0 0 0 / 0 0 0 0 0 0 0 0 | | | | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |

FIGURE 3(b) The integers in the squares form the retinal impression corresponding to Utak's position in Figure I.2.




ELABORATED

FIGURE 5. ACTION CYCLE

# Causality Analysis in Chess

David E. Wilkins
SRI International
Menlo Park, California 94025

**Abstract**

CAPS, KAISSA and PARADISE perform causality reasoning in chess. This paper defines the problem, demonstrates its difficulty, describes the contributions of PARADISE, and compares the approaches of the three programs.

## 1. Introduction

Human chess masters discover things during their searching process that they then use for analysis. For example, suppose a master mentally searches a move in a position and discovers that the move loses because of a three-move mate he had not previously noticed. Having discovered this three-move mate, he will not mentally search other moves unless he knows they will avoid this mate. Most chess programs will search all moves in this situation, continually rediscovering the mate each time (which may mean generating a large and expensive tree).

Three programs, KAISSA [1], PARADISE [5,6], and CAPS [2], try to avoid this by doing causality reasoning. KAISSA uses a technique called the "method of analogies", CAPS its "causality facility". PARADISE also employs a causality facility, which is based on the ideas used in CAPS, but with some improvements and clarifications. These programs all try to determine the "influence" a suggested move might have on an already searched line, so that moves that cannot influence a winning attack by the opponent are not searched. The term "causality facility" will be used in this paper to refer to all three methods (i.e., includes the method of analogies). Sections 2 and 3 of this paper define the problems facing a causality facility and illustrate their difficulty. Section 4 describes the improvements PARADISE has made on the causality facility in CAPS, while Section 5 briefly compares the three approaches.

## 2. The Problem

The position in Figure 1 derives from an actual master game (position 6 in [4]) after White plays his rook from b6 to b7. (This saves the rook from capture by the Black pawn.) Suppose now that a program does not notice Black's threat of a back rank mate, and attempts to save Black's rook by playing it to c5. White replies R-b8 which a program like PARADISE recognizes as equivalent to mate. After backing up to the position in Figure 1, the program has additional suggestions to move the Black rook to d5, e5, and f5. The idea behind the causality facility is that the program will recognize the threatened back rank mate in the refutation of R-c5 so that it does not repeat its mistake. If the discovery of the mate had taken hundreds of nodes of searching, it would have been expensive to try these three other rook moves. In Figure 1 PARADISE, for example, uses its causality facility to do this type of reasoning, and reject R-d5, R-e5, and R-f5 without searching (after searching R-c5).

Figure 1 shows the type of reasoning the causality facility is intended to do, but Figure 2 shows how difficult the problem can be. In Figure 2, R-c5 fails just as it did in Figure 1; now, however, R-e5 should not be rejected, although R-d5 and R-f5 should be. The only influence R-e5 has on the back rank mate is to attack one of the squares over which the "final" check is given. If the system were to say that R-e5 influences the mating line because it attacks a square over which an attack is given, it would search many moves that should be rejected (e.g., R-f5 and R-d5 in this position). On the other hand, a general analysis of which possible interpositions may work is extremely difficult. Each side may have a number of pieces bearing on the interposition square, some of which may be pinned. This example shows the subtle ways in which a move may influence a line.

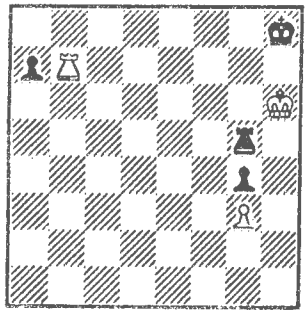


**Figure 1**
**black to move**

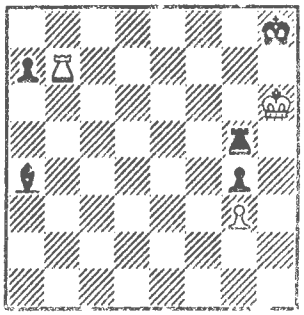


**Figure 2**
**black to move**

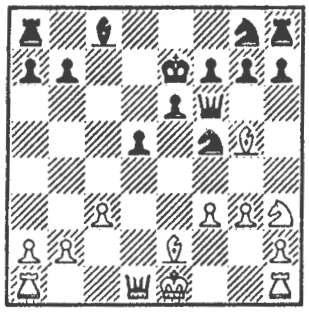


**Figure 3**
**black to move**

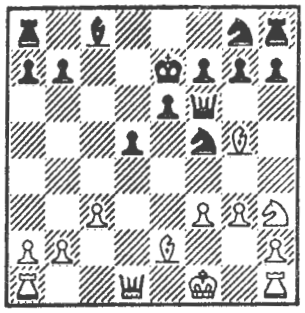


**Figure 4**
**black to move**

The above two positions show that possible counterattacks by the opponent may subtly modify the influence a move might have on a line. In Figure 3, there is no way Black can save his queen. Suppose a program first plays B-d7 for Black (for lack of anything better). It quickly finds that white obtains a distinct advantage after the White bishop captures the queen and Black recaptures. It is fairly obvious to human masters that P-b6, N-h6, and other such moves cannot save the Black queen. The causality facility should prevent a program from searching such moves, since they do not influence the losing line produced for B-d7.

Now consider Black's move K-d7. It seems obvious that this cannot influence white's winning of the queen, and so should not be searched. In Figure 3 this is true, but in Figure 4, K-d7 should be searched because it allows a Black counterattack that may save the position. The only difference in the positions is that the White king is moved over one square in Figure 4, but now if White answers K-d7 by capturing the Black queen, Black can reply N-e3 winning White's queen. The fact that K-d7 helps prepare a counterattack influences White's winning line in Figure 4. Such subtle influences are hard to recognize.

### 3. Tradeoffs

There is a tradeoff between recognizing general influences (i.e., viewing any type of interaction as an influence), and recognizing only more specific influences. By recognizing general influences, it is possible to make almost no mistakes at the cost of searching many moves that the program thinks influence a line (although a human could see that some of them have no influence). By recognizing more specific influences, it is possible to reject more moves without searching at the cost of possibly introducing more errors.

The performance of PARADISE's causality facility on the above examples illustrates this. PARADISE handles the problem in Figure 2 by recognizing the influence of an attack on a square over which an opponent's attack is given only when the side on move already has a piece bearing on the square, and the opponent's attack is a check. This is fairly specific and may introduce errors. PARADISE handles the problem in Figure 4 in a more general way. Neither PARADISE, CAPS, nor KAISSA would be able to recognize the difference in influence of the move K-d7 between Figures 3 and 4. It appears that the causality facility in CAPS would not see K-d7 as avoiding the loss of the queen, so it would not search that move in either position (thus making an error in Figure 4). It also appears KAISSA would not notice that K-d7 can improve the situation in Figure 4. PARADISE assumes K-d7 can influence the loss of the queen, since the queen is no longer captured with check. It searches this move in both positions, thus avoiding the error made by the other programs, but at the cost of unnecessarily searching it and similar moves in positions like those in Figure 3. In this case PARADISE leans towards greater generality and elimination of errors.

These examples show what the causality facility is intended to do and also show how difficult the problem is. PARADISE, CAPS, and KAISSA have facilities for doing this type of reasoning, but they all make compromises between effectiveness and correctness when deciding the specificity of the influences to be noticed.

## 4. PARADISE's Contributions

The causality facility reasons about a move and a tree produced by searching an unsatisfactory line. (CAPS and KAISSA do not have trees to analyze, only certain sets -- see Section 5 for a discussion.) To reject a proposed move without searching, two determinations must be made by the causality facility. First, it must conclude that it was not the first move in the unsatisfactory line that permitted the ensuing consequences. This is referred to as a PERMIT determination in PARADISE. Second, the causality facility must decide that the proposed move could have no influence on the unsatisfactory line. This is referred to as an AFFECT determination. CAPS does not distinguish between these two functions, but PARADISE shows that the distinction can be important and useful. For example, the PERMIT determination is also used by PARADISE in its defensive search to instigate a null move analysis (i.e., letting the opponent make two moves in a row during the search) when all defensive moves have permitted their refutations (see [5]).

During the PERMIT determination, PARADISE looks for any of seven permitting influences the given move may have in the tree (e.g., vacating a square which was moved over by the opponent, removing protection from a piece that was captured). These influences are described in detail in [5]. If any influences are found during the tree traversal, the move permits the unsatisfactory tree. To reject a proposed move from consideration on the basis of an unsatisfactory tree, the causality facility first does a PERMIT determination using this tree -- with the first move of the unsatisfactory tree being the one checked for permitting influences. If any permitting influences are found, this tree normally cannot be used to reject another proposed move since its first move permitted the refutation. At this point, CAPS gives up. However, PARADISE next checks whether the proposed move permits the unsatisfactory tree. If it permits the tree for all the same reasons that the first move in the tree permitted it, the tree can still be used to reject the proposed move since the proposed move would also permit the tree. (See below for an example.) CAPS cannot do this, because its equivalent to the PERMIT determination returns a yes or no answer instead of a list of influences.

If the PERMIT determination finds that the unsatisfactory tree did not permit its own refutation (or that the proposed move permits the refutation for all the same reasons), an AFFECT determination is done to find any influences the proposed move might have on the unsatisfactory line. This involves another tree traversal, in which the causality facility looks for possible counterattacks and any of ten affecting influences which are described in [5]. If any influences are found during the tree traversal or counterattack analysis, the proposed move affects the unsatisfactory tree. If the proposed move does not affect the tree, it is rejected without searching. If it does, then PARADISE checks whether the first move in the unsatisfactory tree also affects the tree. If so, the proposed move is rejected unless it affects the tree for some reason that the original move in the tree did not. Again, this last test is not performed in CAPS since it does not have a list of influences.

How PARADISE's causality facility solves the problem in Figures 1 and 2 should help clarify this description of it. In Figure 1, the unsatisfactory line consists of a tree with only two nodes: one for BR-c5 and one for WR-b8. PARADISE first checks whether BR-c5 permits this line and finds one reason why it does so. While looking at the node for WR-b8, the causality facility notices that BR-c5 unprotects a square over which the white rook delivers a check, and that black has another piece bearing on that square. Thus PARADISE realizes the black rook could have interposed on g8 if it had not moved. This line cannot be used to reject another move unless the proposed move also unprotects g8 (thus permitting this line for all the same reasons). PARADISE next does a PERMIT determination for BR-e5 and finds that this move also unprotects g8, thus permitting the line for all the same reasons R-c5 does. An AFFECT determination is then done for R-e5 and this line; however, since no reasons are found R-e5 is rejected without searching as it does not affect the line. BR-d5 and BR-f5 are rejected in the same manner.

In Figure 2, the same unsatisfactory line is generated for R-c5. All PERMIT determinations produce the same results as in Figure 1. However, the AFFECT determination for BR-e5 now discovers one reason why the move affects the line: it attacks g8, a possible interposition square for the check from b8, that is also attacked by another Black piece. An AFFECT determination is now done for BR-c5 to see if this move affects the line for all the same reasons. R-c5 is found not to affect the line, so BR-c5 is not rejected. PARADISE finds that BR-d5 and BR-f5 do not affect the unsatisfactory line, so they are rejected without searching.

## 5. Comparison of Causality Facilities

KAISSA, CAPS, and PARADISE have different implementations of causality reasoning. The causality facility in PARADISE is explained above. The causality facility in CAPS reasons about an unsatisfactory line. Like PARADISE, it ascertains whether the line was permitted by its first move. If not, CAPS tries only moves generated by the causality facility as countercausal moves (i.e., they should influence the unsatisfactory line). The effect is the same as in PARADISE, where moves generated by any mechanism (countercausal analysis or otherwise) are checked for their countercausal nature by the causality facility. PARADISE has a major advantage over CAPS in that it can compare the influence two different moves have on the same line since it returns a list of influences.

The method of analogies (used in KAISSA) is considerably different from these two causality facilities. It reasons about two positions: the current one, and one that has already been searched and found to be unsatisfactory no matter which move is played. For the method of analogies to apply, all admissible moves (defined in [1] as "moves which make sense in the strategies under consideration") in the current position must also be admissible in the unsatisfactory position. The method of analogies then determines if any of the differences in the current position can influence any of the unsatisfactory lines. (Information about the lines is retained as a number of sets, as described below.) If not, the new position can then be assumed unsatisfactory without searching any moves in it.

Unlike those of CAPS and PARADISE, KAISSA's approach to causality cannot reason about individual moves, but can only determine that a whole position is unsatisfactory. The method of analogies is only applicable when all lines from one position have failed and when there is another position whose admissible moves are all admissible in the unsatisfactory position. It seems that such a situation would not occur frequently, and KAISSA's authors do not assess, even subjectively, how the method of analogies has performed in their program. Reasoning about individual moves is useful, for example, in positions where there is only one good move and many bad ones that lose to the same complicated attack. Once CAPS and PARADISE have tried one bad move, their causality facilities should eliminate all the other bad moves without searching them. KAISSA would have to try all moves, and the method of analogies would help only if it could be applied to positions deeper in the tree.

The major difference among these three causality facilities is the wealth of information returned from a previously analyzed line in PARADISE. In both KAISSA and CAPS, all that is returned from an analyzed line is a number of sets. These sets contain squares that were moved to, squares that were moved over, pieces that moved, squares that were newly attacked by moves that were made, pieces that were attacked, and so forth. These sets do not contain sequencing information (e.g., what order moves occurred in, whether new attacks existed simultaneously), and do not show which move caused which effects (e.g., for a square in the set of "squares moved to", it is not possible to determine which piece moved to that square). PARADISE, on the other hand, does return this kind of information. After searching a line, PARADISE returns a tree that contains every node generated during the search. Each node describes which piece was moved, its origin and destination squares, and a number of attacking patterns (including discovered attack patterns) that matched at that node. With such a tree PARADISE can determine which attacks were happening simultaneously, which moves were first, which pieces moved to which squares, and which moves caused which attacks. Returning this much information would probably not be practical in KAISSA since it produces much larger trees than PARADISE.

As described above, it is very difficult to notice all the subtle ways a move might influence a line. Any of these three causality facilities can make a mistake by not recognizing a subtle influence, which may cause the program to reject a good move without searching it. ([1] cites a "proof" that the method of analogies never makes a mistake. However, the definition given in the proof for "influence" is inadequate and does not appear to recognize such subtle influences as those described in Section 2. For example, all moves in Figure 3 fail and it appears that the method of analogies would be applicable to Figure 4, yet would not detect the influence.) There is always the tradeoff of recognizing general influences and making almost no mistakes at the cost of searching many unnecessary moves, and recognizing more specific influences that will search fewer moves at the cost of possibly introducing more errors.

It is difficult to compare performance of the three implementations. There are a few examples of CAPS's causality facility in [2], and PARADISE appears to make more causality cutoffs over this small set than CAPS does (see [5]). (PARADISE's improvements on the causality facility in CAPS have already been mentioned.) The author is aware of no published example or subjective evaluation of the method of analogies in KAISSA. The algorithms of the three programs are not similar enough

to verify subsumption, since each program uses different tradeoffs in generality when checking for a particular influence. One thing is clear: PARADISE has enough information available both to make fewer mistakes and make more cutoffs than either CAPS or KAISSA. (It would be a difficult project at present to show that either of these has actually been accomplished.) This is clear because PARADISE's trees contain so much more information than do the sets in the other programs. By using its knowledge base to produce small annotated trees, PARADISE provides more knowledge for its causality facility to use.

## 6. Summary

The difficulty of the causality problem has been shown, and PARADISE's contribution described. PARADISE tries to avoid errors by noticing many kinds of influences. The causality facility has been effective and error-free over PARADISE's test domain of 92 positions (see [5]). (The trees generated by PARADISE are so small that all causality decisions can be checked.) However, neither the PERMIT nor the AFFECT influences are adequate to handle all middle game positions without errors (see [5]).

It is not clear at present how to formulate a causality facility that would never make errors, yet be effective for rejecting moves without searching. It is not even clear whether it is possible to do so. ([3] discusses these issues.)


## Acknowledgements

The author is indebted to Hans Berliner for his continuing assistance and counsel during this research, to the Stanford Artificial Intelligence Laboratory which provided the necessary environment and tools for doing this research, and to SRI International which supported the writing of this paper.


## REFERENCES


[1] Adelson-Velskiy G M, Arlazarov V L, Donskoy M V, "Some methods of controlling the tree search in chess programs", Artificial Intelligence 6, pp. 361-371, 1975.

[2] Berliner H, "Chess as problem solving: The development of a tactics analyzer", Unpublished doctoral thesis, Carnegie-Mellon University, 1974.

[3] Berliner H J, "On the use of domain-dependent descriptions in tree searching", in: Perspectives on Computer Science, Jones, A.K. (Ed), Academic Press, 1977.

[4] Reinfeld F, Win At Chess, Dover Books, 1958.

[5] Wilkins D E, "Using patterns and plans to solve problems and control search", AIM-329, Computer Science Department, Stanford University, 1979.

[6] Wilkins D E, "Using plans in chess", Proceedings of Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979, pp. 960-967.

# PATTERN-BASED REPRESENTATIONS OF KNOWLEDGE: IN SEARCH OF THE "HUMAN WINDOW"

M.A. Bramer

Mathematics Faculty,
The Open University,
Milton Keynes, MK7 6AA,
England

## Abstract

The focus of much recent Artificial Intelligence research into Computer Chess has been on programming the endgame. This paper discusses some of the specific reasons for complexity in the endgame and considers the effects of such complexity on human chess-playing strategy, textbook descriptions and the development of programs. In programming the endgame the researcher is faced with a range of decisions concerning the quality of play to be aimed at, the balance between knowledge and search to be adopted and the degree to which the playing strategy should be understandable to human chess-players. The term "human window" has been used to describe that range of programs which not only perform at a high level of expertise but are comprehensible to subject experts. A model for representing pattern-knowledge is described which has enabled the development of algorithms to play a number of endgames. Three algorithms representing different levels of performance for the endgame King and Pawn against King are compared, in order to discuss the tradeoff between complexity and completeness, on the one hand, and compactness and comprehensibility, on the other. Finally, the role of search in reducing the amount of knowledge to be memorised is considered and an extension to the basic model to incorporate a deeper searching element is discussed.

## Introduction

The game of Chess has been used as a task area for Artificial Intelligence research for over a quarter of a century, as one in which complexity is combined with a well-defined structure, together with an extensive background culture against which a given standard of program performance can be evaluated.

The focus of much recent research into computer chess has moved towards the study of endgames (i.e. subgames with only a small number of pieces remaining). Endgames retain much of the complexity of the full game of chess whilst in many cases affording the possibility of controlled experiments and precise quantitative analysis. It is also notable that conventional chess-playing programs using deep search with simple evaluation functions generally perform very badly in endgames, where knowledge, rather than calculation, is probably the major factor in human play.

Studies of fundamental endgames such as King and Pawn against King (KPK) and King and Rook against King (KRK) or of slightly more complex ones such as King and Rook against King and Knight (KRKN) have revealed surprising complexity. Knowledge-based algorithms for a variety of endgames have been given by Bramer (1977b), Bramer and Clarke (1979), Bratko, Kopec and Michie (1978) and elsewhere, in each case developed after a lengthy series of trials and careful refinement.

There are a number of reasons for this unexpected complexity.

(1) Limitations of current theoretical knowledge of chess. Experiments have revealed important errors and omissions in the known theory of particular endgames, with erroneous evaluations previously made by experts and counter-intuitive moves and results found in several important positions.

KPK and KRK are believed to be fully understood theoretically by reasonably strong players. Nevertheless there are difficult cases which are not given in the majority of widely available textbooks, or (in some cases) in any of them. The authors have either been unaware of the difficulties or have excluded them as unimportant.

(2) Boundary effects caused by the board edge. Inability to manoeuvre beyond the Rook files or the first or eighth ranks leads to difficulties and "special cases" affecting general strategies (particularly for KPK).

(3) "Discontinuities" in the rules of chess. Stalemate, the option of an initial double Pawn move and Pawn promotion can be viewed as "discontinuities" in the normal rules (being unable to avoid King capture loses, pieces move in the same way anywhere on the board) and both are significant in endgame play.

(4) Chessboard geometry. The geometry of the chessboard is non-Euclidean (and varies from piece to piece). Measuring in terms of King moves (one square vertically, horizontally or diagonally at a time) the distance from square A1 to square A7 is 6 units either directly or via two sides of a triangle (A1-B2-C3-D4-C5-B6-A7). The situation is compounded by the existence of squares on to which a King may not legally move, which alters its "effective distance" from a given square. Some analysis of effective distance in the KPK case is given in Bramer (1977a).

With this geometry it is difficult to define even apparently simple geometrical relationships, such as "Black King can take Pawn" for KPK.

From the above it is clear that endgames, especially those with only a small number of pieces, differ from middlegames by being much more highly ill-behaved with numerous special and unexpected cases arising. Moreover the traditional

computer chess technique of using a sophisticated search algorithm with a fairly simple evaluation function is not applicable (at least without major modification) to endgames, where it is easy to find examples of positions which would require a search of 30 ply or more deep to find the one (counter-intuitive) winning move.

Although simple in structure, chess endgames are not a "toy" domain (such as the blocks world) but a highly complex and badly-conditioned one. It is not that conventional programming has proved ineffective for the endgame which is surprising, but that human players with even a small amount of training should appear to ignore the difficulties and play so well.

Textbook descriptions of endgame strategies

From the chess literature it is evident that endgame play depends much more on the use of plans based on a knowledge of significant configurations (or patterns) of pieces than on deep analysis of possible variations.

For fundamental endgames such as KPK and KRK, the plans are very simple (e.g. "move as close to White's Pawn as possible") and the search very shallow, in fact almost non-existent.

In conventional chess-playing programs, storing chess knowledge (of weak squares, say) can be viewed as helping to reduce the amount of search required. In endgame play (especially with a small number of pieces) it is probably more accurate to think of using search as a means of reducing the amount of knowledge that must be stored, either for a person or for a program.

A typical textbook description comprises a small number of general "rules of play" together with some example variations from diagrammed positions. The rules are normally only imprecisely worded and omit important details which have to be inferred from the variations given. Nevertheless, this information is considered to be a complete and sufficient explanation for the reader and it is intended to have essentially the same meaning for every reader of the text.

Although standard textbooks such as Fine (1941) are often thought of as definitive and exhaustive, this is far from true. Aside from gaps or errors in chess theory as mentioned previously, there is no attempt made to deal with all possible situations which can arise even in the simplest endgames. Fine remarks "I have given a large number of rules which are at times incorrect from a strictly mathematical point of view, but are nevertheless true by and large and are of the greatest practical value". Thus he concentrates on the typical cases to the exclusion (in general) of rarely arising exceptions even when these are known.

Even in the highly analysed and practically important case of KPK, it is clear that no effort has been made to demonstrate the most efficient strategy (in the sense of the shortest possible win in every position). The emphasis is on the simplest possible explanation of how to handle this complex endgame in practical play and the reader is required to supplement the knowledge he has gained from the textbook as a result of inference and practical experience.

Programming the endgame

In attempting to model the strong player's knowledge of the endgame, the researcher is faced by a number of decisions. One is whether to adopt a structural or a procedural representation, another is the level of performance for which he should aim. A helpful distinction can be made between winning algorithms which are optimal (i.e. the stronger side wins wherever possible in the smallest possible number of moves) and those which are correct (the stronger side wins wherever possible but not necessarily as quickly as possible). The evidence and examples given in Bramer (1978) strongly suggest that, even for KPK, strong players perform sub-optimally, although almost certainly correctly. For complex endgames it is quite possible that strong players do not always perform even correctly.

There is a principle of sufficiency involved here. The game-theoretic maximum number of moves needed for the stronger side to win any winnable KPK position is only 19. The rules allow for 50 moves (without any piece taken or any Pawn moved) before a draw can be claimed. It is simply not worthwhile to overload the memory with numerous special cases (or spend time performing a deep analysis) to achieve optimal play, even assuming this is feasible, if there is a simple algorithm which suffices for correctness, still well within the constraints of the 50-move rule.

On the other hand, the endgame King, Bishop and Knight versus King is thought to require up to 34 moves to win in some cases and, in practice, an error in certain critical positions can easily lead to an exceeding of the 50 move limit. In these cases it is worthwhile memorizing much more detail of difficult cases, although not necessarily all of them.

Clarke (1977) draws attention to the tradeoff between knowledge and search. At the extremes are a program which has full knowledge and uses no search (i.e. it simply looks up the best move in a table) and one which uses extensive search and has no non-trivial knowledge (i.e. it uses only the definitions of won, drawn and lost terminal positions). In general, programs will lie somewhere along this spectrum, with recent Artificial Intelligence research concentrating on programs towards the "knowledge" end.

Kopec and Michie (1979) have described another tradeoff: this time between the performance of a program and its comprehensibility to subject experts. High-performance programs which are also comprehensible are referred to as lying inside the "human window". The importance of this concept can be seen by considering a hypothetical program which performs medical diagnosis and recommends treatment, such as major surgery. Given the well-known prevalence of "bugs" even in well-tested systems, it is essential for medical practitioners to have confidence in such a diagnostic system even when it makes proposals which conflict with

their own experience and judgement. With an appropriate representation, such as the production rules used by Mycin, it may be possible to satisfy subject experts of the accuracy of such judgements, for example by describing the factors which were taken into account, the weighting given to each, the diagnostic rules applied and the reliability attached to the result.

Such considerations argue strongly in favour of the choice of a structural representation, particularly one based on rules or patterns. It would be extremely hard to justify a decision based on deep search, possibly recursive, and fine tuning of the weightings of terms in an evaluation function such as that used in standard tournament chess-playing programs. Another advantage of a pattern-based approach is that as well as judging the value of the rules given, subject experts can add their own experience in codified form.

An interesting case where trust in an unfathomable program was required has already arisen in chess. Michie (1977) reports that the grandmaster Bronstein has made use of a database of the best move in every position for part of the King, Queen and Pawn against King and Queen endgame for analysis of an adjourned position. This comprises more than a hundred million positions stored on nine magnetic tapes. If the move retrieved from the database had conflicted with Bronstein's own judgement it would have been virtually impossible to check whether it arose from an error in creating the database or was in fact accurate.

Rule-based representations of a body of knowledge can be viewed as having two possible functions: one as a replacement for the textbook, to be committed to memory by the chessplayer, the general medical practitioner etc. and used as required, the other is as an expert computerized assistant typically used in an interactive mode. In both cases, there is an important need for comprehensibility. However, a set of rules for the former will generally need to be much briefer than for the latter, to match the limitations of human short-term memory, and again there is a tradeoff, this time between accuracy (or completeness) and compactness within a given framework.

A desirable feature of a rule-based expert system is that learning within it will generally proceed monotonically, i.e. that adding a new rule should not invalidate old ones but should lead to an improvement in performance. This is only likely to be true if the underlying representation is well chosen. The proliferation of rules each covering a small number of cases which the subject expert would not regard as reflecting aspects of the complexity of the domain in question is a good indication that the representation is probably not appropriate.

The weakness of a general-purpose representation such as the hierarchical model used by Negri (1977) for KPK and the standard form of inductive learning performed within it is that it fails to take into account the specific features of the domain under consideration. Thus it may be that in specifying the "King can catch Pawn" predicate, some descriptors should always be used in preference to others where possible or that some descriptors should only be used in conjunction with others, or if others do not appear, etc. Given such knowledge for KPK, a much more compact description may be possible for "King can catch Pawn".

A model for representing pattern-knowledge for chess endgames

A representation designed to enable the chessplayer's knowledge of an endgame to be represented in a structurally simple and compact form, capable of incremental iterative refinement to improve its performance whilst preserving its initial properties, is given in Bramer (1977b) and Bramer and Clarke (1979) and summarized below. It is assumed that the problem is to construct an algorithm to find a move for a chosen side (say White) in any position p, for a given endgame. The basic move finding algorithm is then as follows:

(a) generate the set Q of immediate successors (Black to move) of p
(b) find the highest ranked member of Q, say q
(c) play the move corresponding to q.

To achieve step (b) an implicit ranking is defined on the set Q* of all legal BTM (Black to move) positions for the endgame in question. Each such position is assigned to exactly one of a number of disjoint and exhaustive classes which partition the set Q*.

The ranking of each BTM position is then determined by its class value (which is constant for all the positions in any class) and the values of a number of associated functions. These vary from one class to another, in general. For positions in the same class, the functions used are always the same although their values will vary from one position to another. To compare the values of two positions, their class values are compared, with the larger value indicating the higher-ranked position. If there is a tie, the first associated function is used for comparison. If there remains a tie, the second associated function is used, and so on. (Any ties remaining after all the associated functions have been used are resolved arbitrarily.) When comparing the values of associated functions, in some cases the larger value is preferred, in some cases the smaller is, depending on the particular function. The intention is that each class should correspond to some significant static feature of the endgame as perceived by chessplayers, e.g. "Black is in check". The associated functions correspond to relevant numerical values, such as the distance between the two Kings.

Assigning a position q to a class is achieved by working through a series of predicates (called rules) in turn until one is satisfied. (Subsequent rules are not evaluated.) A position

q is defined to belong to a particular class N if and only if rule N is satisfied by q and none of the preceding rules are satisfied.

This procedure ensures that each position belongs to only one class and helps to simplify the definition of the rules. To ensure that each position belongs to some class, the final rule is defined to be always true for any position q.

This model was used initially to develop an algorithm for the stronger side (White) of KPK which was thought to be a fully correct strategy as a result of extensive testing, reported in Bramer (1977b). Subsequent analysis revealed that the algorithm was not entirely correct. An optimal strategy was developed by a process of iterative refinement using a database of the shortest-path winning move (or moves) in every position. A correct algorithm has now been refined from the original 'near correct version' by the semi-automatic refinement process based on inspection of "Win-trees" given in Bramer (1979) and is summarized in the Appendix. (The development of a correct algorithm for KRK is described in Bramer (1979).)

## Three algorithms for King and Pawn against King - a comparison

The following figures give basic information about each of the algorithms.

Figure 1 - Three algorithms for KPK

| Algorithm | Description | Classes | Associated Functions | Max. depth of win (ply) |
|---|---|---|---|---|
| A | "near correct" strategy | 19 | 9 | |
| B | correct strategy | 20 | 10 | 44 |
| C | optimal strategy | 38 | 13 | 38 |

Figure 2 - "Optimality Levels" for algorithms A (near-correct) and B (correct)

| | Algorithm A* | Algorithm B |
|---|---|---|
| Move played is optimal | 59,888 (95.93%) | 60,462 (96.77%) |
| Move played increases depth by 1 | 1,526 (2.44%) | 1,075 (1.72%) |
| " " " " " 3 | 673 | 660 |
| " " " " " 5 | 251 | 222 |
| " " " " " 7 | 68 | 47 |
| " " " " " 9 | 19 | 11 |
| " " " " " 11 | 5 | 1 |
| " " " " " 13 | - | - |
| " " " " " 15 | 2 | 2 |
| Total | 62,432 | 62,480 |

(Breakdown for all legal WTM positions, which are theoretical wins).

* Excluding non-win preserving moves

Figure 3 - Class membership for algorithm B (correct strategy)

| Class | Number of positions (BTM) | Class | Number of positions (BTM) |
|---|---|---|---|
| 1 | 10,093 (10.3%) | 12 | 60 |
| 2 | 9 | 13 | 176 |
| 3 | 12,985 (13.3%) | 14 | 1,620 (1.7%) |
| 4 | 35,026 (35.7%) | 15 | 8,507 (8.7%) |
| 5 | 14,422 (14.7%) | 16 | 2,632 (2.7%) |
| 6 | 694 (0.7%) | 17 | 1 |
| 7 | 50 | 18 | 4,971 (5.1%) |
| 8 | 6,045 (6.2%) | 19 | 5 |
| 9 | 42 | 20 | 4 |
| 10 | 66 | Total | 97,992 |
| 11 | 584 (0.6%) | | |

(Pawn on file A-D, ranks 2-8.)

Classes 2,7,9,10,12,13,17,19 and 20 total together 413 members (0.42%).

Figure 2 shows the "Optimality Levels" for algorithms A and B; i.e. the amount by which the move selected in each theoretically won WTM position changed the maximum depth. For an optimal move the depth is decreased by one ply, for all other moves it is increased by an odd number of ply (the fewer the better). The figure shows that in both cases the great majority of moves are either optimal or increase the depth by only one ply. The differences between the two algorithms are small and, in fact, algorithm B was formed from A by the addition of one new class (with only one member) and one new associated function plus slight changes affecting a few other classes.

However, the difference in performance is substantial. Algorithm A fails to win from as many as 4,602 theoretically won positions (WTM) and 4,351 (BTM). Only in 48 positions (WTM) is a move played which does not preserve White's winning advantage, the other positions simply transform into one another in cycles. This result reinforces the evidence given in Bramer (1979) for the KRK endgame that a change to the move played in a small number of positions can drastically alter the overall performance of an algorithm. Testing even by expert human players might never reveal that A was not a correct algorithm. Its errors will in general result in a cycle but this may be after many moves of otherwise expert play, possibly in response to poor play by Black.

To improve the performance of KPK from correct (B) to optimal (C) requires an increase from 20 to 38 classes and from 10 to 13 associated functions. This near-doubling of the size of the algorithm results in a relatively minor improvement in performance. The number of individual positions played optimally rises from 96.77% to 100% and the maximum depth is reduced from 22 moves (44 ply) to 19 (38 ply). The definitions of 38 classes would probably be too many to commit to memory, if the algorithm were to be used as a replacement for the textbook, whereas 20 classes would probably be acceptable. Either number would be satisfactory for an expert computerised assistant. The pattern of distribution of depths for algorithm B is very similar to that for C (theoretical maximum depths), and this is also true for algorithm A, which would tend to support the appropriateness of the representation adopted. In making the transition from algorithm B to C, it is clear that a "diminishing returns" effect is involved. The maximum depth of 22 moves for algorithm B is still well within the 50-move drawing limit. For a practical player to take on the additional memory burden required to play optimally would simply not be worthwhile, an unnecessary violation of the principle of sufficiency.

The class membership table for algorithm B (Figure 3) shows that a fairly small number of classes account for the great majority of positions. The nine smallest classes contain less than a half of one percent of the positions, and four classes contain less than ten positions each. It is useful to consider whether classes with a low-level of membership reflect "special cases" of the domain in question or merely result from the particular representation used. In the case of algorithm B, the classes concerned do seem to correspond to clear special cases arising from the boundary effects, rule discontinuities etc. referred to previously. For example, Class 17 is used to deal with recognized difficulties with a Knight Pawn and Class 2 contains all the positions where Black is stalemated. Some of these special cases although important are not given in the major textbooks (or not in all of them) and this is more markedly so for algorithm C (optimal strategy). Bramer (1978) gives several examples of positions which are clearly special cases but are of no practical significance and would certainly never be quoted in textbooks. It is evident that textbooks omit many special cases, even those which are necessary for correct play, and that a major reason for this is to reduce the amount to be memorized.

It is not certain that algorithm B is a minimal correct strategy for KPK, i.e. one with the fewest number of rules possible. By removing certain of the special cases or by other changes it might be possible to construct an algorithm which was still correct but had a greater maximum depth, although still within the 50-move limit. Such a strategy would doubtless still have to include classes to deal with a (possibly reduced) number of special cases. If the aim were not to achieve correctness, but to perform expertly in practical play or to serve as a teaching document superior to standard textbooks, it could be argued that an abbreviated algorithm which omitted special cases but was still reliable in the great majority of cases was preferable. To return to the idea of the "human window", it may be that the window includes all three algorithms A, B and C and a spectrum of others including some which are not fully correct. In each case there is a tradeoff between complexity and completeness, on the one hand, and compactness and comprehensibility, on the other.

## Extending the model

The most important way in which people reduce the amount of knowledge memorized is by making use of analysis or search.

For experienced players, search plays little part in playing simple endgames (although exact counting does) but increasingly more as the problem becomes more complex. Unfortunately, tournament chess-playing programs have found it necessary to use search of a volume and kind which is most unlike that of expert human players. An extension of the model previously described enables the use of pattern-knowledge to be combined with search deeper than one ply, but which is capable of careful control.

The final class (defined to be always true) is called the residual class; those with class values greater than that of this class are called positive, those with lower values are called negative. These latter two categories broadly

reflect features of the endgame which are particularly desirable or especially undesirable. The most likely source of difficulty in programming complex endgames lies in specifying a sufficiently large number of positive or negative classes, with many important positions thus falling into the residual class.

There are four possibilities for a given set of successor positions:

(a) at least one belongs to a positive class;
(b) all belong to negative classes;
(c) all belong to negative classes, except one which belongs to the residual class;
(d) two or more positions belong to the residual class and the remainder (if any) belong to negative classes.

In cases (a), (b) and (c) the most favourable position can be found statically using position values in the usual way. In case (d), either the positions in the residual class can be taken as terminal and the associated functions used to calculate the value statically, in the usual way, or an analysis tree can be generated from each of the residual class positions, with the negative class positions rejected altogether.

Constructing an analysis tree (either depth-first or breadth-first) in this way has the effect of reducing the amount of search by pruning all branches to positions in negative classes (unless there is no alternative) and defining terminal states of a given set of positions as a whole (cases (a), (b) and (c) above). Since a residual class position can, at any stage, be regarded as terminal, with the static value of the position backed up the tree, the amount of analysis performed is subject to close control. In general, this form of the model is distinguished from conventional tree-searching in that search is intended to be used in a controlled way only as necessary to supplement the pattern-knowledge which it is believed is the fundamental component of the chessplayer's endgame knowledge. It corresponds roughly to the high-level rule "if in an unfamiliar situation, search for possible forcing variations into known positions". An implementation of this extended model, known as Kappa 2, is currently in progress.

## Appendix

### A correct algorithm for the endgame King and Pawn against King

Figure 4 - Classes for King and Pawn against King (Summary)

| Class | Property of position q (Black to move) | Class value | Associated Functions |
|---|---|---|---|
| 1 | Pawn en prise | 1 | 2, 3 |
| 2 | Black is stalemated | 2 | - |
| 3 | Pawn is on eighth rank | 20 | - |
| 4 | Pawn can "run" | 19 | 1 |
| 5 | Black King is effectively closer to the Pawn than White (adjusted for second rank case) | 3 | 2, 3 |
| 16 | (Some Rook Pawn cases) | 4 | - |
| 6 | Black can move to "blockade" square | 5 | 1, 2, 3 |
| 17 | (Knight Pawn position - special case) | 11 | - |
| 7 | White is on the "blockade" square and Black can take opposition | 6 | - |
| 8 | Black King at least two files from White King on same side of Pawn, White King not below Pawn's rank | 18 | 1, 4, 7 |
| 9 | Kings on critical squares on same rank | 17 | 1 |
| 10 | Kings on critical squares, Black one rank above White | 16 | 1 |
| 11 | White King on Pawn's rank, above Pawn | 15 | 1, 7 |
| 12 | Kings in vertical opposition, with the White King on a critical square | 14 | - |
| 13 | Kings in opposition, White King above Pawn or both on sixth rank | 12 | 1 |
| 14 | White King on a critical square | 10 | 1, 6, 5, 10 |
| 18 | White King on a file above Pawn | 9 | 4, 1, 8 |
| 19 | (Pawn on sixth rank-special case) | 8 | - |
| 20 | (Pawn on fifth rank-special case) | 13 | - |
| 15 | (always true) | 7 | 2, 3, 7, 9 |

Figure 5 - Associated Functions for King and Pawn against King

| Function | Value of function |
|---|---|
| 1 | The Pawn's rank* |
| 2 | The file or rank distance between the Kings, whichever is the larger** |
| 3 | The file or rank distance between the Kings, whichever is the smaller** |
| 4 | The file distance between the White King and the Pawn** |
| 5 | The file distance between the White King and the Pawn* |
| 6 | The number of ranks the White King is above the Pawn* |
| 7 | The White King's rank* |
| 8 | The rank distance between the White King and the Pawn** |
| 9 | The file distance between the Kings* |
| 10 | The White King's file** |

* The largest value should be taken
** The smallest value should be taken

References


Bramer, M.A. (1977a). King and Pawn against King: Using Effective Distance. Open University, Faculty of Mathematics, Technical Report.

Bramer, M.A. (1977b). Representation of Knowledge for Chess Endgames: Towards a Self-improving System. Ph.D. thesis, Open University.

Bramer, M.A. (1978). Representing Pattern-Knowledge for Chess Endgames: an Optimal Algorithm for King and Pawn against King. To appear in Clarke, M.R.B. (ed.), Advances in Computer Chess 2, Edinburgh University Press.

Bramer, M.A. (1979). Testing Correctness of Strategies in Game-playing Programs. Proceedings of the Sixth International Joint Conference on Artificial Intelligence.

Bramer, M.A. and Clarke, M.R.B. (1979). A Model for the Representation of Pattern-Knowledge for the Endgame in Chess. Int. J. Man-Machine Studies, 11, pp. 635-649.

Bratko, I., Kopec D. and Michie, D. (1978). Pattern-based Representation of Chess End-game Knowledge. Computer Journal, 21(2), pp. 149-153.

Clarke, M.R.B. (1977). A Quantitative Study of King and Pawn against King. In Clarke, M.R.B., Ed., Advances in Computer Chess 1. Edinburgh University Press, pp. 108-118.

Fine, R. (1941). Basic Chess Endings. David McKay, New York.

Kopec, D. and Michie, D. (1979). Problems of the "human window". Paper presented at AISB Summer School on Expert Systems, Edinburgh, July 1979.

Michie, D. (1977). End-game Secrets. Computer Weekly, 3rd November, (Chesslab).

Negri, P.G. (1977). Inductive Learning in a Hierarchical Model for Representing Knowledge in Chess End Games. In Elcock, E.W. and Michie, D. (eds.), Machine Intelligence 8. Ellis Horwood, Chichester, pp. 193-204.

# Searching Game Trees In Parallel

Selim G. Akl, David T. Barnard and Ralph J. Doran

Department of Computing and Information Science
Queen's University
Kingston, Ontario
Canada
K7L 3N6

## Abstract

Preliminary experiments are described with a program that simulates concurrency to process game trees in parallel. The alpha-beta algorithm is implemented to search disjoint subtrees simultaneously using knowledge about the behavior of the sequential algorithm to increase the number of cutoffs. This approach is found to result in a considerable speedup of the search. Some future research directions are suggested.

## 1. Introduction

> "Even though the real world is inherently parallel, our algorithmic view of it is basically sequential. This is due in part to 300 years of sequential mathematics and more than 20 years of sequential Fortran programming." [9]

Tree search occupies a fundamental place in artificial intelligence research [14]. In particular, the majority of game playing programs generate and search game trees. To date all game playing programs have sequentially searched the trees they generated. With the advent of concurrent processing technology, however, this state of affairs is very likely to change.

Two papers have recently addressed some of the problems associated with processing game trees in parallel. In [6] the emphasis is on developing a specific structure of microprocessors in a master-slave configuration to minimize the solution time of certain tree decision problems. One particular application is dealt with in the paper: that of two ply chess move generation. It is shown that, for the typical two ply move, the total real solution time decreases with the increasing number of slave microprocessors. A theoretical upper bound on the number of useful slaves for this particular problem is also derived.

In [3] the alpha-beta pruning algorithm is studied. It is argued that in a parallel implementation which explores different subtrees of the game tree concurrently and independently, the power of the algorithm would be lost. As an alternative, the paper suggests that the problem of finding a path in a game tree can be viewed as the problem of locating the root of a monotonic function over some interval. A parallel implementation of the algorithm is thus proposed in which the processes work independently by searching the entire game tree for the solution over disjoint subintervals. Here the efficiency of the alpha-beta pruning algorithm is measured by the average number of terminal nodes explored during the search. The main result is that when the degree k of parallelism, i.e. the number of processes cooperating in the search, is small (k=2 or 3), the parallel algorithm shows an improvement over the original algorithm by a factor which is larger than k. The maximum speed-up achievable, however, is believed to be limited to 5 or 6. The author concludes by suggesting that a better way to implement the alpha-beta pruning algorithm with a large number of processes in parallel would be to combine both his strategy of decomposition and the independent exploration of different subtrees of the entire game tree.

The present paper describes preliminary experiments with a program that simulates concurrency to search game trees in parallel. As our model of computation we use a parallel computer of the MIMD (Multiple Instruction stream Multiple Data stream) type as defined in [8]. The machine we intend has a number of asynchronous processors with a communication facility provided by common memory or communication lines. A processor can initiate another processor, send a message to another processor, or wait for a message from another processor. Apart from these interactions, processors proceed independently.

The simulated environment provides multiple software processes and multiple

This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grants NSERC-A3336 and A3014.

hardware processors. A process is created for each node that is searched. The number of processors is a parameter of the program.

Unlike the algorithm of [3], our implementation of the alpha-beta algorithm searches disjoint subtrees concurrently. It uses knowledge about the behavior of the sequential algorithm to simultaneously increase the number of cutoffs and reduce the search time. The major experimental finding reported in this paper is that this approach results in a considerable speedup, thereby contradicting the claims in [3].

We assume the reader is familiar with the sequential alpha-beta algorithm and the associated tree search jargon insofar as the terminology is unified [11,13,14]. We will essentially use the nomenclature used in [13]. As usual, the trees to be searched are those of a two-person, zero-sum, non-chance, board game of perfect information [10]. In Section 2 we outline the basic principles used in the design of our parallel implementation of the alpha-beta algorithm. The algorithm itself together with the procedures it calls are described in Section 3. In Section 4 we present and analyse the results of our experiments.

## 2. Basic Design Principles

It is shown in [16] that the minimum number of terminal nodes scored by the alpha-beta tree search algorithm under the best circumstances is

$$B^{\lceil D/2 \rceil} + B^{\lfloor D/2 \rfloor} - 1$$

where B is the branch factor and D is the maximum depth of the (uniform) tree. This is illustrated in Fig. 1 where the tree is perfectly ordered so that the best moves for both players are always to the left.In such a tree, the terminal nodes shown with a score (and only these terminal nodes) must be examined in order to determine the principal continuation (PC), i.e. the best sequence of moves found for both sides to follow based on the computer's finite depth search [2].

This fact represents the basis for our parallel implementation of the alpha-beta algorithm: assuming that the tree to be searched is perfectly ordered, those nodes that have to be scored are (concurrently) visited first. The algorithm is designed with two objectives in mind: to minimize the run time of the search and to perform as many cutoffs as possible, thereby minimizing the cost of the search (total number of operations).

In order to achieve these goals a distinction is made among the sons of a node. The first son of a node is called the "left son". The subtree containing the left son is called the "left subtree" and the process that searches this subtree is the "left process". All other sons of a node are called "right sons" and are contained in "right subtrees" which are searched by "right processes". This is illustrated in Fig. 2.

The left subtree of a node is searched by a left process (which is spawned by the parent node) until a final value for the left son is backed up to the parent node. To obtain this final value, the left son's process spawns processes (lefts and rights) to search all of the left son's subtrees. Concurrently, a single, temporary value is obtained for each of the right sons of the parent node. These values are then compared to the final value of the left son and cutoffs are made where appropriate.

The temporary value for a right son is obtained by the right son's process spawning a process to search its left subtree. This new process searches the subtree, backs-up a value to the parent's right son, and then dies. If after a cutoff check the right subtree search continues, then a process is generated to search the second subtree of the right son. This procedure continues until either the subtree is exhaustively searched or the search is cut off.

It is clear that, by applying the above method, those nodes that must be examined by the alpha-beta algorithm will be visited first. This ensures that needless work is not done; a cutoff check is performed before processes are generated to search subtrees that may be cut off.

In a search with more processors than running processes it may be possible to minimize the run time of the search by generating processes to search the sons of a right node concurrently using the idle processors. This brute force approach is not used since it conflicts with the other aim of our design, namely minimizing the cost of the search. The cost of any tree search consists mainly of the cost of updating the system in moving from parent to son and in the cost of evaluating or scoring a node. Therefore even though a processor (which could be doing concurrent work) is idle, the overall cost in operations is minimized by not searching subtrees which may not have to be searched.

In Fig. 3 (assuming an infinite number of processors running at the same speed) the root node process generates processes

P1, P2, and P3 which execute concurrently. P1 being a left process generates processes P1.1, P1.2, and P1.3 to search all of the subtrees of the left son of the root. P2 and P3 are searching right subtrees and therefore generate only processes to search the left subtrees of the right sons of the root (P2.1 and P3.1 respectively). After generating its son processes, a parent process suspends itself and waits for its children to terminate, backup a value, and die.

At this point an interesting question arises. We stated earlier that one of our design objectives is to increase the number of cutoffs. How do we expect our algorithm to perform in this respect compared with the sequential version? To answer this question we shall make a distinction between shallow and deep cutoffs.

1) Shallow Cutoffs

(i) All shallow cutoffs that would occur in a sequential search due to the (temporary) value backed up to a node from its left son also occur in a parallel search. This is because all temporary values obtained for the right sons of the node are compared to the backed up final value of the left son for a cutoff check before the right subtree search continues. An example illustrating this situation is shown in Fig. 4. Initially, the root is assigned (temporarily) the final value of its left son, i.e. 8. The two right subtrees are searched in parallel, resulting in temporary values of 3 and 5 being assigned to the first and second right sons respectively. Clearly the circled sections of the two right subtrees are cut off in exactly the same way as in a sequential search.

A right subtree that is exhaustively searched and not cut off compares its final value to the temporary value of the parent and changes the parent's value if necessary. Any cutoff that would have occurred in other right subtrees due to the value backed up to the parent from its left son will also occur due to any right son value that changes the parent's value.

(ii) Some shallow cutoffs that would occur in a sequential search can be missed in the parallel search due to the way in which processes are generated to search game subtrees. In the example of Fig. 5 a sequential search would cut off the circled portion of the tree whereas the parallel search would not. The parallel search misses the cutoff since a process is generated to search that subtree before the first right subtree of the root completes its search and updates the root's value.

(iii) Some cutoffs that are missed in the sequential search may occur in the parallel search due to the way in which processes are generated. A right subtree search that terminates early and causes a change in the parent's value, may cause cutoffs in other right subtrees that would not occur in the sequential search, as in Fig. 6.

In Fig. 6 both right sons of the root compare their initial values, 6 and 7 respectively, to the final value of the left son, i.e. 5. Neither right subtree search is cut off so processes are generated to search the second sons of the right sons of the root. But since the second right son of the root has only one son, its subtree has already been exhaustively searched and, therefore, the root's value is updated to 7. Thus when the first right son of the root performs a cutoff check, this time a cutoff occurs. This cutoff is missed by the sequential search.

2) Deep Cutoffs

In order for deep cutoffs to occur at a node, values from searches of other parts of the tree must be available. In a sequential search the values or scores at each ply are known to every node and are stored in a single global score table. In a parallel search this is impossible since the best sequence of moves (found so far) from the root to a leaf is not always returned.

In Fig. 7 the left son of the right son of the root is searched at the same time as the son of the left son of the root. If the right son is backed up the value 3 at ply 1, and then the left son is backed up the value 1 (overwriting the score table value of 3 at ply 1), then when the second subtree of the right son is searched its value of 2 will not be recorded at ply 1 (since $1<2$ and we are minimizing at ply 1). Therefore the value of 2 will not be backed up to the root as it would be in the sequential search. This means that instead of returning the best sequence of moves (m2,m5) from the root, the sequence, m1,m3 will be returned.

Since there can be no global score table, an individual score table is assigned to each node when a process is generated to search the subtree containing that node. This table is initialized to the values in the score table of the node's parent. Therefore, the information necessary for a deep cutoff to occur is not available in general, see Fig. 8.

In practice, a node is not given a

complete score table, but rather just a small table containing the scores for the two previous plies and the node itself. This means that the complete score table for a node (as described above) is actually distributed throughout the tree along the path from the root to the node. With this structure it would be possible to obtain additional deep cutoffs as follows. Suppose that during a search of the tree in Fig. 8 (b) the following sequence occurs:

(i) the search of the left subtree (of the root) begins,

(ii) the search of the right subtree begins, and

(iii) the search of the left subtree completes, backing up a temporary score to the root.

At this point searching along some paths in the right subtree could be cut off, the information indicating this being available in the score table of the root node. However, to effect this deep cutoff, the information must be propagated down the right subtree. The algorithm could be extended to deal with this circumstance but as "deep cutoffs have only a second order effect on the average behavior of the alpha-beta pruning algorithm" [3], we have avoided the additional psychological complexity and administrative overhead.

This Section has described some basic principles that were considered in the design of the parallel adaptation of the alpha-beta algorithm. The algorithm itself is presented in the next Section.

## 3. The Algorithm

There are seven main components of the algorithm: Initialize, Handle, Score, Generate, GenerateMoves, Apply, and Update.

1) Initialize reads in the original board position (i.e., the configuration for the root node of the search tree) and the depth to which the tree will be searched. Handle is then invoked to create a process for the root.

2) Handle is a recursively-defined process. It searches a node in a game tree by calling either Score (for a leaf) or Generate (for a non-leaf) and then calling Update.

3) Score is a static evaluation function. It accepts as parameter a board configuration, and returns an integer representing the value of the position.

4) Generate is called to search a subtree that is not a leaf. It calls GenerateMoves to produce a list of moves from the current position. If the root of the subtree is a left node, then Handle is invoked once for each son. The processes thus created run concurrently, and Generate waits until they all terminate. If the root of the subtree to be searched is a right node, then the sons are searched in sequence by calling Handle for one of them, waiting for it to complete, and performing a cutoff check before searching the next son. Procedure Apply is used to produce board configurations for sons.

5) GenerateMoves accepts as parameter a board configuration and produces all of the legal moves from that position.

6) Apply accepts as parameters a board configuration and a move, and produces the board configuration that results when the move is made.

7) Update waits until the parent's score table is free and then copies the value derived as a score for the current subtree into the table, if applicable.

This algorithm has been implemented in order to empirically investigate its behavior. Experiments with the implemented algorithm yield statistics on the cost of a tree search in terms of the total run time, the number of nodes scored, and the number of nodes visited. By varying the number of processors used to search the tree we can investigate the value of parallelism in the algorithm. This experimental procedure can be applied to a number of search trees.

Since we did not have a multiprocessor available on which to implement our algorithm, it was necessary to simulate physical parallelism. The simulation language GASP IV [15] was chosen as a vehicle for this because it would easily simulate a variable number of processors and because it facilitates writing complex transactions (in our case: processes to search a tree node) in a high level language (FORTRAN). Additional details can be found in [1].

## 4. Experiments and Conclusions

> "I think we're coming more and more to the frontier where what artificial intelligence is waiting for is the mechanization in hardware of the sorts of parallel interaction which will allow an order of magnitude increase in intelligent behavior with a given outlay of cost." [12]

The implemented parallel alpha-beta algorithm, executing in the simulated multiprocess-multiprocessor environment, was tested for various inputs. The purpose

of our experiments was to study the effects of parallelism on the cost of a tree search, this cost being expressed in three different forms: (i) run time of the tree search, (ii) number of terminal nodes scored and (iii) total number of non-terminal and terminal nodes visited.

Essentially, a uniform tree of a given depth, D, and branch factor, BF, was generated and stored prior to the search. (Note that this is just for experimental purposes; in a typical decision tree search, the tree is generated as it is searched and only the relevant parts of the tree are stored.) The terminal nodes of this tree were assigned scores chosen from a particular probability distribution. The principal continuation was then sought using a varying number of processors and the three measures of cost recorded each time. These values were averaged over several trees with different terminal node scores but keeping the depth, branch factor and probability distribution of the scores fixed. The entire procedure was then repeated by altering these latter characteristics. Typical results of these experiments are shown in Figs. 9-12.

The curves in Fig. 9 show that the run time decreases sharply with an increasing number of processors doing the search. For a given depth of search the savings in run time are more noticeable, however, for large branch factors than for small ones. It is also clear that a saturation point is eventually reached after which the run time remains constant for an increasing number of processors. As expected, the total number of nodes visited as well as the number of terminal nodes scored also increases with an increasing number of processors. Surprisingly, in this case the increment is relatively small and a saturation point is quickly reached as can be seen in Figs. 10 and 11. Finally, the curves in Fig. 12 indicate that the algorithm exhibits the same behavior for various distributions of the terminal node scores.

A few remarks regarding future work are now in order. The experimental results presented above indicate that parallelism is of value in alpha-beta searching. Some modifications are possible in our algorithm. For example, minor changes in the strategy for generating processes to search a subtree, or in process priority assignments might make significant differences in performance. It is also worthwhile investigating the value of deep cutoffs in parallel searching. Because of the different order of node evaluation introduced by parallelism, deep cutoffs may become more than a second order effect. The simulation environment is a natural one in which to investigate these issues. Changes in either the algorithm or the hardware configuration can be implemented and experimented with easily. We believe that the next logical step would be to incorporate the parallel alpha-beta algorithm in an actual game playing program and study its behavior. It will then be possible to address more pragmatically the issues pertaining to the saturation points mentioned above -- in particular how they are related to the depth of search and branch factor. More importantly, it will be possible to determine how the quality of the game played by a game-playing program is affected by parallel tree search. Naturally, one may predict that with the faster processing of trees provided by parallelism, it will be possible to search deeper trees in a given amount of time and hence improve the quality of the play. This is not certain, however. In fact the artificial intelligence community seems to be divided on this matter. In [5] Neil Charness asserts that

> "Adding a few plies does not produce tremendous change in performance. What is gained by adding more lookahead capacity is rapidly lost in time spent evaluating the hundreds of thousands of new terminal positions... Humans have two major advantages over computers. They have both a vast knowledge base about chess and a set of procedures for efficient manipulation of that base".

These statements are to be contrasted with what Hans Berliner [4] recently wrote in an article reviewing the computer chess state-of-the-art:

> "Strangely enough (from my point of view and I believe AI's in general) the breakthrough has come on speed rather than knowledge. From this I must conclude that human chess players largely delude themselves in believing that chess is a 'conceptual' game. Apparently a large part of chess can be solved by exhaustive searching (as done in CHESS 4.6) and it remains to be seen whether such an approach will ultimately allow a machine to become World Champion by taking advantage of small inaccuracies in human play to win a material advantage and then hold on through the end game (where conceptualization still appears to be needed) to win anyway. I believe this will not happen, but such machines may come very, very close. It is already clear, however, that a

full width search going to a depth of six plies plus quiescence will discover things that even a Grandmaster will overlook on occasion."

We hope that our work will contribute towards the resolution of this controversial point. (Similar issues are discussed in [7]).

In conclusion we point out that searching trees in parallel, in our opinion, not only provides a considerable increase in speed, but in addition could allow us to naturally simulate one of the several aspects of human game playing, namely perception. Indeed -- taking the game of chess again as an example -- a generally accepted theory for Grandmaster play asserts that a highly sophisticated pattern recognition ability is used by the skilled human player while analyzing a chess position in order to choose the next move. As Charness writes: "The process of choosing a move seems to involve perception as a primary component, and in particular, the recognition of thousands of stored patterns" [5]. Whether this pattern recognition ability (simulated using parallelism or otherwise emulated) will be essential in the attainment of high-level game-playing by computers is yet another open question.

Acknowledgement

Thanks to Bob Tennent and Glenn MacEwen for reading a preliminary version of this manuscript.

References

[1] Akl, S.G., Barnard, D.T., and Doran, R.J., Searching game trees in parallel, Technical Report 79-87, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, November 1979.

[2] Akl, S.G., and Newborn, M.M., The principal continuation and the killer heuristic, Proceedings of the ACM Annual Conference, 1977, Seattle, Washington, pp. 466-473.

[3] Baudet, C.M., The design and analysis of algorithms for asynchronous multiprocessors, Technical Report CMU-CS-78-116, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1978.

[4] Berliner, H.J., A chronology of computer chess and its literature, Artificial Intelligence, Vol. 10, 1978, pp. 201-214.

[5] Charness, N., Human chess skill, in: Frey, P., Ed., Chess Skill in Man and Machine, Springer-Verlag, New York, 1977.

[6] Coraor, L.D., and Robinson, J.P., Using parallel microprocessors in tree decision problems, Proceedings of the International Symposium on Mini and Micro Computers, IEEE, Toronto, Nov. 8-11, 1976, pp. 51-55.

[7] Fishburn, J.P., Finkel, R.A., and Lawless, S.A., Two papers on alpha-beta search, Computer Science Technical Report 375, University of Wisconsin-Madison, December 1979.

[8] Flynn, M.J., Very high-speed computing systems, Proceedings of the IEEE, Vol. 54, No. 12, December 1966, pp. 1901-1909.

[9] Goodman, S.E., and Hedetniemi, S.T., Introduction to the Design and Analysis of Algorithms, McGraw-Hill, New York, 1977.

[10] Jackson, P.C., Introduction to Artificial Intelligence, Mason and Lipscomb, New York, 1974.

[11] Knuth, D.E., and Moore, R.W., An analysis of alpha-beta pruning, Artificial Intelligence, Vol. 6, No. 9, 1975, pp. 293-326.

[12] MacKay, D., in: McCorduck, P., Machines Who Think, W.H. Freeman and Co., San Francisco, 1979, page 80.

[13] Newborn, M.M., The efficiency of the alpha-beta search on trees with branch-dependent terminal node scores, Artificial Intelligence, Vol. 8, No. 2, 1977, pp. 137-153.

[14] Nilsson, N.J., Principles of Artificial Intelligence, Tioga Publishing Co., Palo Alto, California, 1980.

[15] Pritsker, A.A.B., The GASP IV Simulation Language, John Wiley & Sons, Toronto, 1974.

[16] Slagle, J.R., and Dixon, J.K., Experiments with some programs that search game trees, JACM, Vol. 16, No. 2, 1969, pp. 198-207.

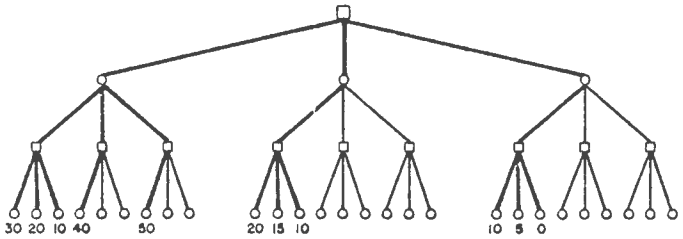



Fig. 1

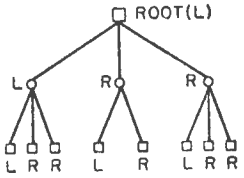



Fig. 2

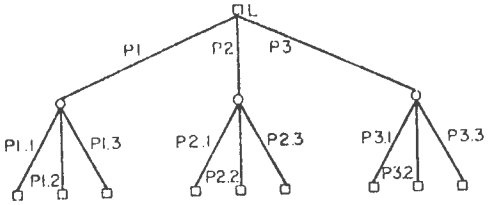



Fig. 3

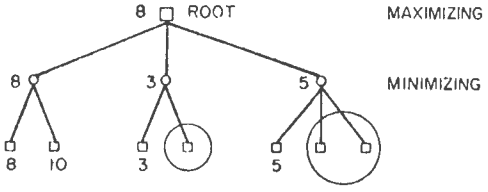



Fig. 4

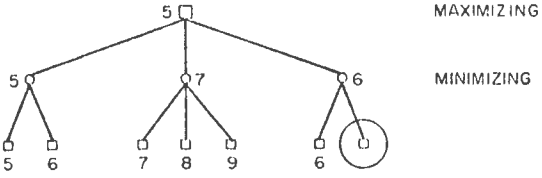



Fig. 5

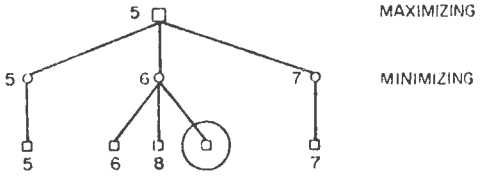



Fig. 6

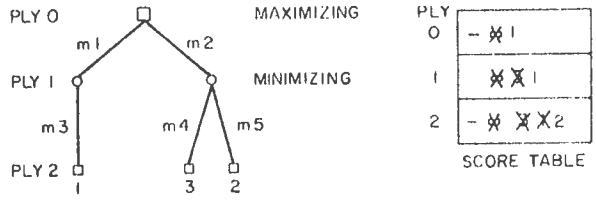



Fig. 7

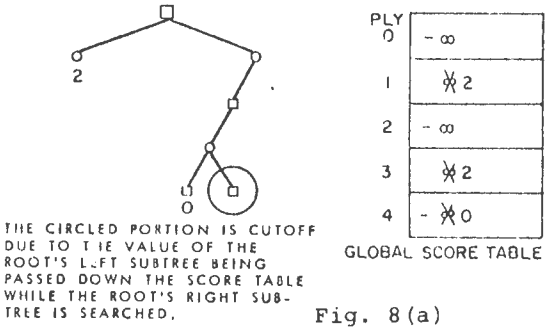



Fig. 8(a)

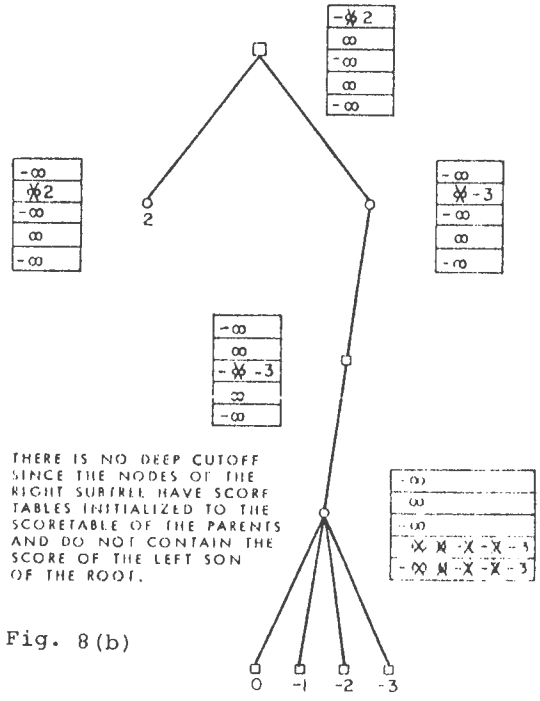



Fig. 8(b)

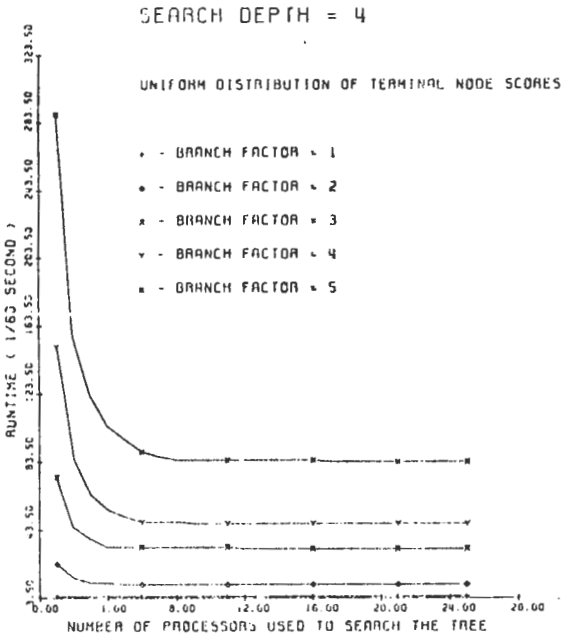



Fig. 9

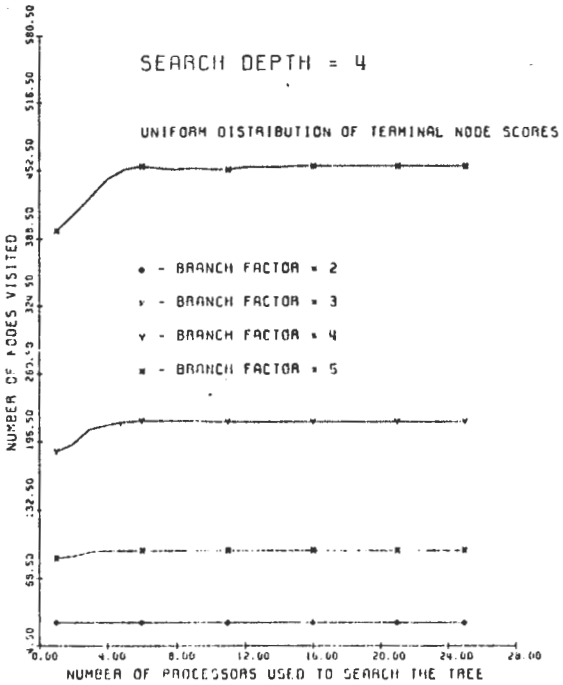



Fig. 11

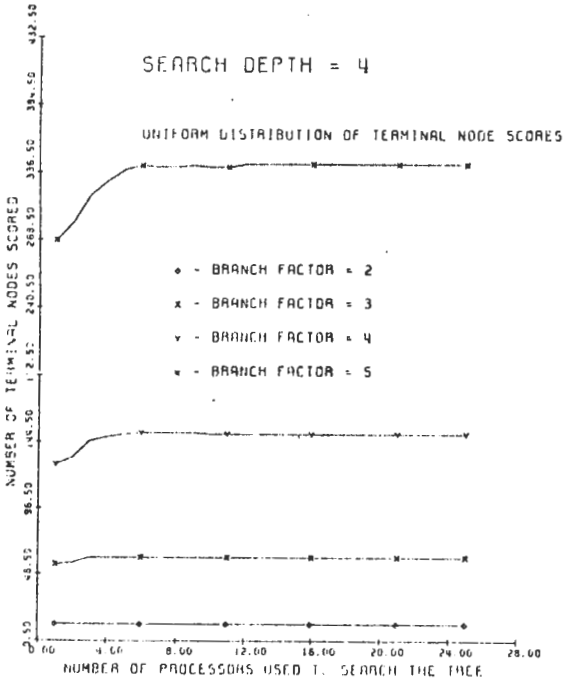



Fig. 10

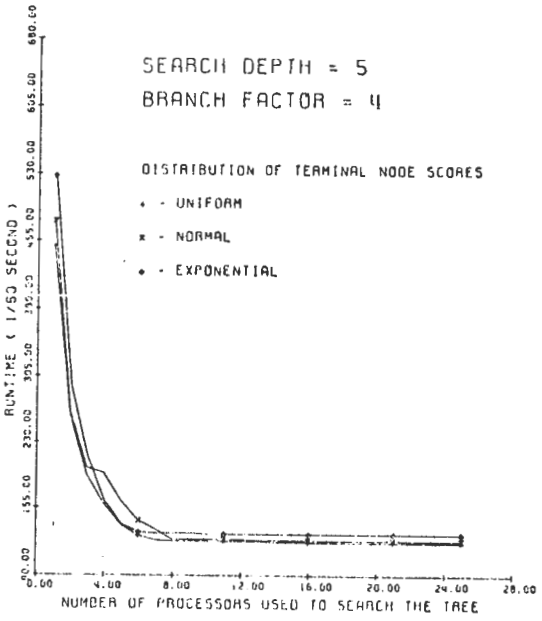



Fig. 12

# APPLICATIONS OF THE CONTRACT NET FRAMEWORK: SEARCH

Reid G. Smith

Defence Research Establishment Atlantic
Box 1012
Dartmouth, Nova Scotia, B2Y 3Z7, Canada.

## Abstract[1]

We discuss the implementation of heuristic search algorithms in a distributed problem solver whose processors interact according to the contract net protocol. Task distribution is viewed as a local mutual selection process based on a two-way transfer of information between processors with tasks to be executed and processors with knowledge-sources capable of executing those tasks.

As an example of the approach, we consider the N Queens problem. We then derive measures of the speedup that can be expected from the application of a distributed processor to search problems that involve regular trees, and discuss the effect of coupling between processors on speedup. Bounds are developed for the number of processors that are required to achieve maximum speedup.

## 1 Introduction

Distributed problem solving is the cooperative solution of problems by a decentralized and loosely coupled collection of knowledge-sources (KSs), each of which may reside in a distinct processor node. The KSs cooperate by sharing tasks and/or results. By decentralized we mean that both control and data are logically and often geographically distributed; there is neither global control nor global data storage. Loosely coupled means that individual KSs spend most of their time in computation rather than communication. Such problem solvers offer the promise of speed, reliability, extensibility, the ability to handle applications with a natural spatial distribution, and the ability to tolerate uncertain data and knowledge.

Search problems are attractive as applications of distributed problem solving for three major reasons. First, exploration of a search space of the size commonly encountered in AI applications consumes a large amount of computing time (see, for example, discussions of Meta-Dendral [Buchanan, 1978], and CONGEN [Carhart, 1976]). Thus, the speedup promised by the distributed approach is attractive. Second, search problems are often modular in form. Numerous relatively independent subtasks are created during the course of a search. These subtasks are ideal candidates for distribution to individual processors. Finally, search is one of the major problem-solving paradigms. It is therefore important to develop tools for applying the new VLSI technology to search problems.

## 2 Task-Sharing And Contract Negotiation

The contract net protocol [Smith, 1978], [Smith, 1979] facilitates cooperation of multiple processors in the solution of a problem. Dynamic matching of tasks and KSs is effected by negotiation. A contract is an explicit agreement between a processor that generates a task (the manager) and a processor willing to execute the task (the contractor). (Note that a processor is assumed to contain one or more KSs.) A contract is normally established by a process of local mutual selection based on a two-way transfer of information. In brief, available contractors evaluate task announcements made by several managers until they find one of interest. They submit a bid for that task. The manager then evaluates the bids received from potential contractors and selects the one it determines to be most appropriate. Both parties to the agreement have evaluated the information supplied by the other and a mutual selection has been made. Control is distributed because processing and communication are not focussed at particular processors, but rather every processor is capable of accepting and assigning tasks.

Contract net messages contain slots for information that aids negotiation. A task announcement contains three such slots. The **eligibility specification** is a list of criteria that a processor must meet to be eligible to submit a bid. It enables a processor receiving the message to decide whether or not it is able to execute the task. This specification reduces message traffic by pruning processors whose bids would be clearly unacceptable. The **task abstraction** is a brief description of the task to be executed. It enables a processor to rank the announced task relative to other announced tasks. An abstraction is used rather than a complete description in order to reduce the length of the message. The **bid specification** is a description of the expected form of a bid. It enables a processor to include in a bid only the information about its

[1] This work was supported in part by the Advanced Research Projects Agency under contract MDA 903-77-C-0322, and the National Science Foundation under contract MCS 77-02712. Some of the work described is being pursued in collaboration with Randall Davis at MIT. Joe Maksym also made a number of valuable comments.

capabilities that is relevant to the task rather than a complete description (called a node abstraction). This simplifies the task of the manager in evaluating bids and further reduces message traffic.

## 3 Distributed Search: Overview

In this section we discuss some general characteristics of distributed search. We then show how the contract net protocol can be used to organize a distributed problem solver to perform such a search.

Consider the exhaustive search of a tree in a distributed processor. To make clear the flow of the search, we make the following assumptions: (i) the basic task for each processor is generation of a successor node in the tree, (ii) as soon as a processor generates a node, it distributes that node to another processor for further expansion, (iii) generation of each node requires a constant processing time, (iv) there is a sufficient number of processors so that the expansion of a node can be commenced by one processor as soon as the node has been generated by another, (v) a processor can distribute a successor node to another processor concurrently with generation of another successor node, and (vi) distribution of a node to a processor and reporting of results require a negligible amount of time compared to the time required to expand the node.

The flow of the search process is shown in Figure 1 for a regular tree of branching factor 2 and depth 3. The numbers inside each node circle indicate the time unit at which the node was generated and the processor that generated it (in the format "time / processor").

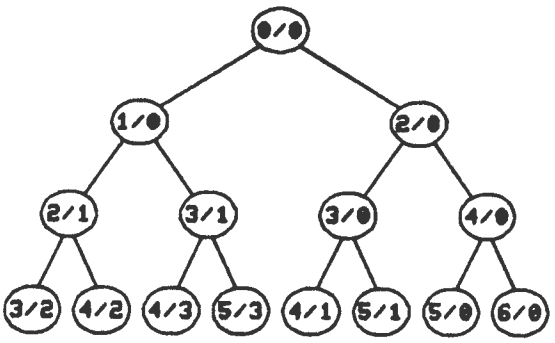



**Figure 1. Distributed Search Of A Regular Tree.**

At time 1, one successor of the root node is generated. This successor is distributed (we assume instantaneously) to another processor, so that at time 2, two successors are generated. The number of processors involved in the search rises from 1 to 4 and then decreases again to 1 before completion.

It is apparent that problems that entail a large amount of search are especially amenable to a distributed approach--they have the potential for large speedups. In addition, trees comprised of OR nodes lend themselves more readily to concurrent exploration than do those comprised of AND nodes. This is because less processor synchronization is required for their exploration. Trees with Ordered-AND nodes (i.e., nodes that must be expanded in a particular order) are the least amenable to concurrent exploration because they require the greatest amount of synchronization (which inevitably means that some processors will stand idle waiting for results to be generated by other processors).

In Appendix A we present performance measures for exhaustive distributed search of regular trees. It is shown that speedups that are close to linear in the number of processors are possible. It has been shown elsewhere (e.g., [Imai, 1979]) that better than linear speedups are possible. This result follows for problems in which application of multiple processors can eliminate fruitless expansion of a large number of nodes. We also see from the analysis that trees that are bushy near the root lead to larger speedups than trees in which bushiness occurs at larger depths. This is due to the fact that more processors get involved quickly, and the nodes they generate can often be queued for later expansion with no increase in search time (because of the decreased demand for processors as the search nears completion). Finally, it is shown that loose-coupling must be maintained if maximum speedups are to be achieved.

### 3.1 Node Selection

In a distributed search, selection of nodes for expansion and generation of their successors are asynchronous, local processes. Node selection is especially different from the uniprocessor case, where a global evaluation function is used to select one node to be expanded next. Distributed search strategies have a local character because many nodes may be selected concurrently for expansion by individual processors, usually based on a more local evaluation.

If interprocessor communication is severely constrained, then as a processor generates new nodes, it queues them locally for expansion and processes them alone as soon as it can (in an order dependent on its search strategy). Only when a processor is idle and has no nodes queued for expansion does it communicate with other processors to acquire new nodes. We call this local queuing. The result is a local approximation to one of the familiar uniprocessor search strategies.

It is possible to impose a global search strategy on a distributed processor by transmitting all nodes ready for expansion to a central repository. A global evaluation function can then order the nodes, and idle processors can remove them (in order) from the repository. We call this global queuing. Unfortunately, it can lead to bottleneck and reliability problems. In addition, when communication costs are high, it can lead to lower speedups than local queuing (see Appendix A). The main advantage of global queuing is that it

offers the potential for ensuring that the best nodes are expanded first because the evaluation function has a global perspective. When local queuing is used, other measures must be taken to approximate a global perspective. This is an example of the general problem of achieving coherent behavior in a system that uses distributed control. Distributed control is necessary if the advantages of distributed problem solving are to be achieved--but it leads directly to a problem in maintaining global coherence.

Better approximations to global strategies are obtained at the price of interprocessor communication. The intent of a best-first search in a uniprocessor, for example, is to select the most appropriate node for expansion at any given time. If interprocessor communication is severely constrained, then an individual processor can only select the best of the nodes that it has stored locally; and none of these nodes may be the overall best node to be expanded. If the processors can communicate more extensively with each other, then several of the overall best nodes can be concurrently selected for expansion by separate idle processors. We will see how this is done with the contract net protocol in the next section.

## 4 Example: The N Queens Problem

The goal of the N Queens problem is to place N queens on an N x N chessboard in such a way that no two are on the same row, column, or diagonal. We discuss one possible implementation of this problem as a simple introductory example of the issues that arise in an application of distributed problem solving. Section 4.1 shows sample messages transmitted by processors during the solution of the problem.

The processor at which the problem is started (the top-level processor) begins with an empty board. It generates N subtasks, each of which corresponds to a partial board with 1 queen in the first column and in a different row for each subtask. These subtasks are announced. Bids are submitted by other idle processors. Successful bidders are awarded contracts for the task of extending the partial boards to completion. The top-level processor is the manager for this task. (It is now free to become a contractor for future subtasks.)

This process is continued recursively for each column of the board; that is, the contractors trying to extend partial boards (here, with 1 queen already placed) generate independent subtasks by placing a queen in the next column (here, the second column) under the no-capture constraint. They then distribute the subtasks (and take on the role of manager for them).

There is thus only one type of task for all processors--extension of a partial board. When a processor node places the $N^{th}$ queen, and thus has a complete solution to the problem, it reports to its manager. (Similarly, when a processor cannot further extend a partial board, it reports to its manager.) Further reports ripple upward to the top level and the search terminates when some pre-specified number of solutions has been compiled; that is, when any manager has received the required number of solutions, it terminates any outstanding subtasks within its span of control and reports to its own manager. This manager in turn terminates outstanding subtasks, and so on. Ultimately, the top-level node reports the solutions to the user.

The task abstraction of each task announcement specifies the type of task to be executed and the present state of the task, relative to the goal state (in this case, the number of queens that have already been placed on the partial board). The number of queens placed gives a potential contractor a method for ranking announced tasks in order to select a task for submission of a bid. It is used by processors in this example to effect an approximation to the desired global search strategy. A breadth-first strategy, for example, is implemented by ranking boards that have a small number of queens placed higher than those that have a larger number of queens placed. Bids are submitted first for these boards, and they are therefore generally executed before the others.[2]

We pointed out earlier that one of the problems associated with distributed control is approximation of the global perspective that enables a uniprocessor to select the best nodes for expansion at any time. This problem is handled in a contract net as follows: Each processor listens to all task announcements and maintains a list of recent announcements. When a processor goes idle, it selects, according to its own criteria, the current optimum task for which to submit a bid from among the tasks contained in its list. Each processor therefore has a kind of window through which to view the currently available tasks. This window lends a more global character to the search strategy because node selection is based on information received from a number of processors. The cost is local storage (for the list of tasks) and communication (to gain information about tasks available from other processors).

Two possible eligibility specifications are shown. The first is a null specification. The assumption here is that all processors have the necessary procedures for executing the extend-board task. A bid then simply indicates that a processor is willing to execute the announced task, and the contract is awarded to the first bidder. In the second case, the eligibility specification names the required procedures. The assumption here is that not all processors are pre-loaded with the necessary procedures. A potential contractor can submit a bid indicating that it needs the procedures to execute the task. In this case the contract is awarded to the first processor that has the procedures, or, in the absence of any such

2 It is of course possible to execute different search strategies at different processors.

bidders, to the first bidder. This is an example of dynamic transfer of knowledge. (See [Smith, 1978] for a more extensive discussion.) A simple award strategy (i.e., award to the first bidder) is possible for this problem because any processor with the procedures has the capability to execute the task.

### 4.1 Sample Messages

<The processor given responsibility for the top-level task issues messages of the following form as it generates the first subtasks.>

```
To: *          <"*" indicates a broadcast message.>
From: 1
Type: TASK ANNOUNCEMENT
Contract: 1

Task Abstraction:
    TASK TYPE EXTEND-BOARD
    BOARD QUEENS 1

Eligibility Specification:
    NIL <or> PROCEDURE NAME EXTEND-BOARD

Bid Specification:
    NIL

-----

To: 1                  <Idle processors respond.>
From: i
Type: BID
Contract: 1

Node Abstraction:
    NIL <or> REQUIRE PROCEDURE NAME EXTEND-BOARD

-----

To: i                  <To the successful bidder.>
From: 1
Type: AWARD
Contract: 1

Task Specification:
    BOARD SPECIFICATION (...)
    PROCEDURE NAME EXTEND-BOARD CODE (...)
                                   <If required.>

-----

To: k                  <Eventually, messages like
From: q                    this are transmitted.>
Type: REPORT
Contract: j

Result Description:
    SUCCESS
    BOARD SPECIFICATION (...)
<or>
    FAILURE

-----

To: n                  <When enough solutions have been
From: m                    accumulated by a manager, it
Type: TERMINATION          sends messages like this
Contract: i                to its contractors.>
```

## 5 Summary

We have shown the use of the contract net protocol in the solution of a search problem. The negotiation process is particularly simple for this problem and a minimal amount of information needs to be transferred between processors. Consequently, only a degree of the power of the approach is demonstrated. The main use of the protocol in this example is to make connections between processors for reliable distribution of the processing load and communication of results. Processors are efficiently used because they can take on multiple roles: A processor that has generated all 1-queen extensions to the current board and distributed them to other processors (contractors) need only deal with reports occasionally (in its role as manager). It is therefore free to act as a contractor for other tasks. The result is that no processors remain idle as long as there are tasks to be executed. Furthermore, processors are able to obtain the procedures necessary to execute tasks as part of the negotiation process. Finally, The explicit manager-contractor links assist in rapid local pruning of the search space (via termination messages) when a sufficient number of solutions has been found. Each manager can directly terminate the execution of subtasks being executed by its contractors as soon as it becomes aware that the results are no longer required. The net does not have to wait for reports to reach the top-level processor before subtasks are terminated.

We have demonstrated the utility of negotiation as an approach to the problem of maintaining global coherence in a system that uses distributed control. The problem is by no means solved, however, and is a focal point for further research. One of the extensions currently under examination is to have processors listen more carefully to the message traffic around them. At present, only task announcements are examined by all processors. It may prove beneficial for other messages (e.g., bids, awards, and reports) to be subjected to the same scrutiny. It may, for example, lead to more informed bid and award strategies.

## Appendix A

## Distributed Search Analysis

We derive bounds on the performance that can be expected from distributed search of regular trees. Although we only explicitly consider regular trees in this analysis, the results are easily extended to a more general class of trees--those that are compositions of regular trees. (See [Smith, 1978] for details.)

## A.1 Speedup

The total number of nodes, n, in a regular tree structure with depth d and branching factor b is,

$$n = (b^{d+1} - 1)/(b - 1). \quad (1)$$

The number of tip nodes, $n_t$, in such a tree is,

$$n_t = b^d. \quad (2)$$

The time required for the search is the most appropriate measure of performance for a distributed processor. The traditional uniprocessor measure of number of nodes examined is still a valid measure of the power of the search strategy, but is insufficient to capture the effect of multiple processors.

### A.1.1 Uniprocessor Search

The search time divides into two components: the time to expand a node, $t_e$, (i.e., the time required to generate all successor nodes of a node), and the time to select a new node for expansion, $t_s$. The time to expand a node can be rewritten in terms of the time required to generate a single successor node, $t_g$, as follows,

$$t_e = b \cdot t_g. \quad (3)$$

The minimum time to find one goal node in regular tree is achieved if the tree is searched in a depth-first fashion and no false paths are explored. Under this assumption, the uniprocessor search time, $t^u_{min}$, is,

$$t^u_{min} = ((d - 1) \cdot b + 1) \cdot t_g + (d - 1) \cdot t_s. \quad (4)$$

We have assumed that a node is completely expanded (i.e., all successor nodes are generated) before a new node is selected for expansion, and that the goal node can be recognized as soon as it is generated. We do not consider search strategies where only some of the successors of a node are generated before a new node is selected for expansion. (See [Smith, 1978] for treatment of this type of strategy.)

The maximum uniprocessor time, $t^u_{max}$, is achieved when exhaustive search of the tree must be performed before the goal node is found. In this case, the search time is,

$$t^u_{max} = (n - 1) \cdot t_g + (n - 1 - n_t) \cdot t_s. \quad (5)$$

### A.1.2 Distributed Processor Search

We assume the search strategy is as presented in Section 3; that is, a node is distributed for expansion by another processor as soon as it is generated; there is thus no time required for selection of nodes and the search time depends on the time to generate a successor node, $t_g$, and the time to distribute a node to another processor, $t_c$. We assume that the $t_c$ cost must be incurred any time the expansion of a node is started by a processor, even if the node was generated by that processor. This is the case if global queuing is used. It leads to a somewhat pessimistic estimate for search time (and therefore speedup) but simplifies the analysis. We will later drop this assumption.

The minimum time for a distributed processor to find a single node in a regular tree, $t^d_{min}$, is,

$$t^d_{min} = d \cdot t_g + (d - 1) \cdot t_c. \quad (6)$$

The maximum time, $t^d_{max}$, is given by,

$$t^d_{max} = d \cdot b \cdot t_g + (d - 1) \cdot t_c. \quad (7)$$

This is equivalent to the time required to expand the nodes that border the tree on one side.

### A.1.3 Comparison

The speedup for exhaustive, or maximum, search, $S_{maxe}$, is given by,

$$S_{maxe} = t^u_{max}/t^d_{max}. \quad (8)$$

Note that $S_{maxe}$ is not the maximum attainable speedup for a regular tree. It is, however, a convenient measure for comparison. We will later derive the address of the tip node for which the maximum speedup is attained.

Note also that as the selectivity of the search strategy is augmented, thus diminishing the need for exhaustive search, the advantage of concurrent computation is also diminished.

In order to draw some simple conclusions from the $S_{maxe}$ equation, we will assume that $t_s \ll t_g$. Under this assumption,

$$S_{maxe} \approx (n - 1)/(d \cdot b + (d - 1) \cdot (t_c/t_g)). \quad (9)$$

$t_c/t_g$ is a measure of the coupling between processor nodes for the distributed search problem. We call this ratio the processor-coupling-factor, $C_p$. (Note that it depends on both the characteristics of the task and the characteristics of the distributed processor.) Thus, rewriting (9), we have,

$$S_{maxe} \approx (n - 1)/(d \cdot b + (d - 1) \cdot C_p). \quad (10)$$

Figure A.1 shows the variation in $S_{maxe}$ as a function of $C_p$ for a regular tree of branching factor 3 and depth 6. The cost of a mismatch between the task grain size and the communications characteristics of the distributed processor is apparent: Loose-coupling must be ensured by a proper match of task grain size to distributed processor communications characteristics if a significant speedup is to be achieved.

Figure A.2 shows the maximum speedups attainable for exhaustive search of three regular trees, of branching factor 2, 3, and 6, as a

function of depth, under the assumption that $C_p = 0$. Also shown is $S_{min}$, obtained by comparing the minimum time for a distributed processor to search a regular tree, $t^d_{min}$, with the minimum time required for a uniprocessor, $t^u_{min}$, assuming that all successors of a node must be generated before a new node can be selected for expansion.

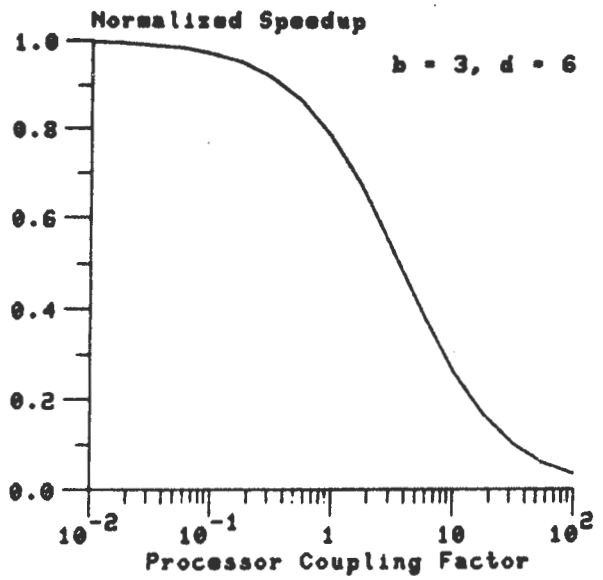



Figure A.1. Speedup And Coupling

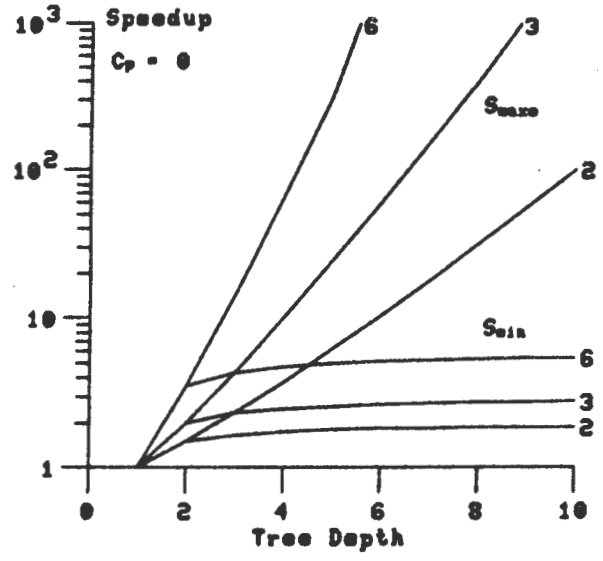



Figure A.2. Speedup For Three Regular Trees

## A.2 Processor Requirement

We derive lower and upper bounds on the number of processors, $P_{max}$, required in a distributed processor to obtain the maximum speedup for the exhaustive search of a regular tree. We assume that $t_c = 0$, and that $t_g = 1$.

As a lower bound, it is apparent that at least $S_{maxe}$ processors are required to achieve a speedup of $S_{maxe}$. This is overly optimistic because it assumes that all processors are fully utilized throughout the period of the computation. Near the start and end of the search, however, very few processors are in use.

In order to improve the estimate, consider the rate at which processors are pressed into service as the search progresses. The number of new tasks generated at each successive time unit in the search of a tree of infinite depth and branching factor b is given by the following generalized Fibonacci series of order b,

$$P^*_j = (P^*_{j-1} + P^*_{j-2} + \ldots + P^*_{j-b}). \qquad (11)$$

$$j = 1,2,3, \ldots$$

$$P^*_j = 0, \quad j < -1.$$

$$P^*_{-1}, P^*_0 = 1.$$

where the "*" superscript is used to indicate that the series is written for a tree of infinite depth. To account for the finite depth of the trees of interest, the equation can be modified as follows. Observe that whenever a processor reaches a tip node in the tree, the effect is to prune a subtree from the infinite tree. This pruning begins after d time units. We can account for the pruning of these subtrees by subtracting Fibonacci series, that start at times when processors reach tip nodes, from the original series. The number of series to be subtracted at each time instant corresponds to the number of tip nodes reached at that time instant. The number of tip nodes reached at each instant of time (starting at the $d^{th}$ time instant, when the first tip node is reached, to the $b \cdot d^{th}$ time instant, when the search is completed) is given by,

$$k_j = C_b(d, j-d). \qquad (12)$$

$$j = d, d+1, d+2, \ldots, b \cdot d.$$
$$k_j = 0, \quad j < d, \quad j > b \cdot d.$$

where $C_m(n, k)$ is the coefficient in the $n^{th}$ row and the $k^{th}$ column of the m-arithmetic triangle. In general, the $C_m(n, k)$ obey the equation,

$$C_m(n, k) = C_m(n-1, k) + C_m(n-1, k-1) + \quad (13)$$

$$\ldots + C_m(n-1, k-m+1).$$

$$0 \le k \le n\cdot(m-1), \quad n \ge 0.$$

$$C_m(0, 0) = 1.$$

$$C_m(1, k) = 1, \quad 0 \le k \le m-1.$$

$$C_m(1, k) = 0, \quad k \ge m.$$

Thus the number of processors required, $P_j$, is given by,

$$P_j = P^*_j - k_d\cdot P^*_{j-d} - k_{d+1}\cdot P^*_{j-(d+1)} - \quad (14)$$

$$\ldots - k_{b\cdot d}\cdot P^*_{j-(b\cdot d)}\cdot$$

$$j = 0,1,2, \ldots, b\cdot d.$$

And an upper bound on the number of processors required is given by,

$$P_{max} = MAX(P_j). \quad (15)$$

$$0 \le j \le b\cdot d.$$

This estimate is an upper bound because of the assumption that nodes cannot be queued for later expansion, but instead must be expanded as soon as they are generated. This is not generally required because of the lower demand for processors as the search nears completion.

Figure A.3 shows the two bounds for the required number of processors for exhaustive search of the same three trees as used in Figure A.2. Also shown (by dashed lines) is the actual number of processors required to achieve the maximum exhaustive search speedups (as determined by simulation).

Figure A.4 shows the increase in normalized speedup for a tree of branching factor 3 and depth 6 as the number of processors is varied. Also shown is corresponding decrease in efficiency (i.e., speedup per processor). This is a conservative estimate of efficiency, in that it includes processors that stand idle near the start and end of the search. These processors might be applied to another top-level problem during this time in a general-purpose distributed processor.

We noted earlier that the speedup estimates for distributed processor search are slightly pessimistic, because of the assumption that the cost $t_c$ is <u>always</u> incurred when a processor acquires a node for expansion. In Figure A.5, we show the effects of dropping this assumption. The figure compares the possible speedups for varying numbers of processors on a tree of depth 6 and branching factor 3 for both the global queuing of nodes to be expanded and the local queuing of such nodes. The speedups are normalized to that attainable with a global queuing strategy. A small number of sample points are marked with symbols to allow the reader to distinguish between the curves.

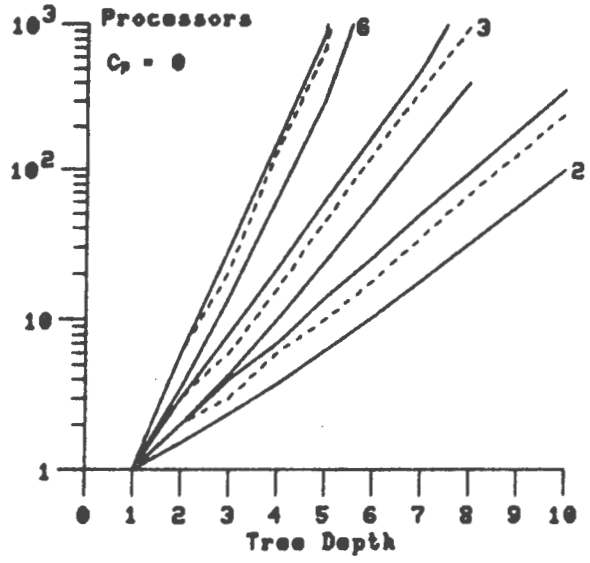



Figure A.3. Processor Requirement

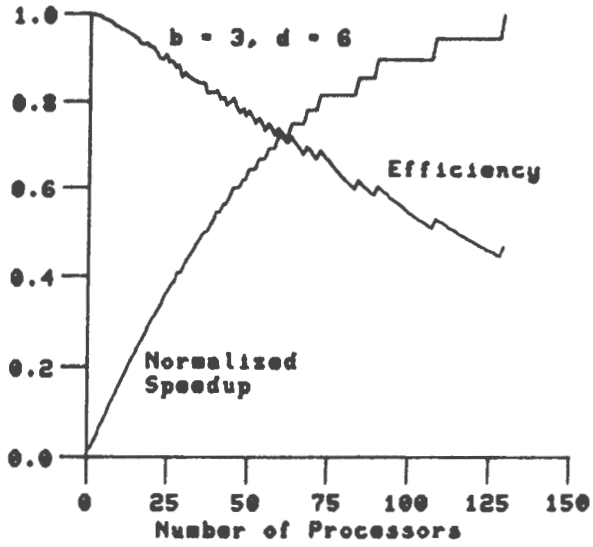



Figure A.4. Speedup And Efficiency

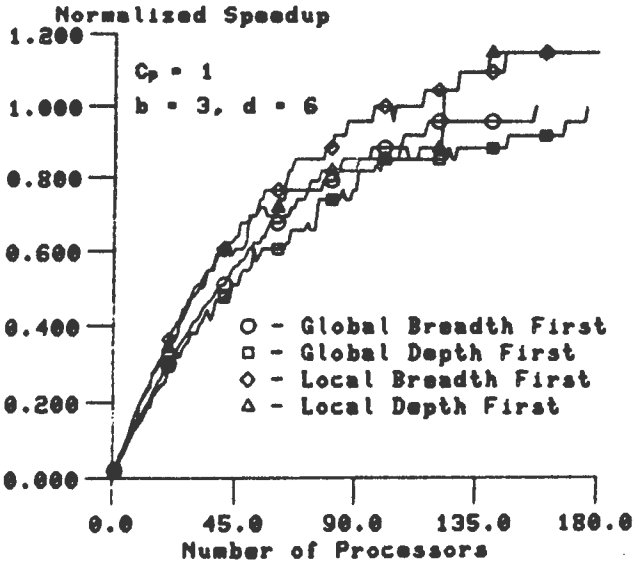



Figure A.5. Speedup And Queuing Strategy

Local queuing strategies are useful when $C_p$ is high. In Figure A.5, $C_p = 1$. We see a small improvement for local queuing in each case.

For further comparison, two selection strategies have been used for the figure: breadth-first and depth-first. We see that a breadth-first strategy leads to slightly better speedups than a depth-first strategy, mainly because tasks get distributed to idle processors earlier in the search.

## A.3 The Maximum Speedup

We now derive the address of the tip node at which the maximum speedup is attained. As in the previous section, we assume that $C_p = 0$, $t_c = 0$, and $t_g = 1$.

We can write the address of a tip node, $a_k$, as follows,

$$a_k = k + n - n_t. \qquad (16)$$
$$0 \leq k < n_t.$$

where k is the index of the tip node. $a_k$ is also the number of time units required by a uniprocessor to reach the tip node with that address, using a breadth-first search algorithm, under the above assumptions.

The number of time units, $t_k$, required by a distributed processor to reach the node with address $a_k$ is given by,

$$t_k = d + \sum_{j=0}^{d-1} w_j. \qquad (17)$$

where $w_j = \{0, 1, \ldots, b-1\}$, $0 \leq j \leq d-1$ are the values of the bits in the b-ary representation of the index, k.

Hence, the address of the node for which the maximum speedup is attained, $a_{smax}$, is the $a_k$ that maximizes $a_k/t_k$.

## References


[Buchanan, 1978]
B. G. Buchanan and T. M. Mitchell, Model-Directed Learning Of Production Rules. In D. A. Waterman and F. Hayes-Roth (Eds.), Pattern-Directed Inference Systems. New York: Academic Press, 1978. Pp.297-312.

[Carhart, 1976]
R. E. Carhart and D. H. Smith, Applications Of Artificial Intelligence For Chemical Inference XX. Intelligent Use Of Constraints In Computer-Assisted Structure Elucidation. Computers In Chemistry, Vol. 1, 1976, p. 79.

[Imai, 1979]
M. Imai, Y. Yoshida, and T. Fukumura, A Parallel Searching Scheme For Multiprocessor Systems And Its Application To Combinatorial Problems. IJCAI6, 1979, pp. 416-418.

[Smith, 1978]
R. G. Smith, A Framework For Problem Solving In A Distributed Processing Environment. Ph.D. Dissertation, STAN-CS-78-700 (HPP-78-28) Dept. of Computer Science, Stanford University, December 1978.

[Smith, 1979]
R. G. Smith, The Contract Net Protocol: High-Level Communication And Control In A Distributed Problem Solver. Proceedings of the First International Conference On Distributed Computing Systems, October 1979, pp. 185-192.

# A GEOMETRIC MODEL APPROACH TO REPRESENTING GRAPH-SEARCH PROBLEMS: FIRST RESULTS

John Gaschnig

Artificial Intelligence Center
SRI International
Menlo Park, CA 94025

Abstract

A change in representation can sometimes simplify a problem considerably. Given a problem (defined in some specific representation or encoding), it would be useful then to be able to find alternative representations for it, preferably ones that simplify the problem. Better yet would be the ability to generate such "simpler" encodings automatically. This paper introduces a new approach toward these goals, based on geometric models of graph-search problems (i.e., embeddings of the vertices of the problem graph in a d-dimensional Euclidean space). We show how the familiar English description of a problem can be mapped into its corresponding geometric model, and we give examples of the "familiar" geometric model and alternative geometric models of several well known problems (including a "Seven squares" version of the Water Jug problem that is simpler to solve than the familiar version). We propose an "operator partition entropy" measure of the simplicity of any geometric model of any problem graph; by this measure the alternative geometric models displayed in our examples are simpler than the "familiar" geometric models, suggesting that the example problems are not inherently as complicated as the "familiar" models would suggest. We comment on possible extensions to these first results, including the possibility of searching for problem representations in a state space of geometric models of a graph, using the operator partition entropy measure as a heuristic to guide the search. Note that the characteristics of a model or representation that make a problem easy or hard to solve are beyond the scope of the present paper; here we are explicitly concerned only with ways to devise alternative models for describing a problem, leaving for future work the connection between the ease of describing a problem and the ease of solving it.

## 1. Introduction

Research to date on problem representation has spanned several approaches and problem domains. The closest in spirit to the present work have focused on identifying and exploiting symmetries in problems (e.g., [Amarel 1968], [Cohen 1977]). The present paper examines the same broadly-defined class of graph-search problems considered in the latter work, a class that includes familiar problems such as the Tower of Hanoi and the Eight Puzzle as elementary examples. The object is to find a path from a given initial node to a given goal node. This class of problems has also been studied extensively as a domain for the A* best-first search algorithm (e.g., [Nilsson 1971], [Pohl 1977], [Gaschnig 1979a]). One advantage of this particular class of problems is that since a problem is defined as a graph, questions about a problem's representation, or alternative representations, or the "simplicity" of such representations can be formulated in precise terms. This paper attempts to be precise in its statements, although sometimes informal for illustrative purposes.

Amarel [1968] investigated a sequence of alternative models for the Missionaries and Cannibals problem, successively grouping the nodes of the graph into larger equivalence classes, and defining new operators (and macro-operators) accordingly. Cohen [1977] extends Amarel's work by proposing a mechanism for partitioning the nodes of a graph into equivalence classes reflecting certain sorts of symmetry. Hence both Amarel and Cohen attempt to identify abstract problems equivalent (in a certain sense) to the given problem, but having a smaller state space (i.e., number of nodes). In contrast, the present approach involves relabeling the nodes of a problem graph so as to partition the edges (rather than the nodes) into a new set of operators that may simplify the problem.

## 2. Geometric Models of Problem Graphs: Basic Concepts

Figure 1 illustrates a basic notion that the appearance of regularity or symmetry in a graph depends on how it is described or depicted. The three graphs depicted in Figure 1 are isomorphic, differing only in the assignment of nodes of the graph to coordinates in the plane. (The coordinates assigned to some of the nodes are printed in Figure 1.) One might say (informally) that regularity is inherent in the topology of a graph, but is realized (or expressed or revealed) in the arrangement by which the graph is depicted or encoded. To find paths between arbitrary nodes in Figure 1a is trivial, in Figure 1b nearly so, but apparently more difficult in Figure 1c.[1]

Now consider the following well known problems (familiarity with which is presumed):

[1] This informal claim could be stated more formally in terms relating to the performance of the A* algorithm, using as a heuristic function the orthogonal distance metric in the Euclidean plane. Such a heuristic function would estimate distances between points in the graph of Figure 1 exactly in the case of Figure 1a, rather accurately in the case of Figure 1b, and poorly in the case of Figure 1c.

| | No. of nodes | No. of edges |
|---|---|---|
| Water Jug (jug capacities 3 and 4) | 14 | 372 |
| Missionaries and Cannibals (3 missionaries, 3 cannibals, boat capacity 2) | 16 | 17 |
| Tower of Hanoi (3 discs) | 27 | 39 |
| Eight Puzzle | 91 = 362,880 | 483,840 |

Table 1. Sizes of graphs of familiar problems

It is common to encode these problems in practice in a concise form, for example, in which the configurations of the problem (i.e., nodes of the graph) are represented as tuples whose elements are drawn from finite sets of numbers, and in which the legal moves of the problem (i.e., the edges of the graph) are spanned (disjointly and exhaustively) by a set of operators, each of which is a function that takes a tuple as argument and returns the tuple of a node connected to the argument node, if any exist for that operator (i.e., if the precondition of the operator is satisfied for that argument tuple). For example, in Figure 1a one such operator might be defined as RIGHT (x, y) = (x + 1, y), whose precondition is satisfied if x < 4.

The configurations of the Water Jug problem can be represented in this way by tuples of the form (C1, C2), where C1 is the current contents of jug 1 (i.e., 0, 1, 2, or 3 units of liquid), and C2 is the current contents of jug 2 (0, 1, 2, 3, or 4 units).[3] The commonly defined operators for this problem can be called EMPTY-1 (i.e., empty jug 1), EMPTY-2, FILL-1, FILL-2, POUR-1 (i.e., pour the contents of jug 1 into jug 2 until either jug 1 is empty or jug 2 is filled), and POUR-2. Similarly, the Missionaries and Cannibals configurations can take the form (M,C,B), where M, C, and B denote the number of missionaries, cannibals, and boats on the left side of the river (i.e., 0, 1, 2, or 3 missionaries; 0, 1, 2, or 3 cannibals; 0 or 1 boat; certain combinations are forbidden). Similarly, a configuration in the Tower of Hanoi problem can take the form (D1, D2, D3), where Di denotes the number of the peg on which disk i currently resides (the pegs are numbered 1, 2, and 3). Similarly, a configuration in the Eight Puzzle can take the form (T1, T2, ..., T8), where Ti denotes the square of the board on which the tile numbered i currently resides (the squares are labeled 1 through 9 in some convenient ordering).[4]

The present approach is to consider each instance of such a tuple as the coordinates of a point in Euclidean space. Then the above tuple encodings determine an assignment of the nodes of the problem graph to coordinates in space, and the edges connect pairs of coordinates in space. Hence the familiar English description can be mapped into a geometric picture. Figures 2a, 3a, and 4a depict the geometric models corresponding to the Water Jug, Missionaries and Cannibals, and Tower of Hanoi encodings given above. In Figure 2a the sequence of edges labelled P1, P2, P3, P4, P5, P6 indicates the moves to transform the initial state (0,0) into the goal state (2,0).

## 3. Examples of Alternative Geometric Models

It is clear that an infinite number of geometric models of the Water Jug problem (or of any other problem graph) is possible, since the nodes of that graph can be assigned to arbitrary distinct coordinates in N-dimensional space (for arbitrary N > 1). Our objective now is to find models that appear to be simpler than the one in Figure 2a (or Figures 3a or 4a, by analogy). Figures 2b, 3b, and 4b depict alternative geometric models for our example problems. The origin of these alternative models reflects diverse sources and methods: trial and error, observations made in previous work for other purposes [Nilsson 1971, p. 82], [Amarel 1968, p. 145], and a method similar to that in [Cohen 1977] for identifying equivalence classes of nodes in the graph that reflect topological symmetries. (For example, in the "Seven Squares" model depicted in Figure 2b and described in English subsequently, the pairing of nodes reflects a symmetry discovered by this mechanical method.) The sequence of edges P1, P2, ..., P6 in Figure 2b correspond to those having the same labels in Figure 2a, and the nodes connected by these edges are similarly in correspondence. Visually at least, the alternative models given in Figures 2b, 3b, and 4b appear simpler than the corresponding "familiar" models in Figures 2a, 3a, and 4a. Apparently these problem graphs are not so complicated as the "familiar" models would suggest.

Just as we transformed an English description into Figure 2a, so we can attempt to describe Figure 2b in English, thus:

The "Seven Squares" problem

> There is a board consisting of seven squares in a row, numbered 0 through 6, and a checker that is colored white on one side and black on the other. The checker can occupy any square with either color facing up. From any (color, square) state, the checker may move to either adjacent square without flipping color. (The end squares of course have only one adjacent square.) These moves (i.e., edges) could be covered by operators RIGHT and LEFT

[2] Note that edges in the Water Jug graph are directed (since some operators have no inverse), while those in the graphs of the other problems cited in Table 1 are undirected.

[3] Certain combinations are precluded by the preconditions of the operators.

[4] Note that the Water Jug problem is a directed graph (since some operators have no inverse), whereas the other problems considered here are undirected graphs.

defined as for Figure 1a. If the checker is on square 6, it may flip color and remain on square 6 (this set of moves covered by an operator called FLIP6). From any state, it may jump to square 1, flipping color if it jumped from an even-numbered square, and not flipping if from an odd-numbered square (operator JUMP1). It may jump to square 0 from any state, flipping if it jumped from squares 1, 3, or 4 (operator JUMP0-FLIP), and not flipping otherwise (operator JUMP0-NO-FLIP).

Although finding a minimal-length path in Seven Squares may not be trivial, it is trivial to find a non-minimal length path for any initial state S and goal state G: if S and G are the same color, simply move RIGHT (or LEFT as the case may be) from S to G, otherwise move RIGHT to square 6, flip, then move LEFT to G.[5]

Superficially the Water Jug problem and the Seven Squares problem are not similar at all, yet their graphs are isomorphic. Just as Figure 1a is an apparently simpler isomorph of Figure 1c, so Figure 2b is an apparently simpler isomorph of Figure 2a.[6,7]

[5]This approach ignores (i.e., doesn't use) the JUMP0, JUMP1, JUMP0-FLIP, and JUMP0-NO-FLIP operators. In a sense, then, we are thereby finding paths in a more constrained graph, i.e., the variant of Figure 2b in which the edges corresponding to the JUMP operators are deleted. An aspect of the implication of deleting edges in problem graphs is considered in an "edge subgraph" approach to devising heuristics [Gaschnig 1979b].

[6]There does not appear to be any simply stated rule for mapping between states in Figure 2a and the corresponding states in Figure 2b. This issue of mapping between geometric models may turn out to be important for practical considerations, but is beyond the scope of the present initial investigation.

## 4. An "Operator Partition Entropy" Measure

So far we have demonstrated how a problem representation can be mapped to a geometric embedding or model of a graph, and have presented instances of such models for several common problems, including "familiar" models and alternative models that appear to be simpler, at least with respect to one's visual intuition. Now we propose a quantitative measure of the simplicity of an arbitrary geometric model of an arbitrary problem graph.

The definition of operator partition entropy is motivated (although only loosely) by the information theoretic approach of Chaitin [1975]. In the present context, we observe that a theory about problem representation can try to account for and measure the information content of a problem.[8] Any member of the particular class of finite, strongly connected graphs considered here can be reconstructed from the closure of applications of the operators to any arbitrary node. Hence here we seek a measure related in some way to the amount of information required to specify the operators, i.e., to encode the precondition and "action" of each operator. In this first effort, we ignore the precondition and consider only the "action" of an operator.

Our approach is that each geometric model of a

[7]Our discovery of the "Seven Squares" model of the Water Jug was aided by the results of a mechanical process for partitioning the nodes (as opposed to the edges) of the Water Jug graph into seven equivalence classes corresponding to the seven squares, each class containing two nodes (corresponding to the checker's color). This symmetry-factoring algorithm involves only topological considerations, and is similar to a method described by Cohen [1977]. The method successively splits the classes of an initial partition, according to the following principles: (1) initially partition the nodes according to the number of incident edges; (2) associate a symbol (e.g., A, B, etc.) with each such equivalence class, (3) choose one equivalence class (e.g., A) and associate with each of its member nodes $n_i$ the list of the nodes $n_j$ connecting $n_i$ by an edge; (4) in the list associated with each such node $n_i$ replace each node $n_j$ by the symbol denoting the equivalence class to which $n_j$ belongs, hence associating with each node $n_i$ a multiset of class names (e.g., (A,B,B,C); (5) partition the nodes $n_i$ into new equivalence classes, each of whose members have identical associated multisets. The method proceeds (details omitted here) until no further partitions ensue.

[8]To be concrete, the Eight Puzzle graph has 9! nodes and 4/3*9! edges, but its connection matrix can be constructed from instructions far less verbose than 9! bits.

graph (i.e., assignment of the nodes to distinct points in an N-dimensional Euclidean space) should determine a unique partition of the edges into exhaustive, mutually disjoint subsets, i.e., into equivalence classes. (By our approach each equivalence class will correspond to an operator.) From this partition of the edges of the graph we shall compute a number.

To illustrate the approach, consider Figure 1a. It is convenient in practice to partition the edges of Figure 1a into four operators: RIGHT, LEFT, UP, DOWN. Hence for all (x, y) such that x < 4, RIGHT(x, y) = (x + 1, y) specifies a (directed) edge from the node (x, y) to the node (x + 1, y). We adopt the shorthand notation RIGHT(x, y) = <1, 0> to indicate that RIGHT(x, y) adds 1 to its x argument and 0 to its y argument. We call <1, 0> the action expression of RIGHT. By this convention the action expression of DOWN is <0, 1>. Hence the edges of Figure 1a are covered by four action expressions: <1, 0>, <0, 1>, <-1, 0>, <0, -1>. In this way a geometric model determines a set of action expressions covering all the edges. (I.e., the operators are not chosen freely but are determined by the geometric model). For simplicity and convenience we shall actually consider undirected edges rather than directed edges, and say that Figure 1a determine two action expressions: HORIZONTAL = <1, 0> and VERTICAL = <0, 1>.

Similarly, Figure 1b determines the action expressions <1, 0>, <1, 1>, and <1, -1>, covering 12, 6, and 6 edges respectively. Similarly, Figure 1c determines action expressions <1, 0>, <1, -1>, <1, 1> <1, -2>, and <1, -4>, covering 9, 9, 4, 1, and 1 edges, respectively. This approach applies to the models in Figures 2, 3, and 4 as well, so that in general a geometric model determines an action class partition (or operator partition) of the edges.[9]

With each operator i we can associate the fraction $p_i$ of the total number of edges covered by that operator (i.e., action expression). This suggests the entropy formula standard in information theory:

$$H_{op} = -\sum_{i=1}^{c} p_i \log_2 p_i,$$

where c is the number of operators (i.e., equivalence classes in the partition). $H_{op}$ takes the maximum value $\log_2 m$ when each of the m edges belongs to a distinct equivalence class; $H_{op}$ takes the minimum value 0 when all m edges belong to the same equivalence class.

We make no claim that $H_{op}$ is the most appropriate measure of the simplicity of a geometric model, but simply that investigations of its properties may provide insight for future extensions of the present results. Nevertheless, we do note the fundamental result in information theory [Khinchin 1957, Shannon & Weaver 1972] that the number of bits to transmit a symbol chosen from a finite collection of c symbols, each symbol $s_i$ having probability $p_i$ of being selected for transmission, is given by the $H_{op}$ formula above. Hence one could transmit the m edges of a graph using $m*H_{op}$ bits. (Whether this is of more than simply metaphorical significance in the present case is debatable.)

The model in Figure 1a has two operators (HORIZONTAL and VERTICAL), each spanning 12 edges, hence P = .5 for each operator. Hence we obtain $H_{op} = -(.5 \log .5 + .5 \log .5) = 1$, compared with the maximum value $H_{max} = \log 24 = 4.58$.

In Figure 1b there are 3 operators, having P = 0.5, 0.25, and 0.25, respectively, giving $H_{op} = 1.5$. The operators in Figure 1c span 9, 9, 4, 1, and 1 edges respectively, giving $H_{op} = 3.605$. As fractions of the maximum value 4.58, the $H_{op}$ values of Figures 1a, 1b, and 1c are $F_{op} = H_{op}/H_{max} = 0.22$, 0.33, and 0.79, respectively.

Table 2 similarly compares the "familiar" and alternative geometric models of the Water Jug, Missionaries and Cannibals, and Tower of Hanoi. Besides the version of the Water Jug problem having jug capacities 3 and 4, the following table lists analogous results for a version having jug capacities 5 and 8. Note that the "Seven Squares" model generalizes in the latter case to the "13 Squares" model, having an appearance and operator partition similar to the those of the "Seven Squares" model. Note that scaling the Water Jug problem parametrically in this way causes little change in the value of $H_{op}$, for either the "familiar" or alternative models, because the parametrically larger larger problems simply have more edges covered by each operator (as defined in terms of action expression) rather than introducing new operators (i.e., new action expressions not included among those for the smaller variant).

| | $H_{op}$ (fam.) | $H_{op}$ (alt.) | $H_{max}$ | $F_{op}$ (fam.) | $F_{op}$ (alt.) |
|---|---|---|---|---|---|
| Water Jug(3,4) | 2.783 | 2.40 | 5.64 | 0.49 | 0.43 |
| Water Jug(5,8) | 2.787 | 2.35 | 6.62 | 0.42 | 0.35 |
| Miss. & Can. | 2.19 | 1.13 | 4.09 | 0.54 | 0.28 |
| Tower of Hanoi | 2.06 | 1.58 | 5.29 | 0.39 | 0.30 |

Table 2. Comparative operator partition entropy values for familiar problems
(fam. = familiar model, alt. = alternative model, as discussed in text)

We have thus proposed one possible measure of the simplicity of a geometric model of a problem graph. Although $H_{op}$ captures a very incomplete notion of "simplicity", its values tabulated above are reasonably consistent with intuition.[10]

---

[9] The approach is generalized somewhat in the case of the Water Jug problem.

[10] Information measures of graphs were apparently first introduced by Rashevsky [1955] in a chemical application, and later investigated by Trucco [1956], Mowshowitz [1968], Boncher, et al. [1976] and others, although they partitioned the nodes of the graph in a topological manner similar to Cohen [1977], as opposed to partitioning the edges according to geometric considerations as in the present work.

## 5. Application to Larger Problems

So far we have presented geometric models of several problem graphs having no more than 27 nodes and 39 edges, which classify as "toy" problems. While the examples may be of some theoretical interest, proving practical significance of the present approach requires applications to larger problems. The Eight Puzzle, having 9! = 362,880 nodes and 4/3*9! = 483,840 edges, is a suitable candidate.

The "familiar" model of the Eight Puzzle (corresponding to the 8-tuple encoding mentioned in Section 2) is a structure in 8-dimensional Euclidean space. Clearly we desire an alternate encoding having fewer dimensions. It is also clear that we do not actually want to draw a model of the complete graph (although we note that chemists do make 3-D models of large molecules). More practical would be an attempt to devise a model of part of the graph, and attempt to derive insight about the structure of the Eight Puzzle by examining it.

The previous section presented evidence in the case of the Water Jug problem that it is possible to scale a geometric model to a parametrically larger version of a problem. This suggests that we approach the Eight Puzzle by first considering the Five Puzzle, a 3 by 2 version having 6! = 720 nodes. Even this graph is relatively large, and smaller versions of the puzzle are degenerate.

To approach the Five Puzzle we take a new tack, namely to identify a problem that is similar to the Five Puzzle or Eight Puzzle, but which scales down to smaller sizes. One such problem is called the "MAXSWAP" problem, introduced by Gaschnig [1979b] in another context. This problem is to sort permutations of the sequence 1, 2, ..., N by iteratively exchanging pairs of elements, subject to the restriction that the element N must participate in every swap (i.e., swap N with some other element). The correlate of the Five Puzzle takes N = 6, and for the Eight Puzzle N = 9. The element N in MAXSWAP corresponds to the hole in the Eight Puzzle or Five Puzzle, so that 9MAXSWAP is like the Eight Puzzle except that any tile can jump into the hole (as opposed to only tiles adjacent to the hole). Hence 9MAXSWAP and the Eight puzzle both have 9! nodes, but 9MAXSWAP has more edges than the Eight Puzzle (exactly 3 times as many, in fact). Considering problems that have more edges than the Five Puzzle (or Eight Puzzle) would seem to make the task of finding a geometric model harder rather than easier, but at least we can start with a very small case, namely 3MAXSWAP, and attempt to scale it to larger versions of MAXSWAP, and then attempt to devise a geometric model for the Five Puzzle similar to those for MAXSWAP graphs. This is exactly what we shall do, as follows.

The 3MAXSWAP graph has 3! = 6 nodes and 9 edges. A "familiar" encoding can take the form (P1, P2, P3), where Pi denotes the position in the permutation of element i. Figure 5a depicts a two-dimensional perspective drawing of this "familiar" 3-dimensional model of the 3MAXSWAP graph. Figure 5b depicts a two-dimensional alternate model for 3MAXSWAP. $H_{op}$ = 1.58 and 0.92, respectively for these two models, compared with the maximum value log 9 = 3.17.

The 4MAXSWAP graph has 24 nodes and 36 edges. Figure 6 depicts a two-dimensional geometric model of 4MAXSWAP, for which $H_{op}$ = 1.69, compared with the maximum value log 36 = 5.17, and the value of $H_{op}$ for the "familiar" encoding, namely 4.17.

The generalization of the "rectangular" model from 3MAXSWAP (Figure 5b) to 4MAXSWAP (Figure 6) suggests that it may generalize further, and may possibly serve as a guide for a simple model for the Five Puzzle. Accordingly, we expanded a portion of the Five Puzzle graph having 147 nodes and 155 edges (i.e., a breadth-first expansion to 5 levels below a particular starting configuration), and attempted to devise a rectangular-like geometric model for it. Figure 7 presents the result, depicting approximately half of the partial model we devised. To visualize the geometric model, imagine that the points labelled A, B, and C are not on the page, but instead are placed in space one unit of distance above the page. Now imagine a copy of the structure in Figure 7, except for points A, B, and C, raised 2 units of distance above the page. Call by the names X', Y', and Z' the points in this raised copy corresponding to the points labelled X, Y, and Z, respectively in the copy on the page. Now imagine X connected to A, X' connected to A, Y to B, Y' to B, Z to C, and Z' to C. Hence the model has two parallel structures (on the page and above the page), each having 72 nodes, connected to three intermediate points. The node labelled A was the root node in the breadth-first expansion. In this model there are 8 action classes, covering 86, 40, 15, 8, 2, 2, 1, and 1 edges, respectively, yielding $H_{op}$ = 1.78, compared with the maximum value log 155 = 7.28. Note that in this portion of the Five Puzzle graph, to go from any point in the portion on the page to any point in the portion raised above the page (or vice versa), one must go through one of the intermediate points A, B, or C. (Of course, the further expansion of this graph beyond the present 147 nodes may uncover other nodes connecting the two portions of the model presented here.) The factoring of this portion of the Five Puzzle graph into two symmetric portions and three intermediate nodes was discovered by the use of a symmetry factoring method similar to that of Cohen [1977].

Intuition suggests that extending the "rectangle" model to the entire 6MAXSWAP or Five Puzzle problems may prove problematic. By way of considering other structures, the symmetry in Figure 6 suggests wrapping around the two ends of the long axis into a regular polygon. Figure 8 shows a "cartwheel" geometric model for the 4MAXSWAP graph.

The exercise presented in this section suggests at least the feasibility of devising geometric models of graphs having more than a few dozen edges, and the possibility that devising such models may promote insight about the structure of the problem. Future work may be able to expand upon these initial efforts.

## 6. Discussion

We have demonstrated several examples of a general approach to studying some issues of problem representation in a restricted context. Given an English description of a problem, we convert it to a "tuple format" suggested by the English description, then partition the edges into equivalence classes (operators) in a mechanical fashion according to the arithmetical differences between the tuples representing nodes incident to each edge (e.g., all edges having the effect of "Move right 1 unit" belong to the same class), then compute $H_{op}$ based on the fraction $p_i$ of edges in

each edge class i, and then attempt to find an alternative geometric model whose $H_{op}$ is smaller than that for the "familiar" model. All but the last of these steps are (more or less) methodical.

These first results leave many questions unanswered. Our intention here has been merely to introduce the ideas for further consideration by other researchers. Hence the remainder of this discussion focuses on possible extensions to the present initial efforts.

To be concrete, we imposed several simplifying assumptions, e.g., to ignore the preconditions of operators, to consider only additive action expressions for operators, to consider only the $H_{op}$ measure. Alternative assumptions and generalizations also merit investigation.

At present we rely mostly on human creativity to find alternative geometric models for a given problem graph. The present approach suggests the feasibility of searching for problem representations in a state space of geometric models of a graph, guiding the search heuristically by means of a metric such as the proposed operator partition formula. The size of this state space is so enormous, however, as to require a careful consideration of efficiency requirements, perhaps in generating candidate models selectively, or in devising additional heuristics as alternatives to $H_{op}$. One possibly fruitful approach may be to generate alternative models for a given problem interactively, allowing the user to guide the exploration for a simpler model.

The practical benefit of discovering an alternative simpler representation of a given problem to be solved depends on the ability to translate a specified instance of a given problem into its new encoding, solve the instance therein, and translate the resulting solution path back into the terms of the original representation. Our example of the Water Jug/Seven Squares isomorphs suggests the possibility that simple mappings between alternate encodings may be difficult to identify, especially between encodings that seem intuitively to be rather different. Such a finding (assuming it could be formalized) could severely limit the practical utility of this whole approach, although perhaps theoretically interesting.

The examples suggest in a concrete way that the familiar representation of a problem may be far from the simplest possible. It would be interesting then to know whether there exist simpler versions of other common problems, or yet simpler versions of the present examples (i.e., an issue of optimal geometric models).

Despite the limitations of our initial results, one advantage we perceive of the present approach is that a geometric model of a graph can be visualized and manipulated as a structure in Euclidean space, which seems to promote some insight or intuition about the elusive concept of problem representation. For example, some two-dimensional models of a problem graph may prove useful in analyzing a given heuristic for the problem, since the value assigned to each node by the heuristic can be plotted as an elevation above the plane. Then the heuristic can be viewed as a polyhedral surface, whose "hills" and "valleys" may permit a visual analysis of the efficiency of the heuristic function.[11]

In any case, further investigations of this geometric modelling approach to understanding more quantitatively some issues of problem representation would seem to be merited on both theoretical and (potentially) practical grounds.

References


1. Amarel, S., "On Representations of Problems of Reasoning about Actions," in Machine Intelligence 3, D. Michie (ed.) American Elsevier Publ. Co., New York, 1968.

2. Bonchev, D., D. Kamenski and V. Kamenska, "Symmetry and Information Content of Chemical Structures," Bulletin of Mathematical Biology, Vol. 38, 1976, pp. 119–133.

3. Chaitin, G., "Randomness and Mathematical Proof," Scientific American, May 1975.

4. Cohen, B., "The Mechanical Discovery of Certain Problem Symmetries," Artificial Intelligence, Vol. 8, No. 1, 1977, pp. 119–131.

5. Gaschnig, J., "Performance Measurement and Analysis of Certain Search Algorithms," Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., May 1979 (1979a).

6. Gaschnig, J., "A Problem Similarity Approach to Devising Heuristics: First Results," Proc. Sixth Intl. Joint Conf. on Artificial Intelligence, Tokyo, August 1979 (1979b).

7. Khinchin, A. I., Mathematical Foundations of Information Theory, Dover Publications, Inc., New York 1957.

8. Mowshowitz, A., "Entropy and the Complexity of Graphs: I. An Index of the Relative Complexity of a Graph," Bulletin of Mathematical Biophysics, Vol. 30, 1968, pp. 175–204.

9. Nilsson, N., Problem Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

10. Pohl, I., "Practical and Theoretical Considerations in Heuristic Search Algorithms," in Machine Intelligence 8 (E. Elcock and D. Michie, eds.), Ellis Horwood Ltd., Chichester, England, 1977.

11. Rashevsky, N., "Life, Information Theory, and Topology," Bulletin of Mathematical Biophysics, Vol. 17, 1955, pp. 229–235.

12. Shannon, C. and W. Weaver, The Mathematical Theory of Communication, University of Illinois Press, Urbana, Illinois, 1972 edition.

13. Trucco, E., "A Note on the Information Content of Graphs," Bulletin of Mathematical Biophysics, Vol. 18, 1956, pp. 129–135.


---

[11] For example, Gaschnig [1979a, p. 250] discusses a particular Eight Puzzle heuristic as descending toward a "trench" and then following it to the goal node.

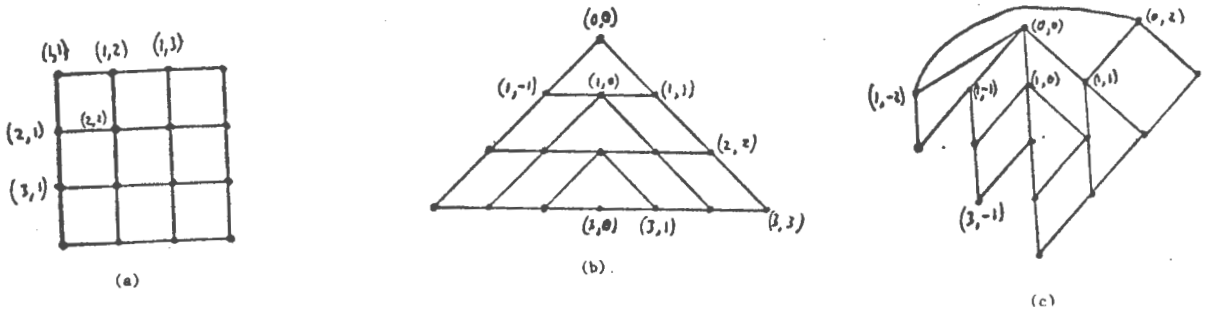



Figure 1. Different geometric models of isomorphic graphs

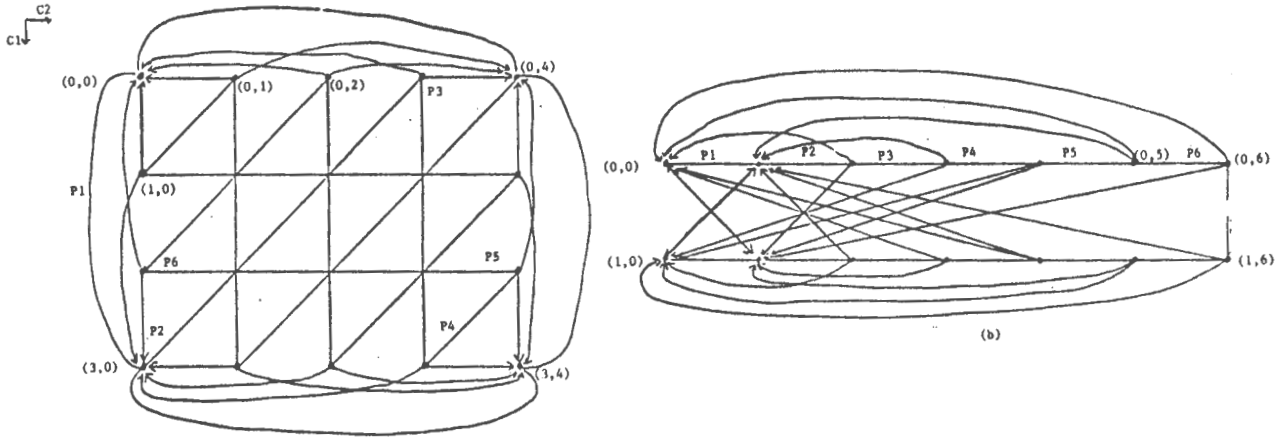



Figure 2. "Familiar" geometric model (a) and "Seven Squares" geometric model (b) for the Water Jug problem

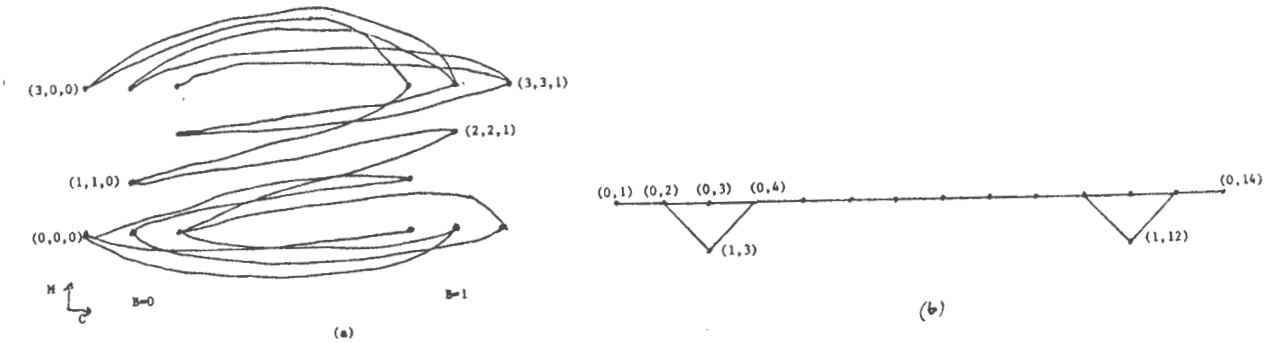
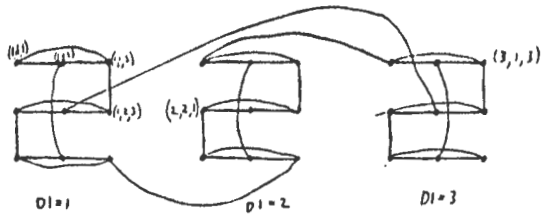
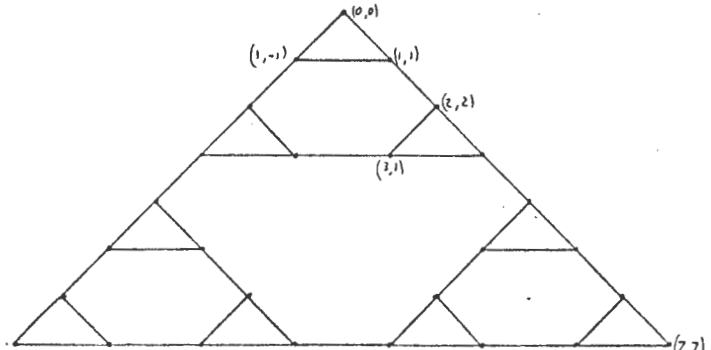



Figure 3. "Familiar" (a) and alternate (b) geometric models for the Missionaries and Cannibals problems




(a)




(b)

Figure 4. "Familiar" (a) and "Triangle" geometric models for the Tower of Hanoi problem

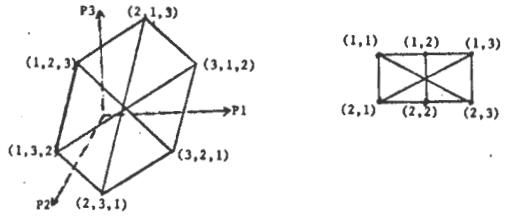






Figure 5. (a) "Familiar" 3-D geometric model of 3MAXSWAP problem graph
(Note: origin is (1,1,1).)
(b) Alternative "rectangular" geometric model of 3MAXSWAP problem

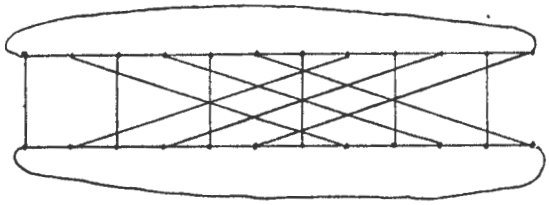


Figure 6. "Rectangular" geometric model of 4MAXSWAP problem graph

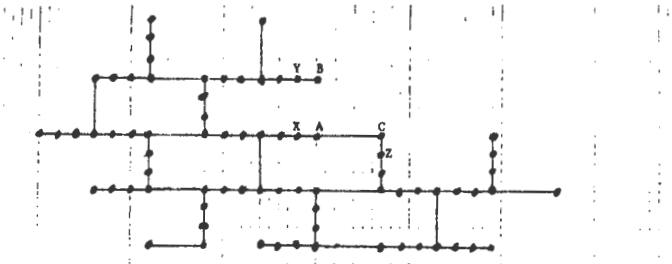



Figure 7. Portion of a geometric model for a portion of the Five Puzzle graph
(See text.)

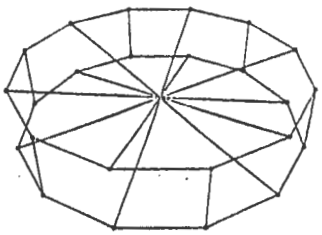


Figure 8. "Cartwheel" geometric model of 4MAXSWAP problem graph (Compare with Figure 6)
Note: Six edges pass through the center of the picture, but there is no node of the graph there.

# PLANNING IN A DYNAMIC MICROWORLD

Gordon I. McCalla

Department of Computational Science
University of Saskatchewan
Saskatoon, Saskatchewan

Peter F. Schneider

Department of Computer Science
University of Toronto
Toronto, Ontario

## Abstract

This paper discusses a planning system that works in a dynamically changing geographic microworld (an abstraction of the world faced by a taxi driver). The discussion concentrates on explaining how geographic knowledge is represented in a single data structure, the route; how plans are constructed from such routes; and how such routes can be automatically acquired to augment the knowledge base. The important lesson to be learned from this approach is the usefulness of taking an integrated view of planning, execution, and acquisition.

## 1 Introduction

Recently there has been some discussion in the literature (Sacerdoti (1979)) about the need for complete planners, i.e. planners which not only produce plans, but also execute them. There has also been interest (e.g. Sacerdoti (1979)) in the problems of planning for a changeable or dynamic microworld. For the last couple of years (McCalla *et al* (1978), McCalla and Schneider (1979)) we have been working on such a complete planning system to produce and execute plans for a dynamic geographic microworld. Specifically our planning system is designed to produce and execute plans that guide a simulated robot taxi driver (named ELMER) through a simulated city which not only contains streets, intersections, and other unchangeable features, but also contains dynamic features such as traffic lights, other cars, and pedestrians.

The architecture of the ELMER system is shown in Figure 1. Basically, a plan to go from some point x to some other point y is produced by the Planner (using route information provided by the Map) and sent for execution to the Executor. The Executor then augments the plan with common world sense (e.g. how to maintain speed, how to avoid pedestrians, how to stop at red lights or stop signs, etc.) and proceeds to carry it out. Once the plan has been successfully completed, the "instantiated" plan is sent to the Map where it is adapted to form a route from x to y. This route can then be used by the Planner to aid in future plan production. What we have, then, is not only a complete planning system which operates in a dynamic microworld, but also a planning system which can in some sense learn from its past behaviour. In this regard the ELMER system supports Simon's (1979) contention that learning may once again be an appropriate AI endeavour.

The complete ELMER system has been described in McCalla *et al* (1978) and the Executor has been further elaborated in McCalla and Schneider (1979). This paper discusses the Planner and the Map. A forthcoming technical report (Reid and McCalla (1980)) describes the implementation aspects.

## 2 The Map

The Map stores all information using a single data structure: the route (in contrast to Kuipers (1977) where multiple representations are used for geographic knowledge). Figures 3 through 6 illustrate typical routes through Simon City (Figure 2). Figure 3, for example, describes a path along Kuipers Crescent. Each box represents the traversal of a particular region in Simon City. the lower the box, the more local the region. Thus, box 16, representing a route along Kuipers Crescent from Lenat Lane to Schank Strip consists of three sub-boxes (boxes 17, 18, and 19), representing routes through the obvious three sub-regions. Box 17, in turn, is still further specified. (Note that for clarity many of the low-level route boxes have been omitted from Figures 3-6.) The labelled arrows leaving a box represent transitions from one route to another; the associations in a box are pointers to other

routes which are related to this route in the sense that it is "easy" to get from this route to those other routes. Transitions are important in plan execution but are not especially significant to the Planner or Map. On the other hand, associations are critical to the planning process and should be further explained.

To illustrate the nature of associations, it is instructive to look at an example. Sub-route 41 in the Lenat route (Figure 4) associates into sub-route 17 in the Kuipers route (Figure 3) since if ELMER were driving along Lenat from Schank to Kuipers (a la sub-route 41) he could readily transfer so as to be driving along Kuipers from Lenat to Winograd (a la sub-route 17). Similarly sub-route 82 (entering the Lenat@Kuipers intersection from the west) associates into sub-route 70 (leaving the Lenat@Kuipers intersection heading north). But note that sub-route 90 (leaving Lenat@Kuipers heading east) does not associate into sub-route 70 because such a transfer would require ELMER to make a U-turn and retrace his steps. This emphasizes the one-way nature of both routes and associations. It is important to realize, however, that if there is an association from route A to route B, there is an inverse association (inassociation) from B to A, thus allowing associations to be traced both directions.

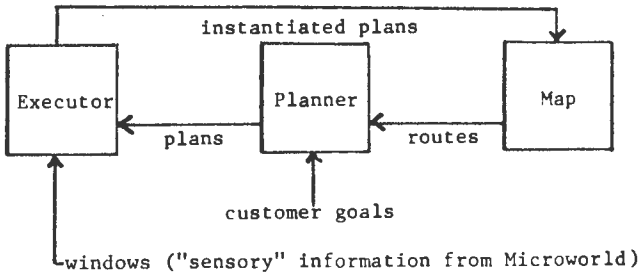



Figure 1 - Basic System Architecture

## 3 The Planner

### 3.1 Overview of the Planner

The Planner is charged with the task of producing a plan to go from some start region x to some end region y. (From the Planner's point of view points are just very small regions.) Eventually we would like to enhance the Planner to force it to produce plans satisfying constraints as to how fast a destination must be reached or specifying certain milestones which must be passed en route (as in for example Hayes-Roth and Hayes-Roth (1979)). Currently the Planner is satisfied with producing the first successful plan that it comes upon.

The Planner basically works by adapting Map routes to satisfy the requirements of its current task. Initially the start region x must be defined by a route X from the Map and the end region y by a route Y from the Map. These defining routes for all intents and purposes become the Planner's start and end points. Essentially, routes containing sub-routes into which X associates are candidates for getting ELMER off to a good start, and routes containing sub-routes to which Y inassociates are candidates for getting ELMER to his destination. If there is a route C containing a sub-route Ri(C) into which X associates and another sub-route Rj(C) to which Y inassociates, and if the

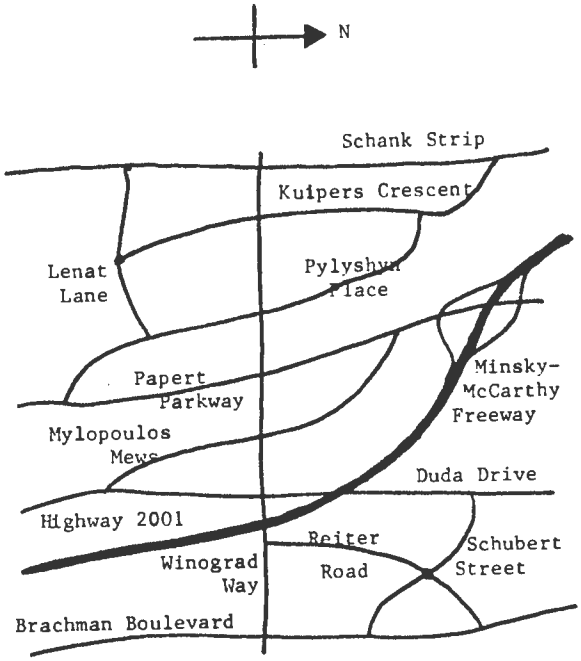



Figure 2 - Simon City

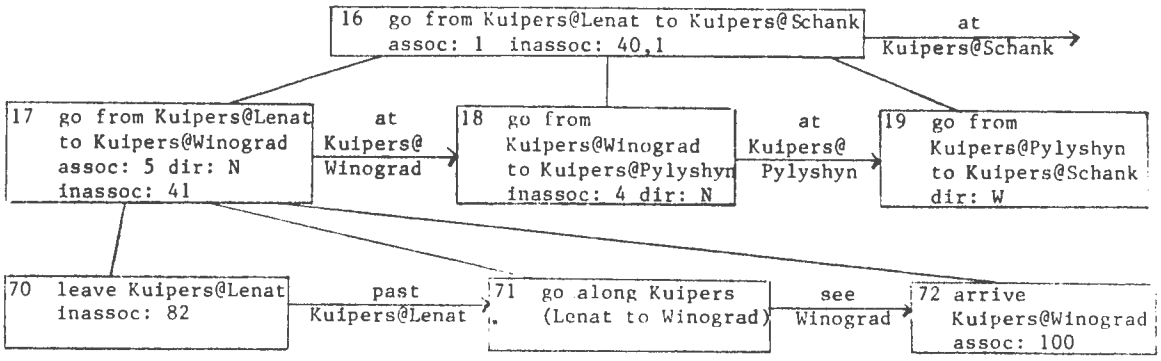



Figure 3 - Kuipers Route

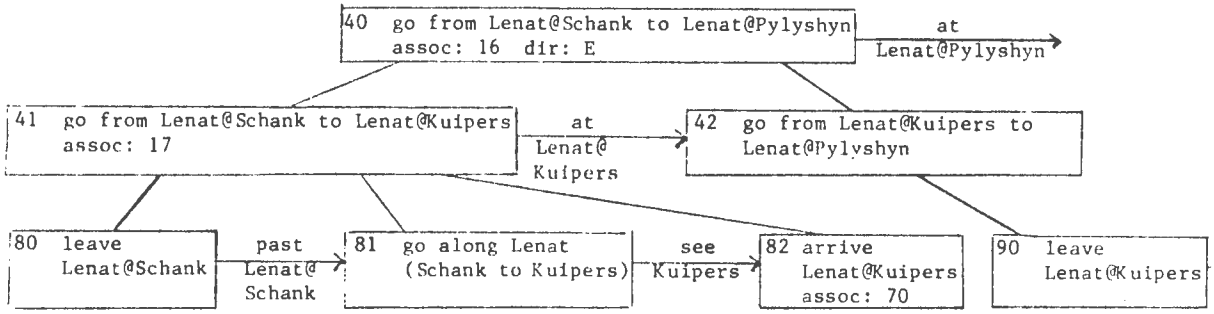



Figure 4 - Lenat Route

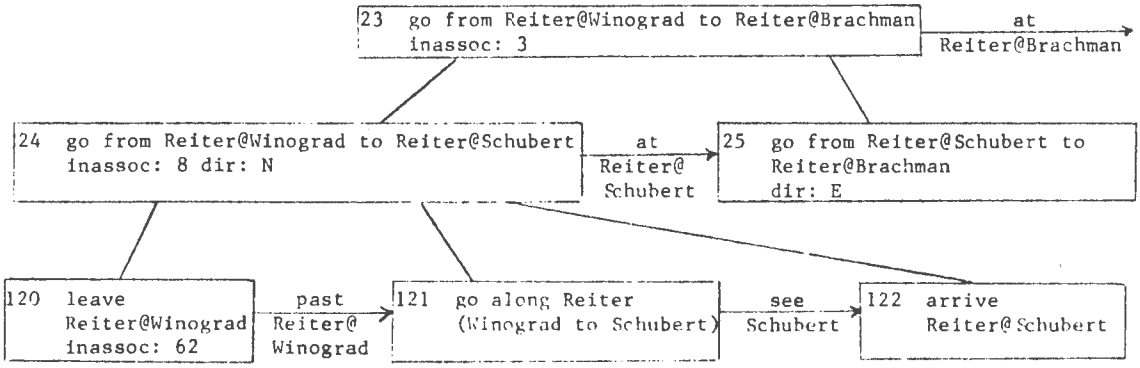
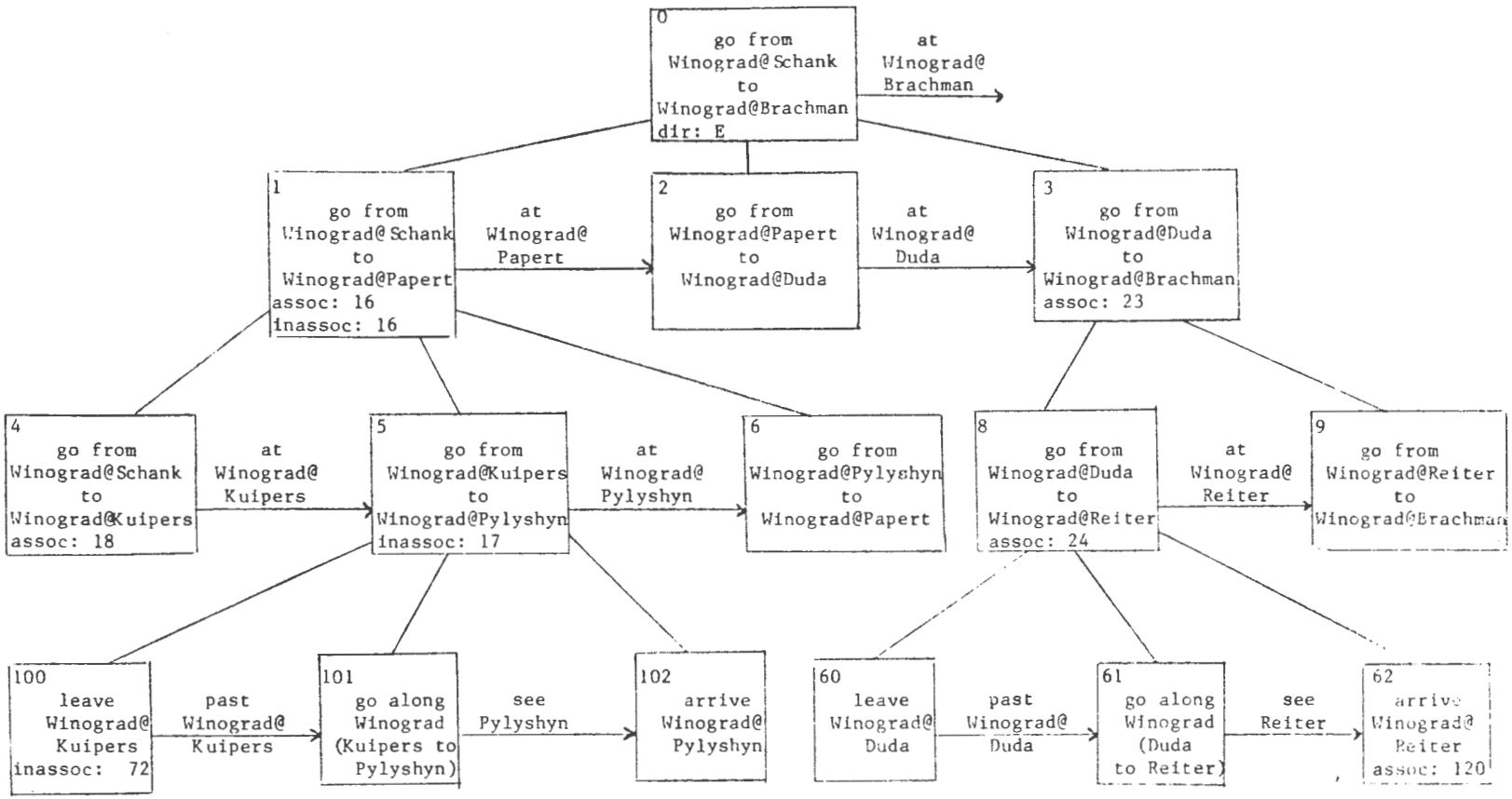


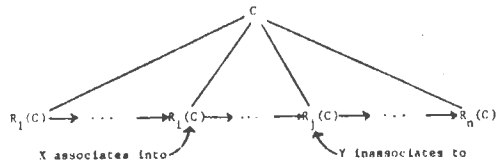

Figure 5 - Reiter Route




(naming convention: the first street named in an intersection pair is the street along which ELMER is heading)

Figure 6 - Winograd Route

sub-route Ri(C) comes before (i.e. to the left of) Rj(C), i.e.




then C can be rather easily adapted to form the plan to get from x to y. This process requires further elaborating Ri(C) (because of a different start region x) and further elaborating Rj(C) (because of a different end region y) but adopting Ri+1(C) through Rj-1(C) exactly as they already exist. The ability to use such large chunks of pre-planned routes makes the planning process very efficient.

If a route C containing both an association from X and an inassociation from Y cannot be found, then a route A containing an association from X must be spliced with a route B containing an inassociation from Y. The process is more difficult but is still manageable and reasonably efficient (since large parts of A and B can still be adopted intact). The current Planner implementation does not try such splicing, although the algorithm is well defined (see below).

## 3.2 Detailed Planning Algorithms

In the above overview of the planning process it was briefly described how a problem of the form "go from region x to region y" could be reduced to the three sub-problems "go from region x to sub-route Ri(C) of an existing Map route C", "adopt the intermediate sub-routes Ri+1(C) to Rj-1(C) unchanged from C", and "go from sub-route Rj(C) to region y". After making this initial breakdown, the Planner attempts to solve the first sub-problem in its entirety before proceeding to the other two sub-problems. This enables it to complete a partial plan as quickly as possible to pass on to the Executor so that ELMER can begin his travels without undue delay. Although this contrasts with NOAH's (Sacerdoti (1977)) complete level by level expansion, there doesn't seem to be the potential for the sub-goal conflicts which NOAH's methodologies were meant to avoid. This is due to the lack of unordered conjunctive goals (the kinds of goals leading to most of the difficulties) and due to the fact that the Planner is largely adapting previously successful routes.

Three types of problems have been illustrated so far: the region-to-region problem, the region-to-route problem, and the adapt-sub-routes problem. There are two other types of problem: the route-to-route problem and the route-to-region problem. To solve an adopt-sub-routes problem, the Planner need only copy over the sub-routes from the existing Map route. Solving a route-to-region problem is completely analogous to solving a region-to-route problem. All that remains, therefore, is to give detailed algorithms for solving region-to-region, region-to-route, and route-to-route problems.

### 3.2.1 Solving Region-to-region Problems

The following algorithm is called in when it is necessary to solve the problem of going from region x to region y. First the defining routes X and Y must be sought out (currently these are predefined). In the algorithm s (e) marks those routes into which X (Y) directly associates (inassociates); S (E) marks those routes which have a sub-route into which X (Y) directly associates (inassociates). The algorithm iterates through steps 2 to 5 looking for the elusive route C connecting X to Y. If such a route isn't found during an iteration, the current version of X (Y) is replaced by its super-route before proceeding around the loop again. This corresponds to replacing initially small regions by larger regions that are more likely to be connected by an existing Map route. The generalization stops when both X and Y have the same super-route. This is analogous to finding a region containing both X and Y, implying that no further information can be gained from the Map. In this case step 6 is called in to try route splicing. The algorithm proceeds as follows:

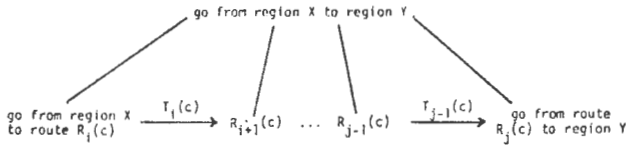
1. Mark route X (Y) with an s (e) and go to step 3.

2. Find all routes which are both
   i. not marked with an s (e) and
   ii. direct super-routes of routes marked with an s (e) and not marked with an e or E (s or S)
and mark them with an s (e).

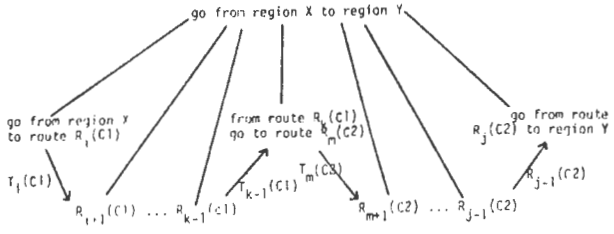
3. For all routes that have just been marked with an s (e), mark any associated (inassociated) routes not already marked with an s (e) with an s (e).

4. Mark all routes which use a route marked with an s (e) as a sub-route (not necessarily directly) with an S (E).

5. If some route C is marked with both an S and an E and, moreover, has a sub-route Ri(C) marked with an s or S and a sub-route Rj(C) marked with an e or E and $i \leq j$, then C is a route from some region containing x to a region containing y and the region-to-region problem can be solved by adapting C as follows:




6. If in step 2 or step 4 no routes are marked then there is no single connecting route C. In this case a pair of routes must be found and spliced together. This is done by searching all routes marked S and e or E and s for a pair of routes C1 and C2 with the following properties. First, C1 is marked S and e and C2 is marked E and s. Second, C1 associates into C2. Third, C1 has a sub-route Ri(C1) marked s or S and C2 has a sub-route Rj(C2) marked e or E (so Ri(C1) "contains" x and Rj(C2) "contains" y). Fourth, C1 has a sub-route Rk(C1) which associates into sub-route Rm(C2) of C2 ($i \leq k$, $m \leq j$). Then the region-to-region problem can be solved by adapting C1 and C2 as follows:



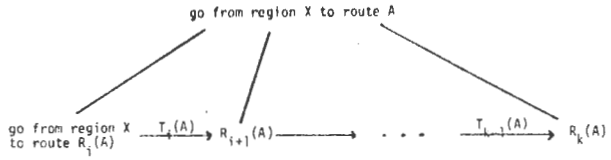

### 3.2.2 Solving Region-to-route Problems

Region-to-route (or route-to-region) problem expansion is slightly different from region-to-region expansion. It may not be necessary to find a new route if the region and route are close enough together but may suffice to refine the node slightly. For the problem "go from region x to route A" (and once again assuming x is defined by route X) the algorithm proceeds as follows:

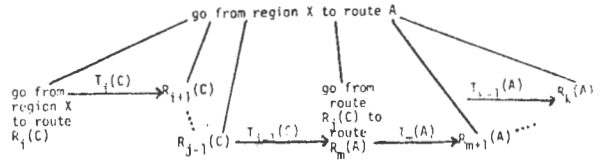
1. If A has no sub-routes then
   i. if X associates into A, then their common region must be very small (e.g. an intersection or a single block) so expand by fabricating Executor commands to get on route A from X. (These will be commands such as 'left turn' or 'follow signs to A'.) In the interests of plan brevity replace the problem node in the developing plan by these fabricated low-level commands.
   ii. otherwise treat as a region-to-region problem except that in any sub-nodes produced A is labelled as a route.

2. Perform the marking portion of the region-to-region algorithm starting at X only to see if some sub-route Ri(A) of A is marked by an s or S.

3. If so then expand as follows:




4. Otherwise treat as in region-to-region expansion except that Y becomes the set of sub-routes of A ({Ri(A)}). That is, attempt to find a route connecting X to one of the Ri, say Rm. The expansion results in:



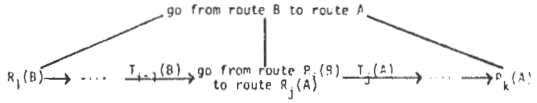

### 3.2.3 Solving Route-to-route problems

Route-to-route expansion is essentially region-to-route expansion with minor changes. For the problem "go from route B to route A", the algorithm proceeds as follows:

1. If B has no sub-routes, then treat as a region-to-route problem except that sub-nodes produced will have B labelled as a route.

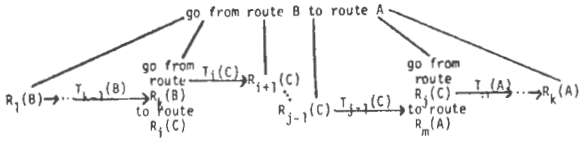
2. If A has no sub-routes then treat as a route-to-region problem except that sub-nodes produced will have A labelled as a route.

3. If some sub-route Ri(B) of B or one of its sub-routes associates into some sub-route Rj(A) of A or one of its sub-routes, then expand as follows:




4. Otherwise treat as a region-to-region problem except X becomes {Rn(B)} and Y becomes {Rn(A)} and a route connecting one of the {Rn(B)}, say Rk(B), to one of the {Rn(A)}, say Rm(A), should

be chosen. The problem expansion looks like:




## 4 Example

Figure 7 shows a plan produced by the Planner for the problem "go from Lenat@Kuipers to Reiter@Schubert". Assuming the routes in Figures 3 - 6 are in the Map and that ELMER is sitting on Lenat at the intersection of Lenat@Kuipers, the defining route for ELMER's start region is sub-route 82 of the Lenat route (Figure 4). Further assuming that ELMER wants to get exactly to the Reiter@Schubert intersection but doesn't care which direction he is facing when he arrives, the defining route for ELMER's end region is sub-route 122 of the Reiter route (Figure 5).

Indicated under each plan node in Figure 7 is information about the node's origin. Some nodes are copied unchanged from Map routes. Others must be further expanded using one of the various planning algorithms, in which case particulars of the algorithm are specified, and a sequence number indicating the order of expansion is provided.

Note the similarity of the plan to the Map routes from which it has been constructed. Once the plan has been suitably augmented and successfully executed by the Executor, it can be passed to the Map where information gleaned during execution (e.g. as to speed limits, directions, etc. discovered en route) can be abstracted up the plan hierarchy to the appropriate level of detail and where associations and inassociations can be made between this plan and existing Map routes. Both tasks are relatively straightforward, surprisingly even the latter since the plan was originally built out of Map routes whose associations are known. To illustrate, new sub-plan 300 can associate into old sub-route 16 (since sub-route 1 from which new sub-plan 300 has been built is so associated) and, conversely, old sub-route 16 can inassociate to new sub-plan 300. Once these two tasks have been accomplished, the recently created and executed plan can take its place as the newest addition to the pantheon of Map routes.

## 5 Conclusion

The most important contribution of this work is the unified approach to planning that it promotes. The system is not only capable of efficiently making plans but also executing these plans and acquiring the knowledge necessary to enhance its ability to plan. Next steps in the research involve more closely integrating planning and executing to allow re-planning after a plan goes awry; enhancing the Map and Planner to more extensively incorporate knowledge as to the relative merits of routes in order to allow the "best" plan to be constructed; figuring out how to allow the Map to infer "hypothetical routes" based on world knowledge (e.g. the reverse of an existing route may also be a valid route --- some of the Kuipers (1977) and Hayes-Roth and Hayes-Roth (1979) geographic knowledge structures may be appropriate here); and finally discovering whether other applications (e.g. process control?) can benefit from this approach.

## 6 Acknowledgements

We would like to thank Larry Reid, Hector Levesque, and Robin Cohen for their contributions to this research. The financial support of the National Sciences and Engineering Research Council is acknowledged.

## 7 References

Hayes-Roth and Hayes-Roth (1979). B. Hayes-Roth, F. Hayes-Roth, "A Cognitive Model of Planning", Cognitive Science 3, 4, Oct.-Dec.

Kuipers (1977). B.J. Kuipers, "Representing Knowledge of Large Scale Space", AI-TR-418, AI Lab., MIT, Cambridge, Mass.

McCalla and Schneider (1979). G.I. McCalla, P.F. Schneider, "The Execution of Plans in an Independent Dynamic Microworld", Proc. IJCAI-6, Tokyo, Japan.

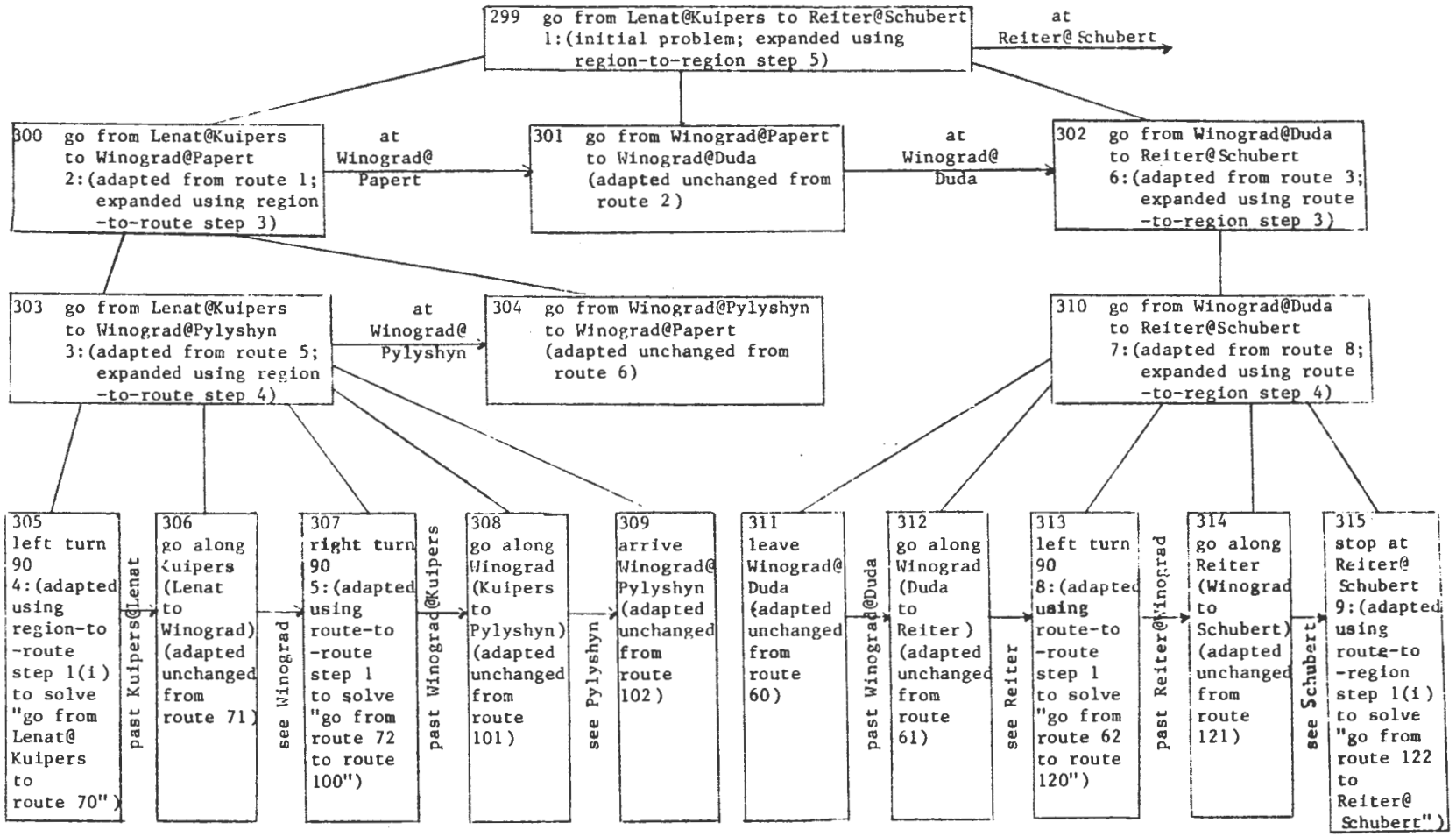
McCalla et al (1978). G.I. McCalla, P.F. Schneider, R. Cohen, H. Levesque, "Investigations into Planning and Executing in an Independent Dynamic Microworld", AI Memo 78-2, Dept. of Computer Science, U. of Toronto, Toronto, Ontario.

Reid and McCalla (1980). L. Reid, G.I. McCalla, "Planning and Executing in the Taxi Microworld: An Implementation Guide", Technical Report, Dept. of Computational Science, U. of Saskatchewan, Saskatoon, Saskatchewan (forthcoming).

Sacerdoti (1977). E. Sacerdoti, A Structure for Plans and Behaviour, Elsevier, New York.

Sacerdoti (1979). E. Sacerdoti, "Problem Solving Tactics", Proc. IJCAI-6, Tokyo, Japan.

Simon (1979). H. Simon, "Artificial Intelligence Research Strategies in the Light of AI Models of Scientific Discovery", Proc. IJCAI-6, Tokyo, Japan.




(note: associations and directions are left out for clarity)

Figure 7 - A Plan Constructed by the Planner

# What's The Point? *

Robert Wilensky

Computer Science Division
Department of EECS
University of California, Berkeley
Berkeley, California 94720

## 1.0 INTRODUCTION

Research into story understanding has become an important concern of Artificial Intelligence natural language understanding. (e. g., Charniak, (1974), Schank and Abelson (1977), Cullingford (1978), and Wilensky (1978)). Stories constitute a naturally occurring domain of coherent texts through which problems involving real world knowledge and contextual understanding may be conveniently studied. Thus stories provide the researcher with both an important set of problems and a realistic domain in which those problems may be productively pursued.

Attempts to produce computer story understanding systems have generated a number of interesting ideas, particular in the areas of knowledge representation and organization. However, many very basic questions still remain largely unaddressed. In particular, the idea of what actually constitutes a story has never been clearly delineated. In fact, most of the work on story understanding has really had little to do with stories per se. Rather, the objects that have been studied are coherent texts.

The difference between the two is that there are many texts that cohere but which are not recognizable as stories. As a result, the problems that have been considered so far are primarily concerned with how to find the connection between sentences of a text. While these are certainly important issues, the question of what exactly it is that a story is about has been left unasked.

As I will argue below, developing a theory of stories is not merely an academic exercise in providing a means for categorizing texts into stories and non-stories. There would be little point in developing the theory if that were its purpose. Rather, the theory of stories that I propose is intimately connected with some basic issues of language understanding, language generation, cognition and memory. Such a theory of stories is not only be related to these issues, but is as necessary for the construction of intelligent story understanding programs as are theories of inference and knowledge representation.

Before I attempt an exposition of this theory, it is important to point out how its thrust is different from another approach that ostensibly seems to have the same goal. This is the formalism known as a story grammar. Using a grammar to try to capture the notion of "storyness" seems to have been introduced to the AI/cognitive psychology literature by Rumelhart (1975). Since then, the notion of story grammars has been expanded theoretically by a number of researchers, and even used as the basis for a number of empirical studies (e. g., Mandler and Johnson (1977), Stein and Glenn (1977), and Thorndyke (1977)).

*This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.

Unfortunately, the story grammar concept is lacking in a number of ways that make it inadequate for its intended purpose. The chief problem is that story grammars purport to capture the idea of what a story is by trying to express the structure of a story text. My claim is that a theory of stories must be much more concerned with the content of a text than with its form. Moreover, when story grammars are examined closely, most of the story structure they aim to capture dissolves away, and they end up saying little more than that story is a coherent set of sentences.

A detailed critique of story grammars is found in Black and Wilensky (1979), and will not be repeated here. This paper is concerned with a theory of stories I have been developing based on text content. The theory is by no means complete. Nevertheless, some parts of it are well-formed enough to be presented, perhaps even refuted. In any case, the piece of the theory described below should provide a picture of what I believe a theory of stories needs to look like.

## 2.0 BACKGROUND

Most of the work that goes under the label of story understanding is really concerned with coherent text comprehension. Understanding a text involves finding the implicit connections between story sentences, and thus much of the work in this area addresses the problem of inference generation. In particular, the problem of representing knowledge needed for this task has played a large part in this work.

Most of this work is very basic. It has application not only to understanding narratives, but to other forms of cognitive processing, even non-linguistic ones. This is precisely its failure as a theory of stories. For example, consider the following "story," which is used by Schank (1977) to demonstrate his script idea:

(1) John went to a restaurant. The hostess seated John. The hostess gave John a menu. John ordered a lobster. He was served quickly. He left a large tip. He left the restaurant.

The point of this example is to demonstrate that knowledge about what typically goes on at a restaurant is needed to infer implicit events, such as John's eating the lobster. The notion of a script is introduced as a way to organize such knowledge.

The ability to understand utterances like (1) seems to underlie much of language processing. However, the existence of such knowledge structures and their utility in making a text coherent has little to do with the notion

of a story. It is unlikely that anything conforming to the content of a mundane knowledge structure will constitute a story. Problem is that they just too dull. No one will be heard telling story (1) to anyone else because there is little reason to believe anyone would have any reason to express it, nor is anyone likely to be amused upon hearing it.

Moreover, as was well understood at the time of their inception, rigid knowledge structures such as scripts are inadequate for much of the inference processing necessary to establish the coherence of a non-conforming text. Script-like knowledge structures reflect the repeated experience of mundane situations, and are directly useful for comprehending these situations. However, they are less clearly useful for processing situations that do not conform entirely to stereotypes.

## 2.1 PAM

PAM (Plan Applier Mechanism) is a program that can understand a number of short texts about situations it may never have encountered previously. PAM has knowledge about the kinds of plans and goals people have, and uses this knowledge to find explanations for the events described in a text. PAM can then paraphrase the text from the points of view of the different characters in the text, as well as answer questions about the text.

The following are some examples of stories PAM can currently process:

**********

Input text:

WILLA WAS HUNGRY. SHE PICKED UP THE MICHELIN GUIDE AND GOT INTO HER CAR.

Input: WHY DID WILLA PICK UP THE MICHELIN GUIDE?
Output: BECAUSE WILLA WANTED TO KNOW WHERE A RESTAURANT WAS.

Input: WHY DID WILLA GET INTO HER CAR?
Output: BECAUSE WILLA WANTED TO GET TO A RESTAURANT.

Input: WHAT WERE THE CONSEQUENCES OF WILLA PICKING UP THE MICHELIN GUIDE?
Output: THIS ENABLED WILLA TO READ THE MICHELIN GUIDE.

Output:

THE STORY FROM WILLA'S POINT OF VIEW:

I WANTED TO GET SOMETHING TO EAT, BUT I DIDN'T KNOW WHERE A RESTAURANT WAS. SO I PICKED UP THE MICHELIN GUIDE, AND I GOT INTO MY CAR.

**********

**********

Input text:

JOHN WANTED BILL'S BICYCLE. HE WENT OVER TO BILL AND ASKED HIM IF HE WOULD GIVE IT TO HIM. BILL REFUSED. JOHN TOLD BILL HE WOULD GIVE HIM FIVE DOLLARS FOR IT, BUT BILL WOULD NOT AGREE. THEN JOHN TOLD BILL HE WOULD BREAK HIS ARM IF HE DIDN'T LET HIM HAVE IT. BILL LET JOHN HAVE THE BICYCLE.

Input: WHY DID JOHN WALK OVER TO BILL?
Output: BECAUSE HE WANTED TO GET HIS BICYCLE.

Input: WHY DID BILL GIVE HIS BICYCLE TO JOHN?
Output: BECAUSE HE DIDN'T WANT TO GET HURT.

Input: WHAT HAPPENED WHEN JOHN ASKED BILL TO GIVE HIM BILL'S BICYCLE?
Output: BILL TOLD HIM THAT HE WOULDN'T GIVE IT TO HIM.

Input: WHAT HAPPENED WHEN JOHN THREATENED BILL?
Output: BILL GAVE HIM THE BICYCLE.

Output:

THE STORY FROM JOHN'S POINT OF VIEW:

I WANTED TO GET BILL'S BICYCLE. SO I WALKED OVER TO HIM AND I ASKED HIM TO HAND IT OVER. HE TOLD ME THAT HE WOULDN'T GIVE IT TO ME. SO I ASKED HIM TO SELL IT TO ME FOR FIVE DOLLARS. THEN HE TOLD ME THAT HE WOULDN'T. I TOLD HIM THAT IF HE DIDN'T HAND IT OVER THEN I WOULD BREAK HIS ARM. HE HANDED OVER HIS BICYCLE.

THE STORY FROM BILL'S POINT OF VIEW:

JOHN CAME OVER. HE ASKED ME TO GIVE HIM MY BICYCLE. WELL, WANTED HIM TO KNOW I WON'T GIVE IT TO HIM, SO I TOLD HIM THAT I WOULDN'T GIVE IT TO HIM. THEN HE OFFERED TO BUY IT FOR FIVE BUCKS. I WANTED HIM TO KNOW I WOULND'T GIVE HIM MY BICYCLE. SO I TOLD HIM THAT I WOULDN'T GIVE IT TO HIM. HE TOLD ME THAT IF I DIDN'T GIVE IT TO HIM THEM HE WOULD BREAK MY ARM. I DIDN'T WANT TO GET HURT. SO I GAVE HIM MY BICYCLE.

**********

PAM can understand a number of texts that are considerably more complicated than this one. These capabilities of the program will be examined later on. For the present, references to PAM will essentially be references to the part of PAM that can find explanations for events like those in the stories above. I will refer to this program (and the algorithm it embodies) as "naive PAM".

## 3.0 POINTS

PAM is a somewhat more flexible text understander than previous systems since it does not require that a text conform to a rigid structure. However, many of the same criticisms applicable to previous systems insofar as stories are concerned are just as applicable to naive PAM. PAM's "stories" may be less stereotyped than the texts other systems can process, but they are hardly any more reasonable. A relatively large number of inferences have to be generated to understand the Michelin guide example, but it is no more of a story than texts that conform to scripts. Once again, it is hard to imagine someone not in the field of natural language processing bothering to tell this story to someone else.

While goals and plans are important elements of real stories, the pursuit of a goal does not in and of itself make for good reading. For example, contrast the following two paragraphs:

(2) John loved Mary. He asked her to marry him. She agreed, and soon after they were wed. They were very happy.

(3) John loved Mary. He asked her to marry him. She agreed, and soon after they were wed. Then one day John met Sue, a new employee in his office, and fell in love with her.

Paragraph (2) is typical of the simple goal-based stories that can be understood by naive PAM: A character has a goal (wanting to marry Mary) generated by a theme (being in love with her) and pursues a plan (asking her) that results in the goal being fulfilled. While (2) is cogent enough, it is not a good story. Most readers would be surprised, for example, if they

were promised a story and were given paragraph (2). In spite of its coherent intentional structure, paragraph (2) seems much more like the setting of a story than a story itself.

In contrast, paragraph (3) seems much more promising. Although paragraph (3) does not appear to be a complete story either, it seems to get further along than paragraph (2) before it terminates. Here the reader probably expects the story to be continued with an elaboration of John's situation. In the case of story (2), it is much harder to guess what the story will be about.

What makes paragraph (3) more of a story than paragraph (2) is that paragraph (3) has a point to it. By a point I mean some element that invokes the interest of a reader. The point of a story is what the story seems to be about. For example, paragraph (2) does not seem to be about anything, while paragraph (3) is about a married person who falls in love with someone else.

Points play a significant role in story understanding because the main goal of the story reader is to determine the points of a story intended by the story teller. In a sense, points are intended to play a similar role for stories as meanings play for individual sentences. A reader often forgets the precise words and form that occurred in a sentence, and yet still retains the meaning of that utterance. In a story, a reader will often forget whole concepts and episodes, and yet remember what that story is about. A theory of points should be able to aid understanding and structure memory so that it behaves in this manner.

The role points play in understanding and memory is illustrated by examining a text that contains an entire story. The following story will be used for this purpose:

---

The Xenon Story

When John graduated college, he went job hunting and found a job with the Xenon corporation. John was well liked, and was soon promoted to an important position.

One day at work, John got into an argument with his boss. John's boss fired John and gave his job to John's assistant.

John had difficulty finding another job. Eventually, he could no longer keep up the payments on his car, and was forced to give it up. He also had to sell his house, and move into a small apartment.

Then one day, John saw a man lying in the street. Apparently, the man had been hit by a car and abandoned. John called a doctor and the man's life was saved. When he was well, the man called John and told him he was in fact an extremely wealthy man, and wanted to reward John by giving him a million dollars.

John was overjoyed. He bought himself a huge mansion and an expensive car, and lived out the rest of his life in the lap of luxury.

---

The interesting feature of the Xenon story is which parts of it are forgetable. For example, consider the following attempts at summarizing the story:

(1) John graduated college and went looking for a job. He found one at the Xenon corporation, and did well there.

(2) John graduated college. One day, he found a man who had been hit by a car, and called the police.

(3) A man had been hit by a car and left abandoned. Someone who used to work for the Xenon corporation called the police.

(4) A man whose life John saved gave him a lot of money. John bought himself a house and a car.

(5) John lost his job and came upon hard times. Then one day John helped a rich person in need, and was rewarded with enough money to last him a lifetime.

All these summaries describe events that occur in the original story. However, only paragraph (5) can be considered a reasonable summary. The other four all seem to miss the point of the story, while summary (5) encapsulates the gist of the story to the exclusion of all else.

This summary conforms to a description of the story's point. We will attempt to define the exact content and structure of this point more rigorously below. First, let us consider how the poignancy of these events affect story processing.

### 3.1 Points Structure Memory

The conceptual representation of a story in memory is structured according to its point content. That is, memory is hierarchical, and consists of at least the following levels:

1. At the highest level, a story is represented as a point or set of points that comprise the important parts of the text.

2. Beneath this level is a description of the actual events that comprise these points.

3. Beneath this is a level of events that connect up with the major events of the story but do not in themselves constitute points.

4. Finally, there is a level consisting of the actual words of the sentences used to express these ideas.

For example, in the Xenon story, the highest level describes that fact that John had a particular kind of problem. The level beneath this would describe the particular nature of the problem, e. g., that John couldn't keep up the payments on his car. The next level would include events like John spotting the hit and run victim. Then of course is the actual text.

### 3.2 Points Affect Processing

Since readers are presumably looking for a point as a text is being read, points often give rise to expectations, or predictions, about what will happen next in the text. For example, a reader of paragraph (2) does not find any point; if the story ends here the reader's expectations are not met and the reader is surprised by the stories pointlessness.

A reader of story (3), on the other hand, interprets the text as the beginning of a poignant episode. The reader of this story will also be surprised if the story terminates here. However, this time, the surprise is due to a point being introduced but not completed. That is, the reader would be equally confused if the

story were to continue and introduce other points without continuing to expand on the point already introduced. Without some predefined notion of point, a reader could not judge that one of these stories is better formed than the other.

Thus a reader must possess some notion of what constitutes a point in order to recognize one's occurence in a story. This knowledge about story points also plays a predictive role as well, since once a point has been referred to by a text, a reader must determine how the subsequent episodes in the story relate to that point.

While text comprehension is normally influenced by points, it is possible that points play a greater role in special kinds of processing. For example, skimming is a text comprehension technique in which the desire to process poignant information dominates the reader's concerns. We shall not be concerned with this sort of processing here, but with the role points play in ordinary comprehension. For a model of skimming as a goal-directed understanding process, see DeJong (1979).

### 3.3 Kinds Of Points

By definition, some points are liable to be idiosyncratic. However, enough points seem to be sufficiently pervasive to allow people to agree that the same point structure exists in the same texts. My goal here is to isolate these common points.

Many of these points have to do with human dramatic situations. A human dramatic situation is a sequence of goal-related events that contains some problem for a character. For example, in the Xenon story, the problem involves John losing his job and not being able to afford the lifestyle to which he had become accustomed.

Dramatic situations usually also involve solution components that describe how a problem is resolved. In the Xenon story, the solution in a fortuitous circumstance in which John saves the life of a rich person who subsequently rewards him.

The notion that problems form the basis of many stories was noted by a number of people, in particular, by Rumelhart (1976). Rumelhart uses the notion of a problem in his theory to refer to any situation involving a goal. The concept of a problem introduced here differs from Rumelhart's in that it requires a character to have trouble fulfilling his goal. For example, Rumelhart's theory does not make the distinction between story (2) and (3) above, although one story clearly appears to be better formed than the other.

In particular, the problematic dramatic situations that initiate story points usually involve some complex interactions between goals that can create difficulties for a character. For example, in the Xenon story, John's problem involves a relationship between his recurring goals of living in a certain style, and the state of having a job. In addition to situations involving recurring goals, frequently occurring dramatic situations include those in which there are a number of characters with opposing goals, and in which an individual has goals that are in conflict with one another.

The following is a short description of the goal relationships that play a significant role in creating poignant situations. As I have pointed out previously (Wilensky, 1978a and 1978b), these relationships are themselves quite complex, and a great deal of knowledge about them is needed by story understanders in order to understand the situations in which they appear. The following section is meant only to illustrate these relationships.

1. Goal Conflict

A goal conflict is a situation in which one character has several goals such that the fulfillment of one goal will preclude the fulfillment of the others. For example, consider the following story:

(4) John wanted to watch the football game but he had a paper due the next day. John decided to watch the football game. John failed Civics.

Story (4) is an instance of a goal conflict because John's goal of watching the football game may interfere with his other goal of writing his paper.

2. Goal Competition

Goal competition refers to those situations in which several characters' goals may interfere with one another. For example, the following story contains an instance of goal competition:

(5) John told Bill he would break his arm if Bill didn't give John his bicycle. Bill got on the bicycle and rode away.

John's goal of possessing Bill's bicycle cannot be fulfilled along with Bill's goal of preserving possession of the bicycle. If Bill succeeded in preserving possession of the bicycle, then John would have failed to fulfill his goal.

3. Goal Subsumption

Goal subsumption refers to a situation in which a character's plan is to achieve a state that will make it easier for a character to fulfill a recurring goal. For example, the following story contains an instance of goal subsumption:

(6) John was tired of frequenting the local singles' bars. He decided to get married.

In this story, John decides to get married in order to make it easier to achieve the goals he had been achieving previously by going to a singles' bar.

These particular goal relationships are important here because the situations to which they give rise account for a large class of story problems. That is, dramatic situations involve a difficulty in fulfilling a goal, and these difficulties often arise due to goal interrelations. In particular, goal relationships can give rise to these problems and associated solutions:

1. Goal Conflict - If a character is unable to resolve a goal conflict, then one of that character's goals may fail. Thus goal failure due to goal conflict, and attempts to resolve goal conflict both provide interesting story situations. In addition to attempts at resolution, a goal conflict may resolve itself spontaneously under a set of fortuitous circumstances.

2. Goal Competition - As with goal conflict, the existence of competitive goals implies that some character may have trouble fulfilling his goal. Interesting stories therefore exist involving goal failure due to goal competition, struggles against the plans of other characters (which we call anti-planning), attempts at easing the competition, and spontaneous resolution of the problem.

3. Goal Subsumption - Goal subsumption gives rise to dramatic situations when a subsumption state is terminated. For example, if John is happily married to Mary, and then Mary leaves him, all the goals subsumed by their relationship may now be problematic - John may become lonely, and miss his social interactions with Mary, for instance. Closely related to problems based on goal subsumption are those caused by the elimination of normal physical states. For example, becoming very depressed or losing a bodily function can give rise to the inability to fulfill recurring goals, and can therefore generate some interesting problems.

The resolution of goal subsumption termination involves establishing a new subsumption state to re-subsume the recurring goals.

## 4.0 GOAL RELATIONSHIP POINTS

Goal subsumption termination is a problem point component because previously subsumed goals become problematic. Goal conflict and goal competition endanger the fulfillment of some of a character's goals, and therefore generate dramatic impact. On the solution side, we have goal conflict resolution, goal abandonment, antiplanning, re-subsuming subsumption states, and spontaneous conflict and competition removal.

However the dramatic nature of goal relationships is not independent of how these relationships are presented in a text. For example, consider the following misuse of a potentially poignant goal relationship:

(7) John lost his job. Then he found another one.

This is not a particularly dramatic situation. It contains an instance of goal subsumption termination (John losing his job) and a solution to the problem this creates (John getting a new job). Nevertheless, (7) hardly qualifies as an interesting story.

The problem with (7) is that it contains the cause of the problem, the termination of a subsumption state, but no description of the problem itself. Contrast (7) with the Xenon story given at the beginning of this paper. John also lost his job in that story, but the situation contains considerably more dramatic impact. In the Xenon story we are given a description of John's problem state. He could no longer afford all the things he had become used to. Since the problem is spelled out in this story, its dramatic effect is more fully realized.

Thus the mere appearance of a problematic goal relationship does not guarantee its poignancy. The problem must appear in a form that spells out its implications. I call these forms point prototypes. A point prototype is a kind of distillation of the dramatic element of which the goal relationship is a part. The Xenon story above will serve to illustrate such a prototype.

The problem for John in the Xenon story is caused by a goal subsumption state terminating. To make this poignant, the story uses the problem point prototype in Figure 2 to fill out the circumstances of the problem.

Figure 2
Goal Subsumption Termination Prototype

1. Subsumption state
2. Cause of termination event
3. Problem state description
    1. Unfilled precondition
    2. Problematic goals
    3. New goal (optional)
    4. Emotional reactions (optional)

That is, to use subsumption state termination as a problem point, first state the subsumption state, followed by the cause of termination event. Then describe the problem state itself by listing the goals that are no longer subsumed; the goal of re-establishing a subsumption state may be stated also, along with any emotional reactions to the termination.

In the Xenon story, this prototype is instantiated as is shown in Figure 3.

Figure 3
Instantiated GST Prototype

1. Subsumption state - John has job.
2. Cause of termination event - Boss fires John.
3. Problem state description
    1. Unfilled precondition - John doesn't have enough money.
    2. Problematic goals - Maintaining car and house.
    3. New goal - John wants to resubsume these goals.
    4. Emotional reactions - not explicitly stated.

This problem is resolved in the story through a very common solution point called Fortuitous Circumstances. Spontaneous goal conflict resolution and external goal competition removal are also instances of this solution point component, which is shown in Figure 4.

Figure 4
Fortuitous Circumstance Solution Prototype

1. Undesired state
2. Fortuitous event
    1. Incidental action
    2. Fortuitous outcome
    3. New state
3. State consequence description

This solution prototype is instantiated in the Xenon story as Figure 5 shows.

Figure 5
Instantiated FC Solution Prototype

1. Undesired state - John doesn't have enough money.
2. Fortuitous event
    1. Incidental action - John saves rich man.
    2. Fortuitous outcome - Rich man gives John money.
    3. New state - John is rich.
3. State consequence description - John is happy and gets lots of possessions.

4.0.1 Some More Solution Point Components

Solution point components and their associated prototypes have not yet been analyzed in as much detail as the problem components have been. However, in addition to the fortuitous circumstances solution given above, several other solution point components seem to be common.

One such solution is called "More Desperate Measures". In this point, a problem is attacked by some plan that is normally not considered because of its high risk. Because of this risk, More Desperate Measures solutions tend to generate goal conflicts their user, thus creating another problem point component for the story. For example, in the Xenon story, after John loses his job, he might decide to rob a bank to get some money. Robbery entails a number of risks, so the use of this plan would create a goal conflict for John between his desire to have money and to preserve his well-being. This point would then be developed further in the story.

Overcoming a Limitation is another solution point seen with some frequency. This case can occur when a problem is based in part on a character's inability or lack of courage. Here the character attempts to overcome his personal limitation or see the error of his ways in order to resolve a problematic situation. For example, a typical fairy tale type plot might involve a character who is a subject of ridicule by his piers because he is a coward, and then overcomes his cowardess in some heroic deed.

5.0 CURRENT STATE OF PAM

As was mentioned previously, the naive explanation algorithm fails to find proper explanations for events in stories involving goal relationships. However, a more sophisticated version of PAM has been implemented that possesses knowledge about the goal relationships described above. PAM can use this knowledge to infer explanations for events in many complex goal relationship situations.

The following simple examples illustrate some of the situations involving goal relationships that PAM can understand:

**********

Goal Subsumption:

Input text:

JOHN AND MARY WERE MARRIED. THEN ONE DAY, JOHN WAS KILLED IN A CAR ACCIDENT. MARY HAD TO GET A JOB.

Input: WHY DID MARY NEED EMPLOYMENT?
Output: JOHN DIED AND SO SHE NEEDED A SOURCE OF MONEY.

**********

Pam infers that John's death terminates a subsumption state for Mary, and that she may seek to replace it. PAM uses this inference to infer that the explanation behind Mary's goal of getting a job.

**********

Goal Conflict:

Input texts:

WILMA WANTED TO HAVE AN ABORTION. WILMA WAS CATHOLIC. WILMA CONVERTED FROM CATHOLICISM TO EPISCOPALIANISM.

WILMA WANTED TO HAVE AN ABORTION. WILMA WAS CATHOLIC. WILMA WENT TO A ADOPTION AGENCY.

FRED WANTED TO TAKE HIS GUN HUNTING. FRED WANTED WILMA TO HAVE A GUN AT HOME. FRED ONLY HAD ONE GUN. FRED BOUGHT ANOTHER GUN.

**********

In the first two stories, PAM detects a conflict between Wilma's goal of having an abortion and her inferred goal of not having an abortion because she is Catholic. In the first story, PAM infers that Wilma resolved the conflict by changing the circumstance that gives rise to one of her goals, and fulfilled the other (i. e., she decided to have the abortion). In the next case, PAM infers that Wilma abandoned her goal of having an abortion because it menat less to her than violating her religious beliefs.

The third story is a goal conflict based on a resource shortage. Here PAM infers that Fred bought another gun so he could take one with him and leave one at home.

**********

Goal Competition:

Input text:

JOHN WANTED TO WIN THE STOCKCAR RACE. BILL ALSO WANTED TO WIN THE STOCKCAR RACE. BEFORE THE RACE, JOHN CUT BILL'S IGNITION WIRE.

Input: WHY DID JOHN BREAK AN IGNITION WIRE?
Output: BECAUSE HE WAS TRYING TO PREVENT BILL FROM RACING.

**********

This story contains an instance of a goal competition situation involving anti-planning. PAM explains John's action as part of a plan to undermine Bill's efforts by undoing a precondition for Bill's plan.

PAM also have been given some knowledge about poignancy. In particular, PAM knows about goal subsumption termination problem components, and fortuitous circumstance solution points. With this knowledge, pam can now understand the following version of the Xenon story:

**********

JOHN GRADUATED COLLEGE. JOHN LOOKED FOR A JOB. THE XENON CORPORATION GAVE JOHN A JOB. JOHN WAS WELL LIKED BY THE XENON CORPORATION. JOHN WAS PROMOTED TO AN IMPORTANT POSITION BY THE XENON CORPORATION.
JOHN GOT INTO AN ARGUMENT WITH JOHN'S BOSS. JOHN'S BOSS GAVE JOHN'S JOB TO JOHN'S ASSISTANT. JOHN COULDN'T FIND A JOB. JOHN COULDN'T MAKE A PAYMENT ON HIS CAR AND HAD TO GIVE UP HIS CAR. JOHN ALSO COULDN'T MAKE A PAYMENT ON HIS HOUSE, AND HAD TO SELL HIS HOUSE, AND MOVE TO A SMALL APARTMENT.
JOHN SAW A HIT AND RUN ACCIDENT. THE MAN WAS HURT. JOHN DIALED 911 THE MAN'S LIFE WAS SAVED. THE MAN WAS EXTREMELY WEALTHY, AND REWARDED JOHN WITH A MILLION DOLLARS. JOHN WAS OVERJOYED. JOHN BOUGHT A HUGE MANSION AND AN EXPENSIVE CAR, AND LIVED HAPPLY EVER AFTER.

**********

In addition to the many inference that are made to understand this story, PAM also recognizes that John's losing his job is an instance of a Goal Subsumption Termination problem, and that the hit and run victim rewarding John is an instance of a Fortuitous Circumstance solution to this problem. This representation could then be used by a summarization program to produce a summary that included only the events of John losing his job, the problems this caused, John's saving the rich man, and the rich man rewarding him (A summarization component that actually performs this taks in presently under construction. Although it has not yet been completed, it does not appear to be problematic, since all the information it needs is present in the structures PAM already produces).

## 6.0 SUMMARY

Stories constitute a subset of coherent natural language texts. For texts to be stories, they must be poignant in addition to being coherent. This point structure of a story serves to organize the representation of a story in memory so that more important episodes are more likely to be remembered than trivial events. Points also serve to generate expectations about what will happen next in a story, since a story reader is looking for the point of a story as the text is being read.

An important class of story points deals with human dramatic situations, and these most often contain a set of interacting goals that create difficulties for a character. A taxonomy of these goal relstionships and the situations they give rise to is useful for detecting a point of a story, as well as for establishing its coherence as a text. When a goal relationship situation occurs as a problem point component, it will occur as part of a point prototype. These prototypes specify those aspects of the situations that should be mentioned in order to produce a dramatic effect.

The notion of a story point competes with the ides of story grammars as a way to characterize story texts. The story grammar approach attempts to define a story as a text having a certain form, while the story point idea defines a story as a text having a certain content. The form of a story is viewed here as being a function of the content of the story, not a reasonably independent object. Understanding stories, then, is not so much a question of understanding the structure of a text, but of understanding the point of what the text is about.

## References


1] Black, J. B. and Wilensky, R. (1979). An evaluation of story grammars. Cognitive Science, vol. 3, no. 3.

2] Charniak, E. (1972). Towards a model of children's story comprehension. AI TR-266, MIT.

3] Cullingford, R. E. (1978). Script Application: Computer Understanding of newspaper stories. Yale University Research Report #116.

4] DeJong, G. F. (1979). Skimming stories in real time: An experiment in integrated understanding. Yale University Research Report #158.

5] Kintsch, W., and Van Dijk, T. A. Recalling and summarizing stories. Language, 40, 98-116.

6] Mandler, J. M. and Johnson, N. S. Remembrance of things parsed: Story structure and recall. Cognitive Psychology, 9, 111-151.

7] Minsky, M. (1974). A framework for representing knowledge. MIT. AI Memo No. 306.

8] Rumelhart, D. E. (1975). Notes on a schema for stories. In D.G. Bobrow and A. Collins (eds.) Representation and Understanding: Studies in Cognitive Science. Academic Press, New York.

9] Rumelhart, D. E. (1976). Understanding and Summarizing brief stories. Center for Human Information Processing Technical Report No. 58. University of California, San Diego.

10] Schank, R. C. and Abelson, R. P. (1977). Scripts, Plans, Goals, and Understanding. Lawrence Erlbaum Press, Hillsdale, N.J.

11] Schank, R. C. and Wilensky. R. (1978). A Goal Directed Production System for Story Understanding. In D. A. Waterman and F. Hayes-Roth (Eds.), Pattern-directed Inference Systems. Academic Press, New York.

12] Schank, R. C. and Yale A. I. Project (1975). SAM -- A story understander. Yale University Research Report #43.

13] Stein, N. L. and Glenn, C. G. (1977). An analysis of story comprehension in elementary scholl children. In R. Freedle (Ed.) Multidisciplinary perspectively in discourse comprehension. Lawrence Erlbaum Associates, Hilldale, New Jersey.

14] Thorndyke, P. (1977). Cognitive Structures in Comprehension and Memory of Narrative Discourse. Cognitive Psychology, 9:88-110.

15] Wilensky, R. (1978a). Why John married Mary: Understandind Stories Involving Recurring Goals. Cognitive Science, vol. 2 no. 3.

16] Wilensky, R. (1978b). Understanding goal-based stories. Yale University Research Report #140.

# Speech Acts and the Recognition of Shared Plans

Philip R. Cohen
Center for the Study of Reading
University of Illinois &
Bolt Beranek and Newman, Inc.
Cambridge, MA

Hector J. Levesque
Dept. of Computer Science
University of Toronto
Toronto, Ontario

## Introduction

The purpose of this paper is to simplify Perrault, Allen, and Cohen's [1,2,9,10,19,20] plan-based theory of speech acts by revealing an important redundancy -- illocutionary acts. We show that illocutionary act definitions can be derived from more basic statements describing the recognition of shared plans -- plans based on the shared beliefs of the planner and some intended recognizer. Eliminating the redundancy is important for competence models of speech acts [10,19], but maintaining and exploiting it may be useful for computational and linguistic models [1,11,32] especially for those dealing with the "short-circuiting" of certain implicatures [4,18,32]. Our primary interest here is in competence models.

A plan-based theory of speech acts specifies that plan recognition is the basis for inferring the illocutionary force(s) of an utterance. The goal of such a theory is to construct a plan generation and recognition formalism that treats communicative and non-communicative acts uniformly. Such a theory should therefore state the communicative nature of an illocutionary act as part of that act's definition. A reasoning system would then not have to employ special knowledge about communicative acts; it would simply attempt to achieve its goals.

## Communication and the recognition of shared plans

Communication is intimately tied to plan-recognition. Grice [14] showed that "simple" recognition of intention(1) as might be performed by an unseen observer (cf. [24,31]) is insufficient as a basis for defining communicative acts. Instead he argued that speakers must plan for hearers to recognize their plans, and hearers must recognize the plans they were intended to recognize. Unifying Grice's analysis with Austin's [3], Searle [27,28] proposed that a speaker who is performing a speech act, such as a request, must intend to produce the effect of that action (to get the hearer to want to perform the requested act) by means of getting the hearer to recognize the speaker's intention to produce it. It was on this basis that Perrault and Allen [1] developed a scheme for recognizing indirect speech acts.

This research was supported primarily by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under Contract N00014-77-C-0378, and also, in part, by the National Research Council of Canada.

(1) For this paper, "intention" and "plan" should be considered synonymous

However, Schiffer [26] has argued that, to avoid counterexamples based on deception, the Gricean program (and its amendments [30,27]) produce an infinite regress of intending that one recognize an intention. To avoid this difficulty, he claims that the recognition of intention must be mutually believed. In other words, in order to communicate, speakers must make their plans shared or public knowledge.

In view of this problem, Perrault and Allen have suggested that their model be reformulated in terms of mutual beliefs. Since we are proposing such revisions, we shall discuss the essentials of their scheme.

## Allen and Perrault's Model

Building on prior attempts to link speech acts to plans [5,6,7,9,10,24], Allen and Perrault proposed two levels of speech act operators: surface and illocutionary. Illocutionary operators, e.g., REQUEST, are defined by stating propositions as preconditions, bodies, and effects with the understanding that:

a) preconditions are necessary for the succeasful "execution" of the act (or procedure) described by the operator;

b) effects are conditions that become true after the execution, and

c) the body is "a set of partially ordered goal states that must be achieved in the course of executing the procedure." [19, p.23].

Searle's recognition of intention" condition on speech acts is incorporated by defining an illocutionary operator's body to be "hearer believes speaker wants E" (abbreviated HBSW(E)), where E is the operator's effect. So, given the above understanding of operators, the illocutionary acts' operator's body needs to be achieved in the course of executing the operator. The standard ways of achieving them are by surface operators.

The classification of an utterance as a surface operator depends on the utterance's mood -- declaratives become S-INFORMs, imperatives become S-REQUESTs, and questions become S-REQUESTs to INFORM. Surface operators are considered to be primitive -- they represent what agents actually perform -- and consequently have no bodies. Their effects are defined to match the corresponding illocutionary operators' bodies -- i.e., HBSW(E). Thus, the standard way of achieving the body of a REQUEST is via an S-REQUEST. However, different combinations of surface act and propositional content can ultimately yield the same effect.

Mediating between the effects of surface operators and the bodies of the illocutionary ones is a set of plan-recognition inferences. Generally speaking, the inferences take the form: "the agent wanted P to be true because that would enable him to do A (precondition/action inference), which would result in E (Action/effect), which is a means of achieving B (body/action). The agent is then regarded as having wanted [to do] A, E, and [to do] B.

The inference process begins by observing an act and then assuming it was intentional -- the agent wanted to do it. The application of the action/effect inference speech act operator thus results in a proposition of the form: HBSW(HBSW(E)). Perrault and Allen supply rules for inferring new propositions E′ such that HBSW(HBSW(E′)). Each such E′, inferred by these intended plan recognition rules, is regarded as having been communicated, in the Gricean sense.

Illocutionary act identification occurs when the body/action inference applies to the embedded HBSW(E) proposition, yielding, for instance, HBSW( REQUEST(S,H,B)). If the body/action inference occurs immediately after the expansion of a surface act, then a literal interpretation has been found. If there are intervening inferences, an indirect interpretation has been inferred.

Their uncovering of the inferences needed to arrive at indirect interpretations is the key accomplishment. But once those inferences are known, formal (and perhaps computational) models need not recognize illocutionary operators in order to communicatively infer their effects. Since, for their model, illocutionary force is being discovered by a hearer motivated to recognize the speaker's plan in order to facilitate it, the effects are all that is needed. We suggest, then, that the body/action inference collapses two distinct kinds of inference processing -- means/end reasoning and summarizing. The latter has not been shown to be essential to helpful plan recognition.

To demonstrate this point, Perrault and Allen's model will be elaborated upon in two ways:

1) To relate an illocutionary operator's body and effect, a plan will be stated that produces the effect once the body is achieved.

2) To capture the communicative nature of illocutionary acts more accurately, the steps of those plans should be mutually believed.

Surface speech act operators will be redefined to produce mutual beliefs about the speaker's goals, much as Perrault and Allen suggested (cf. Clark and Marshall's [8] analysis discussion of situations producing mutual beliefs). Once these steps are taken, the body/action inference will be unnecessary and illocutionary operators will reduce to redundant theorems.

## The Formalism

This section formalizes actions and plans, in conjunction with a planner's beliefs and desires. The style of formalization owes much to the literature on axiomatic specification of programming languages, and to Moore[17]. We do not intend to give the impression that a complete language with proof and model theories is lurking somewhere offstage. Although we will describe the formalism in terms of axioms, rules of inference and possible world states, it should be understood that these are intended to be more suggestive than definitive and that the formalism itself remains a topic of on-going research.

The formalism is a language with expressions of various types formed from primitive elements through rules of composition. Among the types of expressions we will discuss here are logical expressions (or wffs), action terms and terms denoting agents. For the remainder of this section, we will use "p", "q" and "r" as meta-variables ranging over wffs, "a" and "b" ranging over action terms and "x" and "y" ranging over agent terms. In addition, we will use "|-" as a predicate over wffs holding when the wff is a theorem.

By an action, we mean something an agent can do to change the state of the world. For example, the action term

(GIVE x o)

denotes the giving of the object denoted by "o" to the person denoted by "x". Notice that the action term does not mention the agent involved. This allows actions to be combined into more complex ones without having to fuss over the resulting agent. Examples of complex acts are

(IF p a b) a conditional action
(SEQ a1 ... ak) a sequence of actions
(WHILE p a) an iterated action

Among the actions required for communication, we assume

(S-INFORM x p) saying to x that p is true
(S-REQUEST x p) asking x to make p be true

Among the wffs, we assume the usual connectives

(IMPLY p q) (NOT p)
(AND p1 ... pk) p = q
(FORALL v p)

In addition, there are wffs pertaining to communication

(ATTEND x y) true iff x is attending to y
(BEL x p) true iff p follows from what x believes [15,16]
(WANT x p) true iff p follows from

what x wants

Note that just because a BEL or WANT is true does not mean that the agent involved actively believes or wants the proposition in question. All that can be said is that in every world state that is consistent with what the agent believes, the second argument to BEL will be true. One particular kind of wff peculiar to actions is

```
(RESULT x a p)
```

which is true iff "p" is true in the world state resulting from the execution of "a" by "x" (or "a" does not terminate).

The behaviour of the expressions is governed by the axioms and rules of inference of the formalism. For example, action terms are specified using RESULT as in

```
|- (IMPLY (OWN x o) (RESULT x
                         (GIVE y o)
                            (OWN y o)))
```

The composite actions can be treated much like the axiomatic specification of programming language constructs. The IF rule, for example is

```
|- (IMPLY (AND (KNOWIF x p)
               (IMPLY p (RESULT x a q))
                    (IMPLY (NOT p)
                    (RESULT x b q)))
          (RESULT x (IF p a b) q))
```

where

```
|- (KNOWIF x p) = (OR (KNOW x p)
                       (KNOW x
                          (NOT p)))
```

and

```
|- (KNOW x p) = (AND p (BEL x p))
```

Similarly, the rule of consequence becomes

```
If |- p   then  |- (RESULT x a p)
```

Note that this must be a rule of inference in that the corresponding axiom (as an implication) cannot be a theorem. A related notion to this rule is the wff

```
(CAUSE x p q)
```

governed by the axiom

```
|- (CAUSE x p q) = (FORALL a/ACTION
          (IMPLY (RESULT x a p)
                   (RESULT x a q)))
```

so that a CAUSE is true iff anything that "x" does to bring about "p" also results in "q" being true. In other words, making "p" true makes "q" true.

Of crucial importance to the definition of speech acts, is the concept of mutual belief (or MB) governed by

```
If |- p then |- (MB x y p)
```

and

```
|- (MB x y p) = (BEL x (AND p
                            (MB y x p)))
```

The axiom states that mutual belief is equivalent to an infinite conjunction of beliefs in that, allowing that

```
|- (BEL x (AND p q)) = (AND (BEL x p)
                              (BEL x q))
```

then the following are implied by a mutual belief:

```
(BEL x p)
(BEL x (BEL y p))
(BEL x (BEL y (BEL x p)))   ...(1)
```

Given the notion of mutual belief, we can now state the two rules governing two primitive communication acts, S-INFORM and S-REQUEST:

```
|- (IMPLY (MB x y (ATTEND y x))
          (RESULT x (S-INFORM y p)
             (MB y x (WANT x (BEL y
                              (BEL x p]

|- (IMPLY (MB x y (ATTEND y x))
          (RESULT x (S-REQUEST y p)
             (MB y x (WANT x (BEL y
                              (WANT x p]
```

When discussing the behaviour of goal-directed agents, a useful concept is that of an agent being able to bring about some state of affairs that he wants:

```
|- (CAN x p) = (EXISTS a/ACTION
                   (KNOW x
                      (RESULT x a p)))
```

Note that this is an example of "quantifying in" in that it is not sufficient for "x" simply to know the existence of an action that results in "p", he must also know what action it is. On the other hand,

```
(BEL y (CAN x p))
```

could be true without "y" knowing how "x" will achieve "p" since, in this case, the quantifier is within the belief context. Given this characterization, we now turn our attention to plans, which, loosely speaking, are simply proofs that, given some set of beliefs, an agent is able to achieve some goal. More formally, the definition is

---

(1) Schiffer's [26] definition of mutual belief also includes an infinite conjunction starting from (BEL y p). Since we shall only be concerned about one person's point of view, we only deal with beliefs about mutual beliefs, which reduce to the above.

A plan for agent "x" to achieve some goal "q" is an action term "a" and two sequences of wffs "p0", "p1", ... "pk" and "q0", "q1", ... "qk" where "qk" is "q" and satisfying

```
1.  |- (BEL x (IMPLY p0
                   (RESULT x a q0)))
2.  |- (BEL x (IMPLY pi
                  (CAUSE x qi-1 qi)))
       i=1,2,...k
```

In other words, given a state where "x" believes the "pi", he will believe that if he does "a" then "q0" will hold and moreover, that anything he does to make "qi-1" true will also make "qi" true. Consequently, a plan is a special kind of proof that

```
|- (BEL x (IMPLY (AND p0 ... pk)
                  RESULT x a q)))
```

and therefore, assuming that

```
|- (IMPLY (BEL x p) (BEL x
                       (BEL x P)))
```

and

```
|- (IMPLY (BEL x (IMPLY p q))
           (IMPLY (BEL x p)
                  (BEL x q)))
```

a plan is a proof that

```
|- (IMPLY (BEL x (AND p0 ... pk))
                   (BEL x (CAN x q)))
```

Notice that the assumptions "pi" may be simplified in a plan in that if we have that

```
|- (BEL x (IMPLY p (RESULT x b
                     (AND p0 p1 ... pk))))
```

then we have a reduced plan for "x" to achieve "q" since

```
|- (BEL x (IMPLY p (RESULT x
                       (SEQ b a) q)))
```

This process can, of course, be iterated on the new assumptions. (Since action "b" achieves <u>all</u> the prerequisites, the "non-linearity" problem [21] remains.)

Among the corollaries to a plan are

```
|- (BEL x (IMPLY (AND p0 ... pi)
                  (RESULT x a qi))) i=1,...k
```

and

```
|- (BEL x (IMPLY (AND pi ... pj)
                  (CAUSE x qi-1 qj)))
           i=1,...k  j=i,...k
```

There are two main points to be made about these corollaries. First of all, since they are theorems, the implications can be taken to be believed by the agent "x" in every state. In this sense, these wffs express general methods believed to achieve certain effects provided the assumptions are satisfied. The second point is that these corollaries are in precisely the form that is required in a plan and therefore can be used as justification for a step in a future plan in much the same way a lemma becomes a single step in the proof of a theorem.

We therefore propose a notation for describing many steps of a plan as a single summarizing operator (akin to MACROPs in STRIPS [11]). An operator consists of a name, a list of free variables, a distinguished free variable called the agent of the operator, an effect which is a wff, a optional body which is either an action or a wff and finally, an optional prerequisite which is a wff. The understanding here is that operators are associated with agents and for an agent "x" to have an operator "u", then there are three cases depending on the body of "u":

1. If the body of "u" is a wff, then

```
|- (BEL x (IMPLY prerequisite
            (CAUSE agent body effect)))
```

2. If the body of "u" is an action term, then

```
|- (BEL x (IMPLY prerequisite
     (RESULT agent body effect)))
```

3. If "u" has no body, then it is simply an action and

```
|- (BEL x (IMPLY prerequisite
              (RESULT agent u effect)))
```

An example of this last kind of operator is the action GIVE, described above, which becomes the operator

```
[GIVE y o] agent: x
    effect: (HAVE y o)
    prereq: (HAVE x o)
```

One thing worth noting about operators is that normally the wffs used above

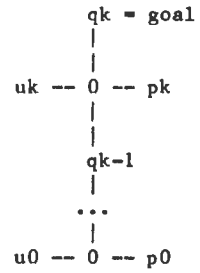
```
|-(BEL x (IMPLY prerequisite ...))
```

will follow from the more general wff

```
|-(IMPLY prerequisite ...)
```

as in the case of the GIVE example. However, this need not be the case and different agents could have different operators (even with the same name). Saying that an agent has an operator is no more than a convenient way of saying that the agent always believes an implication of a certain kind.

Before considering some examples of operators and their use in plans, we introduce the notation for describing plans.




where the "pi" and the "qi" are as before and the "ui" are the operators justifying the transition given "pi" from "qi-1" to "qi". In the simplest case, "pi" will be the prerequisite of "ui", with "qi-1" and "qi" the body and effect respectively. More generally, we need only require that

```
|- (BEL x (IMPLY pi prerequisite))
|- (BEL x (IMPLY qi-1 body))
|- (BEL x (IMPLY effect qi))
```

to satisfy the definition of a plan.

Operator Definitions

Given the above understanding of operator definitions, we present those operator schemas needed for our derivation of REQUEST. The first argument in the parameter list for a schema will be the agent.

[CAUSE-TO-WANT x y p]

```
effect: (WANT y p)
body:   (BEL y (WANT x p))
prereq: (AND  ~(WANT y ~p)
          (HELPFUL y x))
```

Provided y doesn't want NOT(p), and y thinks she is feeling helpfully disposed towards x, then getting y to believe that x wants p will get y to want p. Though this may be one way to influence someone's goals, more generally, one would like to state "y is given a reason for wanting p".

The SHARED-RECOG operator describes shared recognition of the agent's goals:

[SHARED-RECOG x y p q]

```
effect: (MB y x (WANT x q))
body:   (MB y x (WANT x p))
prereq: (MB y x (CAUSE x p q))
```

Of course, not every action produces mutual beliefs about someone's goals. Usually, the two parties must be mutually aware of the other's presence. However, once it is shared knowledge that x wants p, if its mutually believed that anything x does to make p true makes q true, then it will be mutually believed that x wants q. Clearly, we are exploiting the "follows from what the agent wants" interpretation of WANT here -- an agent wants all the inevitable results of his wants. Since this interpretation is currently forced on us by our formal tools, and since we want to formalize shared plans, we WANT this interpretation.

The next operator provides for private recognition of the agent's goals. It is similar to Perrault and Allen's [19] Plan-Deduce operator.

[PRIVATE-RECOG x y p q]

```
effect: (BEL y (WANT x q))
body:   (BEL y (WANT x p))
prereq: (BEL y (BEL x (CAUSE x p q)))
```

PRIVATE-RECOG should appear in plans when SHARED-RECOG is inappropriate, for instance when the conditions implying CAUSE statements are not mutually believed. Lack of shared knowledge can arise because of third parties (e.g., someone tells you what I want), because of the modality of communication (e.g., telephone conversations), or because one of the parties is an unseen observer.

The operator ACHIEVE models getting someone else to make p true.

[ACHIEVE x y p]

```
effect: p
body:   (WANT y p)
prereq: (CAN y p)
```

All that is required is that y know of some action resulting in p (x does not have to know which action that is). Then, simply by getting y to WANT p will CAUSE p to hold. Of course this idealization ignores the possibility of y's being unable or unwilling to actually perform the action. Future versions of CAN, using Moore's [17] RES modal operator, may ensure that y can also perform the action.

To allow for another way of influencing someone's goals, we define:

[FORCE-TO-WANT x y p]

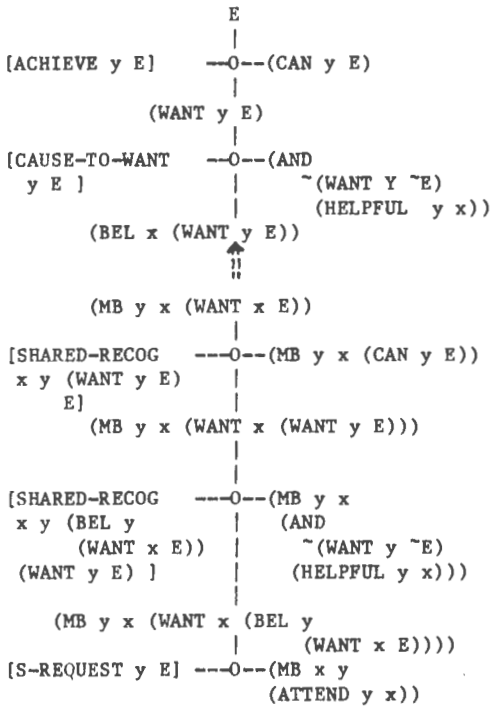
```
effect: (WANT  y p)
body:   (BEL y (WANT x p))
prereq: (BEL y (HAS-AUTHORITY-OVER x y))
```

The semantics of HAS-AUTHORITY-OVER (interpreted as x has authority over y) could be stated by filling out an organizational chart, or determining the status relationships between the parties.

Finally, the last operator we shall need is S-REQUEST, as defined earlier, to produce mutual beliefs about the speaker's goals. The prerequisite is that it be mutually believed between x and y that y is attending to x. (Note the order of x and y -- x must actually believe y is attending.) A crucial but as yet unanalyzed condition on classifying an utterance as an S-REQUEST to some particular hearer H is that it be mutually believed between the speaker, S, and H, that H is the intended addressee. This condition is not always satisfied, since some computer systems are conceptualized as "overhearing" (e.g., Genesereth's [13] ADVISOR).

A Plan

The following is x's plan to achieve E:




Given the individual operators and the interpretation of operators as theorems, the plan itself should be relatively self-explanatory. The prerequisites of the SHARED-RECOG operators shown imply those necessary for each individual step. For instance, since all theorems are mutually believed:

```
|-(BEL x (MB x y [IMPLY (CAN y E)
                       (CAUSE x (WANT y E)
                                E],
```

therefore

```
|- (BEL x (IMPLY [MB y x (CAN y E)]
                 [MB y x (CAUSE x
                          (WANT y E) E)]))
```

the precondition of

(MB y x (CAN y E)) is shown. We have made one such implication explicit in the diagram -- the one marking the transition from shared to private beliefs.

Summarizing the plan

Various portions of the plan can now be summarized. First of all, consider the summary operator REQUEST:

[REQUEST x y E]

```
effect: (MB y x (WANT x E))
body:   (MB y x (WANT x (BEL y
                        (WANT x E))))
prereq: (AND (MB y x (CAN y E)
             (MB y x  ~(WANT y ~E))
             (MB y x  (HELPFUL y x))
```

If the prerequisite holds, any action making the body true achieves the effect. The propositions in the plan not summarized by this operator are achieved by virtue of y's private beliefs. The decision to include illocutionary or perlocutionary effects as part of some operator cannot be made solely on formal grounds. Also, notice that the third argument in the REQUEST schema is a proposition and not an action. While it would be desirable to derive a REQUEST to use an action, the formalism forbids its use since WANT takes a proposition as its argument.

We can also define other operators from this same plan. For instance,

[COMPLY y x E]

```
effect: E
body:   (MB y x (WANT x E))
prereq: (AND  ~(WANT y ~E)
              (HELPFUL y x))
              (CAN y E))
```

Clearly, we could have made the effect (WANT y E). COMPLY subsumes the remainder of the above plan, and progresses from shared beliefs to private ones (which cause y to achieve E). However, it is unclear which proposition should be chosen as the body. Should the body be a mutual belief (therefore involving a previous communication act) or need it only be a private belief? Finally, if E is a KNOWIF or KNOWREF proposition [1,2,10,19], then a more specific operator, ANSWER, can be defined.

Multiple summaries can occur because of some indirect uses of surface speech acts -- as in with an S-INFORM of x's WANT that leads to the same effect as an S-REQUEST [1,2,10,19]. Not only could the early part of the plan be summarized as an INFORM, and the later stages as a REQUEST, but a perhaps computationally useful operator would be one subsuming both the INFORM and REQUEST; call it a WANT-REQUEST. This formalizes the technique used in Woods et al's [32] system to "short-circuit" various chains of reasoning involving indirect speech acts.

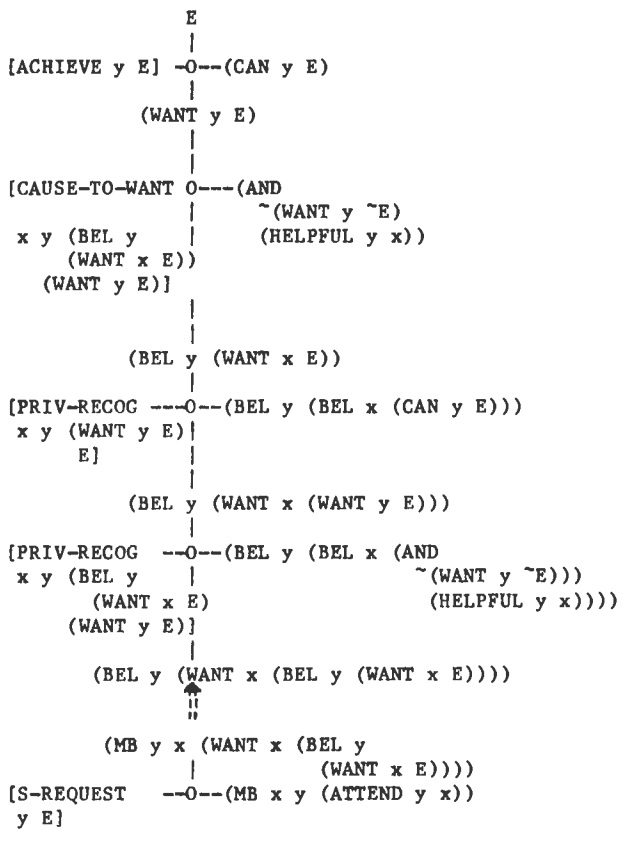
Substituting FORCE-TO-WANT for CAUSE-TO-WANT into the above plan allows us to create a summary termed COMMAND as follows:

[COMMAND x y E]

```
effect: (MB y x (WANT x E))
body:   (MB y x (WANT x (BEL y (WANT x E)]
prereq: (MB y x (BEL y
                 (HAS-AUTHORITY-OVER x y]
```

COMMAND differs from REQUEST in its being insensitive to the hearer's helpful (or non-helpful) disposition and to her prior desires.

Finally, we can create a plan in which the effect takes hold in a non-communicative manner:




Again, the implication marks the shift from mutual beliefs to private ones. By Schiffer's [26] definition, any effect obtained on the basis of private beliefs was not communicated. Thus, on philosophical grounds, one would not classify a summary of this plan as describing an illocutionary act.

## Possible Uses of Illocutionary Operators

The formalism indicates that certain illocutionary operators are redundant--they can be derived from other independently motivated operators. However, the redundancy is only relevant to achieving the illocutionary operator's effects. For the reasons stated below, the redundancy may be useful.

Illocutionary operators might be used to represent the meaning of illocutionary verbs. Consider verbs that report on social interaction. Corresponding operators can be defined to span multiple agents' achievements (e.g., COMPLY and ANSWER). Summary operators can perhaps be used for verbs requiring "uptake" [3]. Thus, a plan summarizable as a bet could contain portions summarizable as offerings and acceptances. The major questions for this approach would be when and to what end would those summaries expanded in the course of processing an utterance. Obviously the linguistic questions related to performatives are also relevant but as yet remain unanswered.

From a computational perspective, summary operators are useful in limiting a planner's search, as demonstrated by the use of MACROPs in STRIPS [12]. Summary operators allow for "short-circuiting" the interpretation of certain indirect speech acts ([18,2,4,31,32]). Further reduction in search could follow ABSTRIPS [22] in assigning priorities to the summary operator's prerequisites. Speech act plans could first be sought using high priority preconditions and later pruned by lower ones. Given suitable priority and threshold schemes, indirect achievement of a communicative goal may be as efficient as direct achievement.

Finally, the issues of dynamically acquiring summary operators, as in STRIPS, become relevant. Though a system may summarize a shared plan, there may be no corresponding illocutionary verb in its lexicon to describe that plan. This problem then presents an interesting challenge to a model of language use -- how could a system plan communicative acts to establish a jointly agreed upon vocabulary?

In summary, our model proposes a foundation for defining a class of illocutionary verbs. However, as the next section shows, there are formal and descriptive limitations to be overcome. Furthermore, other tests need to be applied to support the model.

## Limitations

Our scheme has only been applied to a narrow range of phenomena. First of all, we have only shown the redundancy for two illocutionary verbs ("requests" and "command") though a similar analysis has been done on "inform." Since these verbs are prototypical of Searle's [29] "directive" and "representative" classes, our hope is that this style of formalism can be extended to other members of those classes. Such an analysis is currently limited by our understanding of concepts such as benefit (for suggestions) and danger (for warnings).

We have not yet attempted to handle the class of indirect speech acts addressed by Perrault and Allen. Our efforts are currently hampered by the KNOWIF(P) --> P(or ~P) recognition inference stating that if you believe an agent wants to know whether or not P is true, then it is plausible to believe that agent wants P (or, wants ~P). The inference arises because a planner must determine whether or not an action's preconditions hold. In order to formalize the inference, an axiomatization of the behavior of a planner or a plan-recognizer is needed. Such a formalism would also have to capture stopping conditions for shared and private plan-recognition [1,2,13,25,32], and perhaps rating schemes for choosing the best plan [1,2,32].

Regarding the formalism, a major difficulty is the lack of an adequate axiomatization and semantics for BEL and WANT. For instance, the distinction between actively desiring, and "putting up with" (as the lesser of two evils)

needs to be drawn formally. BEL, given Hintikka's [15,16] treatment is the better understood concept.

A bothersome quirk of the formalism is that actions cannot appear as objects of want, and hence do not appear in the REQUEST summary operator. We are therefore searching for a propositional way to state that an action was done.

## Conclusions

The primary reason for pursuing this formalism is that is allows one to express naturally the communicative nature of illocutionary operators in terms of shared plans. It leads us to conclude that summarizing an utterance as the performance of an illocutionary act is not necessary to helpfully motivated plan recognition. The illocutionary operators that we have studied are redundant for achieving their effects, since the shared plans provide all the power, and their components are independently motivated. However, though we have suggested such operators are unnecessary, we cannot formally prove the point without further research, especially on the logic of WANT. The formalism has led to a foundation for "short-circuiting" certain implicatures, as recommended by Morgan[18], Perrault and Allen[19], and as attempted in Woods et al's [32] natural language system. Finally, it reveals the arbitrary nature of operator definition. Some choices can be decided using the adequacy test of third-party speech acts proposed by Cohen and Perrault [10]. Other decisions must await empirical evidence.

## References


1. Allen, J. A plan-based approach to speech act recognition (Doctoral dissertation, University of Toronto, 1979). Technical Report No. 131/79, Dept. of Computer Science, University of Toronto, January, 1979.
2. Allen, J.F., & Perrault, C.R. Analyzing intention in dialogue, forthcoming.
3. Austin, J.L. How to do things with words. J.O. Urmson (Ed.), Oxford University Press, 1962.
4. Brown, G.P. Indirect Speech Acts in Task-Oriented Dialogue: A Computational Approach, unpublished ms, MIT, 1979.
5. Bruce, B. Belief systems and language understanding (BBN Report No. 2973). January, 1975(a).
6. Bruce, B., Generation as a Social Action, Proceedings of the Conference on Theoretical Issues in Natural Language Processing, Cambridge, MA, 1975(b)
7. Bruce, B., & Schmidt, C.F. Episode understanding and belief guided parsing. Presented at the Association for Computational Linguistics Meeting at Amherst, Massachusetts (July 26-27, 1974).
8. Clark, H.H., & Marshall, C. Definite reference and mutual Knowledge. In A.K. Joshi, I.A. Sag, & B.L. Webber (Eds.), Proceedings of the Workshop on Computational Aspects of Linguistic Structure and Discourse Setting. New York: Cambridge University Press, in press.
9. Cohen, P.R. On knowing what to say: Planning speech acts (Doctoral dissertation, University of Toronto, 1978). Technical Report No. 118, Department of Computer Science, University of Toronto, January 1978.
10. Cohen, P.R. and Perrault, C.R., Elements of a plan based theory of speech acts, Cognitive Science, 1979, 3, 177-212.
11. Fikes, R., & Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 1971, 2, 189-208.
12. Fikes, R., Hart, P., & Nilsson, N.J. Learning and executing generalized robot plans, Artificial Intelligence, 1972, 3, 251-288.
13. Genesereth, M.R., Automated consultation for complex computer systems, (Doctoral dissertation), Dept. of Computer Science, Division of Applied Sciences, Harvard University, September, 1978.
14. Hintikka, J. Knowledge and belief. Ithaca: Cornell University Press, 1962.
15. Hintikka, J. Semantics for propositional attitudes. In J.W. Davis et al. (Eds.), Philosophical Logic. Dordrecht-Holland: D. Reidel Publishing Co., 1969.
16. Grice, H.P. Meaning. The Philosophical Review, 1957, 66, 377-388.
17. Moore, R.C. Reasoning about knowledge and action (Doctoral dissertation, Massachusetts Institute of Technology, 1979). Artificial Intelligence Laboratory, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February, 1979.
18. Morgan, J.L. Two types of convention in indirect speech acts. In P. Cole (ed.), Syntax and Semantics, Volume 9: Pragmatics, New York: Academic Press, 1978.
19. Perrault, C.R., & Allen, J.F. A plan-based analysis of indirect speech acts.
20. Perrault, C. R., Allen, J. F., & Cohen, P. R., Speech acts as a basis for understanding dialogue coherence, in Proceedings of the second conference on theoretical issues in natural language processing, Champaign-Urbana, Illinois, 1978.
21. Perrault, C.R., & Cohen, P.R. Inaccurate Reference, In A.K. Joshi, I.A. Sag, & B.L. Webber (Eds), Proceedings of the Workshop on Computational Aspects of Linguistic Structure and Discourse Setting. New York: Cambridge University Press, in press.
21. Sacerdoti, E.D. A structure for plans and behavior (Doctoral dissertation, 1975). Technical Note 109, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, August 1975.
22. Sacerdoti, E.D. Planning in a Hierarchy of Abstraction Spaces. Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford, Calif., 1973.

23. Schank, R., & Abelson, R. Scripts, plans, goals, and understanding. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1977.
24. Schmidt, C.F., Understanding human action, Proceedings of the conference on theoretical issues in natural language processing, Cambridge, MA, 1975.
25. Schmidt, C.F., Sridharan, N.S., & Goodson, J.L., The plan recognition problem: An intersection of artificial intelligence and psychology, Artificial Intelligence 10, 1979.
26. Schiffer, S. Meaning. Oxford: Oxford University Press, 1972.
27. Searle, J.R. Speech acts: An Essay in the philosophy of language. Cambridge: Cambridge University Press, 1969.
28. Searle, J.R. Indirect speech acts. In P. Cole & J.L. Morgan (Eds.), Syntax and semantics, (Vol. 3), Speech acts. New York: Academic Press, 1975.
29. Searle, J. R. A Taxonomy of Illocutionary Acts, in K. Gunderson (ed.), Language, Mind, and Knowledge, University of Minnesota Press, 1976.
30. Strawson, P.F. Intention and convention in speech acts. In The Philosophical Review, 1964, 5, 73.
31. Wilensky, R., Understanding goal-based stories, (Doctoral dissertation), Research report # 140, Dept. of Computer Science, Yale University, September, 1978.
32. Woods, W.A., Bobrow, R., Brachman, R., Cohen, P., Klovstad, J., Sidner, C., & Webber, B., Natural language understanding, Annual Report, Bolt Beranek and Newman, Inc., 1980

# UNDERSTANDING ARGUMENTS

Robin Cohen
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A7

Abstract

This paper outlines a preliminary design for a system to understand one-sided arguments. These are a particular kind of conversation, where the speaker has one main objective: to convince the hearer of a particular point of view. Arguments are thus characterized by having an overall point, defended by some logical chain of reasoning. We develop methods to analyze arguments, considering them as intentional behaviour. For this first design, we concentrate on developing methods to recognize the logical form of the argument, by examining the relations between sentences.

## 1. The problem area

We are studying a particular kind of conversation - the one-sided argument. This is a speech with a main objective of convincing the hearer of a particular point of view. Arguments differ from other texts in that: (i) there is an overall point (untrue of stories) (ii) the point is an opinion which is to be defended (untrue of news reports) (iii) the individual sentences serve to support the point (untrue of informing rather than convincing arguments) (iv) there is an overall logical form: a method of reasoning, holding the argument together (untrue of non task oriented conversations).

Our main objective is to analyze arguments, producing a representation which reveals (i) the point and overall opinion (ii) the chain of reasoning supporting the point. The restricted form of arguments is used to develop a classification for each sentence as either claim, evidence for some claim, or a statement of control (i.e. a sentence about the structure of the argument - e.g. "We now present our conclusion"). To classify sentences and record the relationships between them, frames are defined for each of the basic logical rules of inference. The main operation of our analysis is thus a matching onto frames, which hold our representation and facilitate further processing.

The underlying philosophy of this system is that arguments may be considered as intentional behaviour. One motivation for this pragmatic approach is that there are some clear distinctions between shared and private knowledge in arguments. Speaker (S) and hearer (H) share some knowledge: both know that the main purpose is to convince; both are aware of standard techniques to convince (e.g. using analogies, contrast, examples, etc.). On the other hand, both the statements of S and the connections between them may be unknown to H. So H's task is both to recognize the logical forms being used and to believe that they are appropriate.

Another important reason for considering intentions is to facilitate the understanding process. H's comprehension process often involves deciphering and interpreting unstated assumptions. H may be able to determine unstated opinions of S or overall argument structure by

examining, for example, the choice of words (e.g. "however", "only"). But H also knows that S must facilitate H's understanding, in order to succeed in convincing H of his main point. H can thus postulate rules of coherence to interpret S's intentions and aid in analyzing the argument.

There seem to be two main levels to H's processing: determining what S believes, and deciding whether or not he, himself, believes it. The second task involves judging the credibility of arguments, and will not be addressed in this paper (See Section 3: Future Work).

This problem area, as defined, is different from other language understanding projects.

DeJong's FRUMP <DeJong 79> analyzes newspaper stories. This kind of text is similar to arguments in that (i) there may be statements of evidence and sources quoted (ii) it is important to believe the story. However, FRUMP does not concern itself with the underlying opinion on the overall topic, or with credibility. In contrast to FRUMP, we must distinguish the evidence in the argument and determine how the evidence supports the main opinion. Further, there is a basic representational difference between arguments and stories. DeJong himself addresses the issue in <DeJong 79>, indicating that his program can't handle editorials because these present arguments in a novel form, and scripts can't be written ahead to include these new ramifications.

Carbonell's POLITICS <Carbonell 78> analyzes opinionated text. But his system is given the underlying opinion (in the form of an ideology, represented as a set of goals). In our case, H assumes that the argument will conform to one ideology, but he must determine that ideology by examining the form of the argument. Furthermore, the main purpose of our analysis is distinct from Carbonell's: we are concerned with the overall form - why sentences are put together in a particular order. Carbonell concentrates more on analyzing individual events.

Allen <Allen 79> analyzes conversation as intentional behaviour. But again the goal of his system is distinct from ours. He is interested in recognizing speech acts; we know that the main purpose is to convince, but must determine how the form of the argument succeeds in convincing. Some of Allen's methods of plan deduction to uncover intentions may be useful to us.

In sum, our problem area presents us with a new language understanding task. We are concerned with determining form and uncovering intentions to perform analysis.

## 2. The Analysis Process

### 2.1 Overview

This section describes the basic procedure the hearer (H) follows to determine the logical form of an argument, leaving aside the issue of credibility.

The basic step in the analysis process is for H to take a sentence of the argument and to determine whether it is a new claim or evidence for some previously stated claim. In this way, H can uncover the intended function of each sentence. The basic unit of analysis is actually a proposition - the propositional content extracted from a sentence. (A simplifying assumption for our system right now is that the propositional content is made available).

To help H in classifying a proposition, there is a standard set of frames, representing rules of inference. In addition to frames representing correct rules like modus ponens and modus tollens, there are some representing bad logic, which is often used in arguments (either intentionally or in an attempt to justify bad evidence). Consider the following set of frames:

SET OF FRAMES:

(Abbreviations: MAJOR - major premise, MINOR - minor premise, CONC - conclusion)

| (correct) | MAJOR | MINOR | CONC |
|---|---|---|---|
| MODUS PONENS | A-->B | A | B |
| MODUS TOLLENS | A-->B | ~B | ~A |
| MODUS TOLLENDO PONENS | Aor~B | B | A |
| MODUS PONENDO TOLLENS | Aor B | B | ~A |
| (incorrect) | | | |
| ASSERTING CONSEQUENT | A-->B | B | A |
| DENYING ANTECEDENT | A-->B | ~A | ~B |

This selection of frames is motivated by <Sadock 77>, which indicates those correct and incorrect logical rules that occur most often in conversation. Our analysis of examples so far seems to function well with this restricted set.

For each of these frames representing rules of inference, it is often the case that they are not completely spelled out in the argument. Any one of the major premise, minor premise or conclusion may be omitted, and H must still be able to recognize the logical form intended, by filling in the missing detail. (This kind of argument is referred to as "modus brevis" in <Sadock 77>). H is aware of these variations in frames.

CLASSIFICATION OF FRAMES(e.g:Modus Ponens)

| | | |
|---|---|---|
| normal | A-->B,A | /B |
| normal MAJOR | A-->B | /B |
| normal MINOR | A | /B |
| MAJOR | A-->B | (assume rest) |
| MINOR | A | (assume rest) |
| CONC (hard) | B | (assume rest) |

How can H make use of these frames to represent the logical form of an argument? Consider MAJOR premise, MINOR premise, and CONCLUSION to be slots of a frame, with the constraint that the premise slots must lead to the conclusion. H is motivated in filling frames in order to classify propositions: we say that A is <u>evidence</u> for B iff they both fill slots in a frame such that A is a premise for B. H tries to instantiate a frame by filling its slots with propositions of the argument, possibly inferring premises that are not "spelled out", and thus choosing one of the "missing" versions of frames. The result is an indication of the logical relations between propositions in an argument.

## 2.2 <u>Details</u>

The overview illustrates the basic frame matching technique used to classify propositions. This section examines the analysis process in detail. In particular, a proposition may be classified in many different ways. We develop a scheme which <u>formulates hypotheses for each proposition</u> as to how it can fit with the rest of the argument, and then <u>rates</u> these hypotheses to determine the most likely interpretation. The rating scheme is based partly on fitting into our logic frames, and partly on other heuristics - e.g. based on the actual choice of words. In addition, this section describes the processing of the entire argument in more detail: how the classification of one proposition affects

another, how to isolate sub-arguments, and what kind of representation to build for the overall argument.

Rating Hypotheses

Consider the following classification scheme for a single proposition:

HYPOTHESES FOR CLASSIFICATION OF PROP(i)

(i) new claim
(ii) evidence for some future claim
(iii) evidence for PROP(i-1)
evidence for PROP(i-2)
...
evidence for PROP(1)

To illustrate that more than one hypothesis is probable for a given proposition, and that rating is thus necessary, consider the following example:

EX1: 1)There is too much crime in the city
2)We need more police

This example gives insight into the possible functions of a proposition, and the need to rate hypotheses. Consider 1) in isolation: it can be either a claim (and we'd expect evidence about the amount of crime) or evidence for some claim. Upon seeing 2), a connection is found between 1) and 2) (e.g. "more police --> less crime"), so 1) is interpreted as functioning as evidence for 2).

Determining whether a proposition is evidence for another is done by trying to fill slots in a frame, as described in Section 2.1. Since there are many frames in our system, each of the hypotheses in (iii) really represents a variety of options - e.g. evidence for PROP(i-1) by modus ponens, evidence for PROP(i-1) by modus tollens, etc. Since propositions are processed one at a time from the start, the only options that can be directly measured are those using propositions that have already been processed - hence the distinction between (ii) and (iii) above.

TRYING FRAMES

The first step is to determine the ratings for hypotheses in (iii) by actually trying to fit frames. To ensure that correct logic frames are given preference over bad logic frames, consider a frame system where the bad logic frames are connected to their correct logic counterparts using SIMILARITY links (as described by <Minsky 75>). In the spirit of <Tsotsos 80>, similarity links trigger alternatives when an exception is raised in trying to fill a slot in a frame. For example:

EX2: 1)Whenever the stock market crashes Carter refuses to appear on TV
2)Carter has refused to appear on TV
3)So the stock market must have crashed

With 1) and 2), modus ponens fails - we have A-->B, then B. So we try "asserting consequent", and with A asserted in 3), find that the bad logic frame succeeds. So bad logic frames are only tried when correct logic ones fail.

Each hypothesis in (iii) thus gets expanded into a list of options: one for each of the correct logic frames. Then, each option is tried. If frame constraints can't be satisfied, the option is given a very low rating. As H tries to instantiate frames, he must be aware that he is often interpreting beliefs of the speaker. So, for instance, modus ponens is usually recognized as: (S believes (A-->B)), (S believes A) thus (S believes B). (And not as "(A-->B) is believed to be true by H"...etc.). This introduces an interesting sub-topic of how H distinguishes beliefs, wants, and goals of S to aid analysis (see Section 3: Future Work).

Even when H succeeds in instantiating a frame, the rating for that frame may be lowered if it was "difficult" to fill in

the necessary "chains of reasoning". Consider the following:

EX3a: 1) There is a national railroad system in the US today
2) Railroads serve more than local needs

EX3b: 1) Railroads deliver goods across state lines
2) Railroads operate on a national scale

To determine 2) as evidence for 1) in 3a requires a rather long chain of reasoning (something like "exists national system --> operates on national scale --> carries goods between localities --> serves more than local needs"). In 3b, the chain is brief ("across state lines --> national"). H may wish to lower the ratings for longer, less certain chains.

In addition to actually measuring the options in (iii), we also consider some heuristics to affect ratings, which include an assessment for (i) and (ii).

LINGUISTIC CLUES

Sometimes H is aided in the classification of propositions by the actual choice of words. For example, H can recognize (and S knows that he will recognize) the organizational function of certain words and phrases, and can thus detect structure. For example, a classification like <Hobbs 78> is reasonable:

| CATEGORY | EXAMPLE |
|---|---|
| SUMMARY | in conclusion |
| PARALLEL | in the first place |
| EXAMPLE | for example |
| DETAIL | in particular |
| CONTRAST | on the other hand |
| CAUSAL | therefore |

The different functions can be interpreted in terms of claims and evidence for H to expect.

In addition, the choice of connective between propositions should provide H with an insight into S's intentions. Certainly, compound sentences suggest a common topic for their clauses - but the presuppositions attached to the words can indicate the function of each clause in the overall form. We are working on a precise description of connectives, and of particular constructions like analogy, with a view to developing a description for overall logical form that further specifies the evidence / claim distinction. It is also interesting to examine the motives of S in choosing a particular construction - an issue which relates more to "credibility".

SYSTEM RULES

Furthermore, there are heuristics called SYSTEM RULES to indicate preferred classifications. These are rules motivated by the intentional nature of arguments. H expects S to conform to certain coherence rules, because he knows S must be clear in order to convince H of his point of view. H thus has some defaults about how propositions relate. Consider the following:

1) a proposition which is a statistic is likely to be evidence

2) a proposition which quotes a particular authority is likely to be evidence

(based on the idea that claims are considered to be disputable, while statistics and quoting authorities are less disputable material)

3)a change in topic often signals a new claim

4) repetition of topic suggests some connection (propositions may support one or another, or both support a common third)

(judging continuity of topic)

5) rules of distance: prefer connections between propositions located closer

together in the argument

* when a proposition rates high as a new claim, strenghten the ratings of previous propositions which let them relate to each other (since it should be hard to find evidence located after this new train of thought)

something like the rule of distance applies for frame fitting:

* if the "chain of inference" needed to instantiate a frame is long, decrease the rating for that hypothesis.

(other sub-rules based on distance may develop)

EX4: 1) Peter is a good musician
2) He has produced 50 songs this year

2) can be either:
*new claim
*evidence for future claim
*evidence for previous claim

Not only does 2) --> 1) fit into a frame (with chain "prolific --> good") but because of the statistic in 2), all evidence options are strengthened.

Updating ratings

So far we have described how H can do a thorough analysis of a single proposition. We now consider how the ratings for one proposition can affect others. Recall that our control structure is such that propositions are processed one at a time. Now, once a proposition is processed, the ratings of previous propositions are re-evaluated. Consider the following interactions:

(i) when PROP(i) is processed, go back and evaluate the "evidence for future claim" option for previous propositions, using PROP(i)

EX5: 1)Motorcycle gangs caused 50% of the deaths in our small town
2)These gangs are dangerous

None of the hypotheses for 1) can be directly measured since it is the first sentence. When 2) is processd, the hypothesis "2) as evidence for 1)" is measured and rates low (hard to establish). Then 1) is re-processed, and "1) as evidence for 2)" is measured. This works with modus ponens and missing premise "caused lots of deaths --> dangerous".

(ii) when PROP(i) rates high as evidence for PROP(j), increase the claim rating for PROP(j)

(iii) if a new proposition is created - i.e. filled in as missing detail in a frame - then this proposition is added to our system. Then, if a future proposition rates high as being related to one of these derived propositions, we increase the rating for the hypothesis which "created" it

Overall form

We now begin to have a feel for how rating propositions can propagate through an argument. We must as well try to develop a representation for the overall argument. What our classification of propositions has done so far is to indicate logical connections between sets of propositions. This, in fact, isolates sub-arguments, each with its own claim and set of supporting evidence. To complete the analysis, H must first of all determine the boundaries, where one sub-argument ends and another begins. Consider a methodology in the spirit of <Tsotsos 80>, strengthening hypotheses that indicate good "continuation" into a separate unit. Since a proposition can participate in more than one frame, the most likely option must be chosen. Some ideas on how to measure boundaries include using (i) change of topic (ii) ratings for "new claim" option (iii) combined ratings

for hypotheses indicating a unit: claim plus some evidence.

EX6: 1)Rogers is a talented songwriter
2)He has won 6 Junos
3)His son Peter has been active on Broadway for the past 15 years
4)Peter has won 4 Tony awards
5)Both Rogers and son are very talented

Here 2) rates high as evidence for 1), and 1) as a claim. Then 3) rates high as a new claim (new topic), so we strengthen ratings for 2) and 1) to consider them a separate unit.

Once sub-arguments have been isolated, using the hypotheses for propositions with the highest ratings, the overall argument structure can be analyzed. For now, this process is done at the end of processing. We try to relate all the individual claims from the sub-arguments into some overall form. Our final output is thus an indication of the most likely structure for the argument, in the form of relations between sub-arguments and forms of sub-arguments, represented as instantiated rules of inference.

Continuing with EX6:
4) and 3) share a common topic. 4) as evidence for 3) is possible, but with a weak inference. 3) and 4) both as evidence for a common claim rates high. Then, with 5) we reach the end and have to wrap up. 5) splits into 5a)Rogers is talented and 5b)Son is talented. We find 3) and 4) are evidence for 5b), and 1) is evidence for 5a). The most likely overall form is thus: two sub-args, one with 5a) as claim, with 1) as evidence (and 2) as evidence for it) and one with 5b) as claim and 3) and 4) as evidence.

Our next step will be to develop a more sophisticated control structure. Ideally, we will be analyzing overall form in parallel with our detection of sub-arguments.

## 3. Future Work

Section 2 presented a preliminary design for the analysis of arguments. In fact, most of the ideas for the design developed from an examination of many examples of arguments ( including a good selection from <Holmes and Gallagher 17>). The set of rules developed so far seem to function well, but are certainly not presented as an optimal solution.

We have already alluded to several areas that need more development, including: (1) more precise formulation of linguistic clues (2) more precise measure of frame fitting (3) more sophisticated overall control structure and (4) thorough description of the rating mechanism, including some actual figures to show comparative worth of different rules.

The major area for future work, however, is to develop mechanisms for H to determine and interpret the intentions of S. Issues of concern include: (i) recognizing and making use of control sentences - sentences about the structure of the argument (We have skipped discussion of this kind of sentence in this paper) (ii) distinguishing the wants, beliefs, and goals of S to aid in analysis (as alluded to before, H often recognizes relations between propositions not as logical truths but as beliefs of S. In addition, H may recognize wants, in particular with claims suggesting a "course of events". For example, when S says "There should be less war", H must recognize this as "S wants (less war)", and when S then says "Less war would mean more prosperity", H must recognize this as a belief of S, and through some logical reasoning conclude "S wants (more prosperity)".) (iii) determining the

motive behind S's choice of argument form and content - examining issues like order of presentation, choice of evidence, and deliberate deception. Furthermore, this investigation of intentionality should lead to some comments on credibility - the factors that H has available to influence his belief in the argument.

## 4. Conclusion

This paper presents some ideas for the design of a system to understand arguments. We give insights into how analysis can proceed: the rules available and the form of representation we want to build. We have only begun to look at the intentional aspect of arguments: how H can guide his analysis by his expectations about S. But we begin to see how this particular natural language task has restrictive characteristics which enable us to formulate specific methods of analysis: not only is there an overall form, but the speaker is forced to limit his choice of overall form so that the hearer can recognize all the points, and be convinced.

## Acknowledgements

I am grateful to Ray Perrault for his supervision of this research, and to Alex Borgida for his comments on this paper.

## Bibliography

<Allen 79> Allen, J.; "A Plan Based Approach to Speech Act Recognition"; U. of Toronto Dept. Comp. Sc. Tech. Rept. No. 131

<Carbonell 78> Carbonell, J.; "POLITICS: Automated Ideological Reasoning"; Cognitive Science 2,1

<DeJong 79> DeJong, G.; "Prediction and Substantiation: Two Processes that Comprise Understanding"; IJCAI 79

<Hobbs 78> Hobbs, J.; "Why is Discourse Coherent?"; SRI International Tech. Note No. 176

<Holmes and Gallagher 17> Holmes, H.W. and Gallagher, O.; Composition and Rhetoric

<Minsky 75> Minsky, M.; "A Framework for Representing Knowledge"; in The Psychology of Computer Vision, P. Winston, ed.

<Sadock 77> Sadock, J.; "Modus Brevis: The Truncated Argument"; in Papers from the 13th Regional Meeting, Chicago Linguistic Society

<Tsotsos 80> Tsotsos, J.; "A Framework for Visual Motion Understanding"; PhD Thesis, Dept. Comp. Sc., U. of Toronto

# Example Generation

Edwina L. Rissland

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

This paper addresses the problem of generating examples that meet specified properties which are used to direct and constrain the generation process, which we call CONSTRAINED EXAMPLE GENERATION. We begin by presenting a few examples of CEG taken from protocols. Based upon such examples, we present a model of the CEG process. We describe the architecture of a system that generates examples from specifications and present examples of problems that it has solved.

## 1. INTRODUCTION

The ability to generate examples that have specified properties is important in many intellectual areas, such as mathematics, linguistics and computer science [Collins 1979]. It is important from the standpoints of learning and teaching as well as performing research. For instance, examples are needed for inductive reasoning, sharpening of conjectures, and concept formation and refinement [Polya 1968, 1973; Lakatos 1963; Winston 1975; Lenat 1976; Soloway 1978]. Having a rich stock of examples is intimately related to understanding [Rissland 1978a, b]. Thus, examples lie at the heart of efforts to learn and reason in a subject.

When an example is called for, one can search through one's storehouse of known examples for an example that matches the properties of the desired example. If a satisfactory match is found, then the problem has been solved through retrieval.

However, when a match is not found, how does one proceed? In many cases, one modifies an existing example that is judged to be close to the desired example, or to have the potential for being modified to meet the constraints.

In some cases, generation through modification fails. Experienced researchers, teachers and learners do not give up however. Rather they switch to another mode of example generation which involves building up an example from very elementary consituents through careful attention to the desiderata and "unpacking" of the concepts involved. This phase of CEG is usually more difficult than either retrieval or modification.

This paper presents a model of CEG that incorporates three phases: RETRIEVAL, MODIFICATION, and CONSTRUCTION. This model is based upon analyses of protocols of example generation tasks taken from mathematics and computer science [Rissland 1979, Woolf and Soloway 1980].

## 2. PROTOCOLS OF CEG

In this section, we describe some protocols for CEG tasks taken from the domain of elementary function theory in mathematics (which deals with concepts such as continuity) and from elementary LISP programming (especially with regard to concepts concerning list structure).

### 2.1 Examples of Retrieval

The type of questions that most people answered through retrieval is:

Give an example of a function that is continuous
but not differentiable (at a point).

Give an example of a list
with three elements.

Most people handled these problems by offering their favorite standard "reference" examples [Rissland 1978a, b]: for the first problem, the absolute value function (at the origin) and for the second, a list like "(A B C)". Responses were usually immediate indicating that the retrieval was very readily made.

## 2.2 Examples of Modification

An example of a problem solved through modification of a known example is:

Give an example of a list
with three elements
where the depth of the first atom is 3.

Subjects frequently modified an example, such as "(A B C)" by adding two more parentheses around the first element, to produce the list

( ((A)) B C)

Other subjects truncated a longer list such as the list of digits or added to a shorter list such as (0 1), as well as adding parentheses. The example chosen for modification depends on the context of the problem (e.g., the sequence of recently solved problems) and the subject's data base of examples and its epistemology (e.g., his favorite references).

An example of a mathematics problem which every subject solved by modification is the following:

Give an example of a non-negative,
continuous function
defined on the entire real line
with the value 1000 at 1, and
with area under its curve less than 1/1000.

Most protocols for this question began with the subject selecting a function (usually, a familiar reference example function) and then modifying it to bring in into agreement with the specifications of the problem.

There were several clusters of responses according to the initial function selected and the stream of the modifications pursued. A typical protocol went as follows:

"Start with the function for a "normal distribution". Move it to the right so that it is centered over x=1. Now make it "skinny" by squeezing in the sides and stretching the top so that it hits the point (1, 1000)."

"I can make the area as small as I please by squeezing in the sides and feathering off the sides. But to demonstrate that the area is indeed less than 1/1000, I'll have to do an integration, which is going to be a bother."

"Hmmm. My candidate function is smoother than it need be: the problem asked only for continuity and not differentiability. So let me relax my example to be a "hat" function because I know how to find the areas of triangles. That is, make my function be a function with apex at (1, 1000) and with steeply sloping sides down to the x-axis a little bit on either side of of x=1, and 0 outside to the right and left. (This is OK, because you only asked for non-negative.) Again by squeezing, I can make the area under the function (i.e., the triangle's area) be as small as I please, and I'm done."

### Comments

Notice the important use of such operations as "squeezing", "stretching" and "feathering", which are usually not included in the mathematical kit-bag since they lack formality, and descriptors such as "hat" and "apex". All subjects made heavy use of curve sketches and diagrams, and some used their hands to "kinesthetically" describe their functions. Thus the representations and techniques used are very rich.

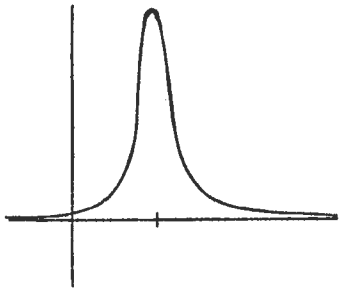


FIG 1A

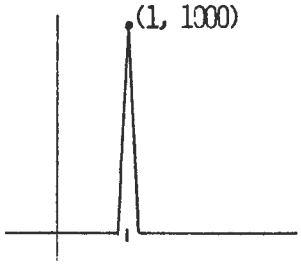



FIG 1B

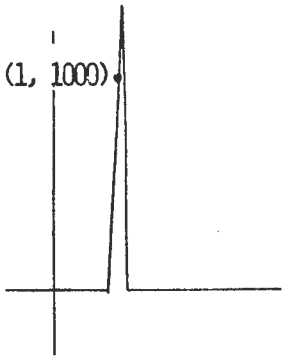



FIG 1C

Another thing observed in all the protocols (of which there were about two dozen for this problem) is that subjects make implicit assumptions about the symmetry of the function (i.e., about the line x=1) and its maximum (i.e., occuring at x=1 and being equal to 1000). There are no specifications about either of these properties in the problem statement; however, they are mathematically simplifying and cognitively natural.

These are the sort of tacit assumptions that Lakatos talks about [Lakatos 1963]; teasing them out is important to study both mathematics and cognition.

Example functions for protocols are shown in Figures 1a and 1b; another mathematically permissible example is shown in 1c.

## 2.3 An example of Construction

In this subsection, we present a protocol of example generation in which the example is built largely "from scratch" by working with the concepts involved in the specifications of the desiderata, instantiating them, and combining exemplars to produce a new example. The problem is:

> Give an example of a list of lists
> each of which has two elements
> the first of which is a literal atom.

A typical protocol began with the subject sketching out the overall structure of the desired list as:

```
( (A1 L1)
  (A2 L2)
  (A3 L3)
  ...
        )
```

where in each sublist, Ai stands for a literal atom, and Xi the second element.

The subject next focussed his attention on instantiating the Xi's. Since he wanted to emphasize the fact that the elements of the sublists could themselves be lists -- "there's a lot of embeddedness possible here" -- he made each of the Xi's a list of atoms (a "LAT").

The subject began to write each Xi as (Ai1 Ai2 ... Ain) and then remarked that this level of generality was more than the problem called for. In particular, nothing was said about keeping the Xi's different: "So, why not make them all the same, like (00 01)".

The candidate example now looks like:

```
( (A1 (00 01))
  (A2 (00 01))
  (A3 ...

                )
```

The subject next decided to pin down the length of the "big" list by making it be "not too short, like 2, and not too long either; why not 7". He tended to the Ai's by noting that A1, A2, A3, ... A7 are perfectly fine literal atoms.

The list thus offered is:

```
( (A1 (00 01))
  (A2 (00 01))
  (A3 ...
  ...
  (A7 (00 01)) )
```

Even though the subject was satisfied with this answer, he noted that it really didn't have to be so complex or long; the following list would do:

```
( (A1 1) (A2 2) (A3 3) )
```

He said he made his list have a length longer than 2 because he didn't want it to be confused with the length of the sublists (i.e., 2). However, he said that a list of length two would be acceptable, but a list of length one would not since "after all the problem called for a list of lists".

"The list:

```
((A B) (A B))
```

would also do just fine. In fact, the possibilities are endless."

### Comments

There are several observations to be made on this protocol. First, the subject had a general model of a list and procedures to instantiate it (e.g., generate literal atoms and lists) and he had procedures to modify lists and properties of lists. Second, the subject made several implicit assumptions on the example to be generated, such as (1) its length, (2) the non-repeatedness of some elements, (3) its complexity (e.g., depth), and (4) uniformity (e.g., of list-structure).

## 3. A CEG MODEL

From analyses of protocols such as presented in Section 2, we developed the following general model of the CEG process. Presented with a task of generating an example that meets specified constraints, one:

(1) SEARCHES for and (possibly) RETRIEVES examples satisfying the constraints. This is done by searching through the knowledge base and judging examples for their match (or partial match) to the desiderata;

(2) MODIFIES an existing example judged to be close or having the potential for fulfilling the desiderata;

(3) CONSTRUCTS an example from elementary knowledge, such as definitions, principles and more elementary examples from the knowledge base.

Thus, there is a spectrum of responses to a CEG task ranging from having a ready answer as in (1) to having no especially close fitting candidate as in (3). In general, Task N depends on and follows Task N-1.

This information processing model of CEG is useful not only in describing human protocols, but also precisely specifying a computational model.

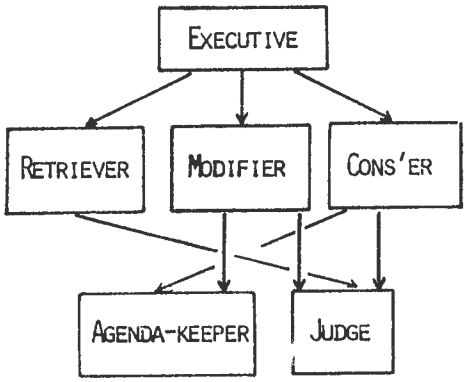



FIG 2

## 4. ARCHITECTURE OF A CEG SYSTEM

From the model of the last section, we have developed a system that solves CEG problems in the LISP domain. It has also been used to hand-simulate CEG problems in the mathematics of linear and piece-wise linear functions.

We have implemented this CEG model in the LISP domain. Written in LISP, it currently runs interpretively on a VAX 11/780 running under VMS. Examples of problems and solutions are given in Section 6.

The knowledge in our CEG system resides in two major sources: the knowledge base upon which the system runs, and the knowledge embedded in the processes operating on that base. The knowledge consists of general epistemological knowledge (e.g., the structure and types of examples) and domain-specific knowledge (e.g., particular example modification techniques).

The system consists of several components -- roughly one for each of the three phases of the model -- which handle different aspects of CEG. The flow of control between the components is directed by an EXECUTIVE procedure. Figure 2 shows the general architecture of our system.

The components use a common knowledge base which consists of two parts: (1) a "permanent" knowledge base of "Representation-spaces" [Rissland 1978]; and (2) "temporary" knowledge generated during the solution of a CEG problem.

There are four representation spaces, each of which is a set of items, represented as frame-like data structures, and organized according to predecessor-successor relationships. Examples-space, which is by far the most heavily used in our current system, consists of known examples organized according to the relation of constructional derivation reflecting which examples are constructed from which others. The other spaces and their relations are: Concepts-space: definitional dependency; Results-space: logical dependency; and Procedures-space: procedural dependency.

Before the system is given any CEG problems to work on, we create an initial set of representation spaces. The initial state of the Examples-space for the set of problems described in this paper is shown in Figure 3. The spaces are modified -- mostly through the

addition of examples to Examples-space -- as the system works through CEG problems.

The temporary knowledge held by the system during a CEG problem run includes a list of the constraints of the problem, an agenda of candidate examples, and various bookkeeping parameters such as "boxscores", "constraint satisfaction counts" and "recency counts".

## 5. CEG SYSTEM COMPONENTS

(1) The EXECUTIVE is responsible for initializing the system for a CEG problem, directing the flow of control among the components, and cleaning up afterwards. It accepts a CEG problem in prescribed format from the user and sets up the problem specifications in the temporary knowledge base.

The problem desiderata are kept on the CONSTRAINT-LIST, which has as many entries as there are constraints. Each constraint is recorded as a pair of properties DESIRED-PROPERTY and DESIRED-VALUE. For instance, the specification of the three constraint problem of "a list, of length 3, where the depth of the first atom is 1" is recorded by the following properties (PLIST's) for the constraints:

```
CONSTRAINT-1    DESIRED-PROP: (TYPE X)
                DESIRED-VALUE: LIST

CONSTRAINT-2    DESIRED-PROP: (LENGTH X)
                DESIRED-VALUE: 3

CONSTRAINT-3    DESIRED-PROP: (DEPTH
                    (FIRST-ATOM X) X)
                DESIRED-VALUE: 1

                Problem 1
```

The EXECUTIVE dictates the behavior of the system as a whole by specifying the orderings used by the other processes, such as the order of retrieval of candidate examples used by the RETRIEVER and the order of application for modification techniques used by the MODIFIER.

(2) The RETRIEVER searches the knowledge base for examples on request from the EXECUTIVE. It searches through Examples-space by examining examples in an order specified in terms of attributes such as "epistemological class" [Rissland 1978], position in the Examples-graph, and recency of creation.

In the problems described in Section 6, the "retrieval order" used was:

```
reference examples before
counter-examples before
start-up examples before
examples without epistemological
          class attribute
```

and in the case of ties

```
predecessors before
successors.
```

This retrieval order biases the system to examine ubiquitous and earlier-contructed examples before others. The order of CANDIDATES retrieved from the initial Examples-space of Figure 3 is thus:

(A B C)

(0 1 2 3 4 5 6 7 8 9)

(0 1)

( )

(A)

(A B C D E)

With each new example selected, the RETRIEVER calls the JUDGE to evaluate the example to ascertain how well it satisfies the desiderata.

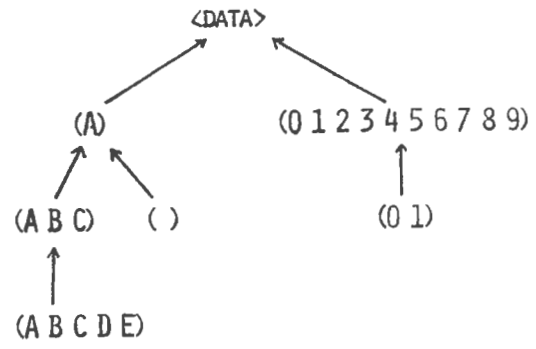



FIG 3

(3) The JUDGE evaluates a CANDIDATE example by cycling through all of the DESIRED-PROPERTY/DESIRED-VALUE pairs on the CONSTRAINT-LIST, comparing them with the actual properties of the CANDIDATE, and recording the results of the comparison. Thus, the JUDGE's basic cycle is evaluation, comparison and record.

The JUDGE records the results of the comparison by FILLING-IN the BOX-SCORE and the CONSTRAINT-SATISFACTION-COUNT ("CSC") slots in the representation frame of the CANDIDATE. The CSC is simply the number of desiderata met by the CANDIDATE.

The BOX-SCORE is a list of 2-tuples, one for each constraint, of the form (ACTUAL-VALUE, T or NIL). The ACTUAL-VALUE is the CANDIDATE's value for the DESIRED-PROPERTY; T is entered if the ACTUAL-VALUE equals the DESIRED-VALUE, and NIL if not.

The BOX-SCORE for the example "(A B C)" in Problem 1 would be:

( (LAT T) (3 T) (1 T) )

The CSC for this example would be 3, that is, all the constraints are met; the success of this example would be recorded as a T in its "SF" (SUCCESS/FAILURE) slot. With the above retrieval order on the Examples-space of Figure 3, Problem 1 would be solved with the first example retrieved.

If the example "(A)" were judged, its BOX-SCORE would be:

( (LAT T) (1 NIL) (1 T) )

The CSC for this example would be 2.

(4) The MODIFIER is invoked by the EXECUTIVE when the RETRIEVER has been unable to find an example meeting the constraints from its search through Examples-space.

The MODIFIER calls the AGENDA-KEEPER to set up an agenda of examples to be modified. The MODIFIER then works down the AGENDA trying to modify each entry in turn until success is achieved or the agenda exhausted.

To modify an example, the MODIFIER examines its BOX-SCORE for the constraints that were unsatisfied. It calculates the DIFFERENCE between the DESIRED-VALUE and the ACTUAL-VALUE for each DESIRED-PROPERTY not satisfied. Using the DESIRED-PROPERTY and the DIFFERENCE as an index in a difference-reducing table, the MODIFIER's DIFFERENCE-REDUCER finds and then applies modification techniques to the example.

For instance, for the example "(A)" with a CSC of 2 for Problem 1, the property not met is that of having a length equal to 3. The DIFFERENCE between the DESIRED- and ACTUAL-VALUE is +2. The difference-reducing technique MAKE-LONGER is found by looking for modification techniques affecting the LENGTH attribute of a list and reducing the DIFFERENCE, i.e., by making it longer by 2. (If the difference were -2, as would be the case for the example "(A B C D E)", the appropriate technique would be MAKE-SHORTER).

When there is more than one unsatisfied constraint, the MODIFIER orders its modification attempts according to the order specified by the EXECUTIVE. For the sample problems of this paper, the modification order is to apply techniques that affect:

TYPE before
LENGTH before
DEPTH before
GROUPING

The modified example is then re-judged and a new BOX-SCORE and CSC calculated. If the CSC is improved, the MODIFIER prints a message to the user of "success" or "failed" and asks whether it should continue modifying this example by going through another difference analysis, difference reduction, judgement cycle. If the CSC goes down, the MODIFIER abandons its attempt to bring the example into line, goes on to the next example on the AGENDA, and does not re-queue the example. Thus the MODIFIER engages in a form of hill-climbing.

The modified example must be re-judged for two reasons: (1) some techniques are heuristic and do not guarantee successful modification; and (2) there can be interaction between the constraints, that is, a successful modification for one constraint may undo satisfaction of another.

For instance, the system can make a NESTED-LIST from the LAT "(A B C)" by GROUPing "A" and "B", i.e., "( (A B) C)". However, before the modification technique was applied the LENGTH was 3, but now, after modification, it is 2. Satisfaction of the NESTED-LIST constraint has undone the LENGTH 3 constraint.

In the next version of our system, we shall re-judge an example after each modification, and also protect some contraints.

(5) The AGENDA-KEEPER is called by the MODIFIER and CONS'ER to set up the AGENDA of examples to be modified or instantiated.

When called by the MODIFIER, the AGENDA-KEEPER compiles an agenda of items to be modified based upon the CSC's calculated and recorded during the retrieval phase: the examples are ranked in order of their CSC's. Thus, the CSC is used as a measurement of the closeness of the example to meeting the constraints. In the case of a tie, the retrieval ordering is used.

(6) The CONS'ER is called by the EXECUTIVE when the MODIFIER is unsuccessful in its attempts to produce a solution or a model needs to be instantiated. The CONS'ER uses the procedural formulations of concepts stored in Concepts-space.

## 6. SAMPLE PROBLEMS

[NOTE: Text in this section is actual computer output generated by our CEG system; however explanatory text has been added (indicated by a "$") and some output modified to improve readability.]

### Problem 2

$Problem 2 asks for a list of length 3 whose first atom has a depth of 3:

```
(x1 (desired-value list desired-prop
 (typep candidate)))
(x2 (desired-value 3 desired-prop (length
 candidate)))
(x3 (desired-value 3 desired-prop (depth
 (first-atom candidate) candidate)))
```

$The retrieval phase is entered with the Examples-space of Figure 2. The retrieval order of candidates is:

<abc>
<**digits>
<**bits>
<empty>
<a>
<abcde>

$The RETRIEVER reports on each candidate tried, by printing out its BOXSCORE, CSC and SF:

```
   candidate name = <abc>
 candidate-value = (a b c)
     csc = 2   sf = nil
(entry-x1 (lat t))
(entry-x2 (3 t))
(entry-x3 (1 nil))
"failed"

   candidate name = <**digits>
 candidate-value = (0 1 2 3 4 5 6 7 8 9)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (10 nil))
(entry-x3 (1 nil))
"failed"

   candidate name = <**bits>
 candidate-value = (0 1)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (2 nil))
(entry-x3 (1 nil))
"failed"

   candidate name = <empty>
 candidate-value = nil
     csc = 0   sf = nil
(entry-x1 (atom nil))
(entry-x2 (0 nil))
(entry-x3 (0 nil))
"failed"

   candidate name = <a>
 candidate-value = (a)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (1 nil))
(entry-x3 (1 nil))
"failed"

   candidate name = <abcde>
 candidate-value = (a b c d e)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (5 nil))
(entry-x3 (1 nil))
"failed"
```

$The problem desiderata are not met by any example in the data base, and thus the modification phase is entered.

$The AGENDA of candidates for modification is (the CSC is given after the candidate's name):

(<abc> 2)
(<**bits> 1)
(<a> 1)
(<**digits> 1)
(<abcde> 1)
(<empty> 0)

```
$The MODIFIER goes to work on the first
candidate, (A B C):

-----------------------------------------
constraint = ((typep candidate) list)
actual score = (entry-x1 (lat t))

modify-candidate   ok

-----------------------------------------
constraint = ((length candidate) 3)
actual score = (entry-x2 (3 t))

modify-candidate   ok

-----------------------------------------
constraint = ((depth (first-atom
 candidate) candidate) 2)
actual score = (entry-x3 (1 nil))

  "find-diff"   (increase-depth-by 2)
  "apply-diff"
      reducer =  make-deeper-x
 new-candidate =  (((a)) b c)

modify-candidate   modified

-----------------------------------------

$The candidate's depth attribute has been
modified   by the modification routine
MAKE-DEEPER-X to produce a new example,
which is then judged and added to the
Examples-space:

   candidate value =  (((a)) b c)
     csc =  3   sf =  t
(entry-x1 (nlist t))
(entry-x2 (3 t))
(entry-x3 (3 t))
("created new frame for example "
 mar11-009 (((a)) b c))
"success!!"
```

Problem 3

```
$The CONSTRAINT-LIST for the next problem
is:

(x1 (desired-value list
     desired-prop (typep candidate)))
(x2 (desired-value 5
     desired-prop (length candidate)))
(x3 (desired-value 2
     desired-prop (depth
         (first-atom candidate)
 candidate)))
(x4 (desired-value 3
     desired-prop (depth
         (first-atom (cdr candidate))
         candidate)))

$The order of candidates retrieved and
judged is:

         <abc>
         <**digits>
         <**bits>
         <empty>
         <a>
         <abcde>
         mar11-009

$Since no example meets the constraints,
the modification phase is entered with
the following AGENDA:

         (<abcde> 2)
         (mar11-009 2)
         (<**bits> 1)
         (<a> 1)
         (<**digits> 1)
         (<abc> 1)
         (<empty> 0)

$The MODIFIER sets to work on the first
candidate (A B C D E):

-----------------------------------------
constraint = ((typep candidate) list)
actual score = (entry-x1 (lat t))

modify-candidate   ok

-----------------------------------------
constraint = ((length candidate) 5)
actual score = (entry-x2 (5 t))

modify-candidate   ok

-----------------------------------------
constraint = ((depth (first-atom
 candidate) candidate) 2)
actual score = (entry-x3 (1 nil))

  "find-diff"   (increase-depth-by 1)
  "apply-diff"
      reducer =  make-deeper-x
 new-candidate =  ((a) b c d e)

modify-candidate   modified

-----------------------------------------
constraint = ((depth (first-atom (cdr
 candidate)) candidate) 3)
actual score = (entry-x4 (1 nil))

  "find-diff"   (increase-depth-by 2)
  "apply-diff"
      reducer =  make-deeper-x
 new-candidate =  ((a) ((b)) c d e)

modify-candidate   modified

-----------------------------------------
   candidate value =  ((a) ((b)) c d e)
     csc =  4   sf =  t
(entry-x1 (nlist t))
(entry-x2 (5 t))
(entry-x3 (2 t))
(entry-x4 (3 t))
```

$The modification is successful and the new example is added to the Examples-space.

("created new frame for example " mar11-011 ((a) ((b)) c d e)) "success!!"

The Examples-space after the successful solution of Problems 2 and 3 is shown in Figure 4.

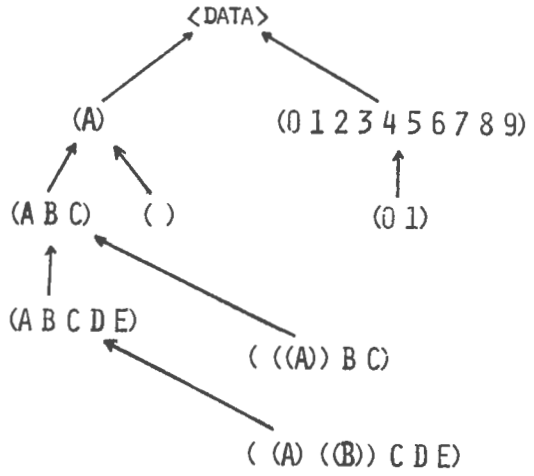



FIG 4

## 7. CONCLUSIONS

In this paper we have described a computer system that models Constrained Example Generation ("CEG") in domains from computer science and mathematics. We described how the CEG system generates examples of data in LISP.

We are currently using the system to explore issues such as

1. the effect of the initial contents of Example-space and the sequence of solved problems on the evolution of Examples-space;

2. the effect of alternative orderings on the retrieval and modification processes;

3. the effect of interacting constraints, e.g., impossible constraints.

We also plan to use our system to study machine learning by the incorporation of adaptive techniques, e.g., by keeping track of the performance of various orderings and techniques and choosing the ones that perform best. Such extensions of our system will enable it to "learn" from its own experience.

## References


Collins, A. and A. Stevens (1979) Goals and Strategies of Effective Teachers Bolt Beranek and Newman, Inc., Cambridge, Mass.

Friedman, D. (1974) The Little LISPer, Science Research Associates, Menlo Park, Calif.

Lakatos, I. (1963) Proofs and Refutations, British Journal for the Philosophy of Science, Vol. 19, May 1963. Also published by Cambridge University Press, London, 1976.

Lenat, D.B. (1976) An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search, Stanford Univ. Artificial Intelligence Memo 286.

Polya, G. (1968) Mathematics and Plausible Reasoning, Volumes I and II, Second Edition, Princeton Univ. Press, N.J.

Rissland (Michener), E. (1978a) Understanding Understanding Mathematics, Cognitive Science, Vol. 2, No. 4.

Rissland (Michener), E. (1978b) The Structure of Mathematical Knowledge, Technical Report No. 472, M.I.T Artificial Intelligence Lab, Cambridge.

Rissland, E. (1979) Protocols of Example Generation, internal report, M.I.T., Cambridge.

Soloway, E. (1978) "Learning = Interpretation + Generalization"; A Case Study in Knowledge-Directed Learning. Univ. of Massachusetts, COINS Technical Report 78-13, Amherst.

Winston, P. (1975) Learning Structural Descriptions from Examples, in The Psychology of Computer Vision, P. Winston (Ed.), McGraw-Hill, New York.

Woolf, B., and Soloway, E. (1980) Analysis of Student Protocals: Misconceptions in Understanding Programming in LISP, in preparation.


## Acknowledgments


Special thanks to my colleagues Elliot M. Soloway, for his invaluable assistance in bringing up the CEG system, and Oliver G. Selfridge, for many useful discussions and critiques.

# A Domain-Independent System for Developing Knowledge Bases

James A. Reggia
Depts. of Computer Science and Neurology
University of Maryland
College Park, Maryland 20742 USA

Abstract: This paper introduces a domain-independent system called KMS (a Knowledge Management System) that supports the development of knowledge-based consultant programs. At present KMS is based on a family of compatible subsystems. Each of these subsystems accepts knowledge stated in a non-procedural, natural language-like format that is understandable to a domain expert while still being processable by machine. The first generation of subsystems is discussed and some simple examples of encoding medical problem-solving knowledge are presented.

## Introduction

A growing concern of research in artificial intelligence (AI) during recent years has been the problem of representing and using (i.e., "managing") real-world knowledge in the development of applications-oriented computer programs. Examples can be found of knowledge-based consultants in a wide variety of domains including medicine, geology, signal processing, and biochemistry. The goal of these systems is to make expert-level decisions about problems that are faced by individuals working in these domains.

In this paper a system for creating and using such knowledge-based consultants is described. This system is based on a family of knowledge management languages and therefore is referred to as KMS (a Knowledge Management System). The term "management" is employed here to indicate that the use of knowledge (e.g. the selection of an appropriate inference method, the specific inferences to make) as well as its representation is of concern.

The KMS languages are applicable to a wide variety of domains, but they are presented here in the context of medical problem-solving because of the author's specific interest in that field. For that reason a brief introduction to computer-assisted medical decision making is provided below for those who are unfamiliar with recent work in this area. The philosophy behind KMS is then explained and the common features underlying its component languages are described. This is followed by some specific examples of clinical problem-solving knowledge encoded in the KMS languages that have already been implemented.

## Background

One research area where knowledge representation and use is a central issue is that of computer-assisted medical decision-making (CMD). Although several different approaches have been taken in the attempt to find appropriate ways to develop CMD systems [Reggia, 1979], each can be viewed within the framework of the simple conceptual model shown in Figure 1. In this model the nucleus of any CMD system is portrayed as having two basic components: a knowledge base of domain specific problem-solving information (e.g., a set of production rules), and a domain-independent inference method that makes decisions based on this knowledge base (e.g., a consequent-driven rule interpreter). In general, such CMD systems are given a description of a patient's symptoms, signs and laboratory test results and then draw upon their knowledge base to produce useful information about the patient's diagnosis, prognosis or treatment.

Three of the approaches to developing CMD systems are germane to the material that follows. First, statistical pattern classification based on Bayes' Theorem is one of the most common methods used to develop CMD systems. In this method the knowledge base is composed of the relevant prior and conditional probabilities, and Bayes' Theorem is used to infer a probability distribution for the possible outcomes (e.g., diagnoses). CMD systems based on Bayes' Theorem have several well-known inherent limitations, but for appropriately chosen medical problems they have been shown to work reasonably well and occasionally have performed more accurately than senior physicians (e.g., [deDombal, 1975]).

A second method for developing CMD systems involves the use of production systems. In these systems medical knowledge is captured as a collection of conditional rules that a rule interpreter uses to make decisions. Example medical domains where this AI approach has been adopted include the diagnosis and treatment of infectious diseases [Shortliffe, 1976] and glaucoma [Weiss, 1978]. While rule-based systems offer the advantages of declarative knowledge representation, modularity, and a limited explanation capability, it is often difficult for domain experts to formulate their knowledge as a set of rules and some forms of knowledge seem especially refractive to this approach [Goldberg, 1978; Reggia, 1978].

Finally, a third technique for developing CMD systems involves the construction of programs that are best described as cognitive models. The term 'cognitive model' is used here in the sense that these systems represent an explicit attempt to model the knowledge structures and diagnostic reasoning of the physician as it has been revealed in empirical research studies. Based on the results of these studies and intuition, medical knowledge is typically modeled as a network of "frames" and a hypothesize-and-test approach to decision making is used. Example medical domains that have been approached in this fashion include internal medicine [Pople, 1975], nephrology [Pauker, 1976], and cholestasis [Mittal, 1979].

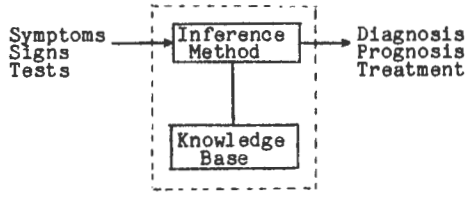



Figure 1: Conceptual model of a CMD system.

In spite of twenty years of research into the development of CMD models, such systems have had relatively little impact on the practice of medicine, even when they have been shown to be more accurate in their predictions than physicians. This is somewhat surprising in view of the potential benefits that CMD systems could bring to medicine and the comparatively successful use of decision support systems in a variety of other domains (e.g., business, engineering, industry, government; see [Kean, 1978]). The reasons for this relative lack of success are complex; we will return to them in a later section.

## The Architecture of KMS

The basic goal in creating KMS is to provide a domain-independent system that will serve as a "workbench" for the development of knowledge-based consultant programs. By doing this it is hoped that KMS will overcome some of the problems that have inhibited the development and use of such knowledge-based consultant programs in the past.

An overview of KMS is shown in Figure 2. Basically, KMS consists of a collection of n subsystems, each of which is organized around a different inference method. These domain-independent subsystems are overseen by the KMS executive.

Each of the subsystems supports a formal representation language. To create a knowledge-based consultant program a domain expert begins by selecting an appropriate subsystem (i.e. inference method). She or he then uses the University of Maryland text editor to write a domain-specific knowledge base in the corresponding representation language. Finally, that knowledge base is given to the appropriate subsystem of KMS which screens it for errors. If errors are found then diagnostic messages are generated and the knowledge base is rejected. The errors are then corrected using the text editor. Once a knowledge base is free of errors that are detectable by KMS it is accepted and converted into an internal form. Subsequent testing of the knowledge base leads to revision of the knowledge base and its resubmission to KMS. Eventually it may become part of one of the knowledge base libraries shown in Figure 2.

Each of the subsystems also supports a formal command language for using completed knowledge bases. A user signs onto KMS and requests a specific subsystem and knowledge base (perhaps being guided by the KMS executive in this task). He or she can then direct the system to perform various tasks: make specified inferences, justify a decision (where possible), display a subset of a knowledge base, etc. In performing these actions KMS carries on a dialog with the user, requesting problem-specific information where appropriate. A subsystem's representation and command languages taken together are considered to form a knowledge management language.

In summary, each of the KMS subsystems is a collection of programs that implement the following components.

> Inference system: When called into action this component will apply a stored knowledge base selected by the user to a specific problem. It makes inferences about that particular case that are of interest to the user.
>
> Knowledge base parser and interpreter: This component parses a knowledge base written in the appropriate KMS representation language and transforms it into an internal interpretable form. It is responsible for detecting syntactic and a limited number of semantic errors in a knowledge base. It is analogous to a parser in a standard programming language interpreter/compiler and performs the same useful services.
>
> Knowledge base compiler: This component stores a transformed knowledge base in its internal form. Later retrievals of that knowledge base do not require re-parsing or re-transformation to internal form and therefore are more efficient. The compiler is intended for use with completed, error-free knowledge bases.
>
> User interface: This component provides for interaction with a user. It includes the command language which is used to direct the control of the system, display part or all of a knowledge base, etc. In addition, by using information in a knowledge base, this component can request problem-specific information from a user in a simple, natural language-like format.

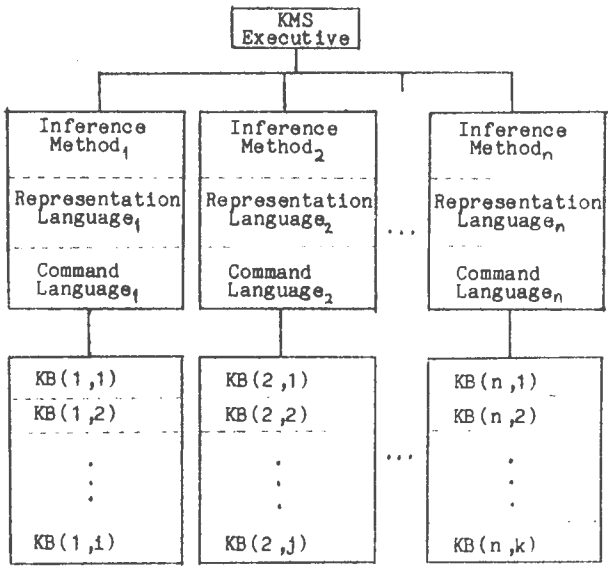



Figure 2: KMS architecture.

Not only is there the usual attention to modularity of elements in a knowledge base (intra-knowledge base modularity) in KMS but there is also inter-knowledge base, intra-subsystem, and inter-subsystem modularity.

## Inter-Subsystem Compatibility

The partition of KMS into a family of subsystems, each based on a different inference method and supporting a corresponding management language, raises the possibility of introducing a great deal of complexity into the system. To counter this the subsystems are designed to be compatible in two ways.

First, regardless of which subsystem is used, the underlying conceptual structure of a problem to be solved is built around a problem-oriented inference network. This network specifies the attributes that are of interest in a particular problem and the different values that they can have. In addition, the mechanism for describing such a network is similar in each of the

representation languages, and this leads to a uniform structure for all KMS knowledge bases. We will examine the idea of a problem-oriented inference network and its relationship to a knowledge base in the next section.

The second way in which the KMS subsystems are compatible is that the user interfaces are similar. The commands that give the user the ability to utilize a knowledge base are essentially the same for each subsystem. For example, regardless of the subsystem with which a knowledge base is associated, a user can direct KMS to OBTAIN <attribute>. This command tells KMS to use the currently active knowledge base to determine the value of the specified attribute. Another command found in all the languages is DISPLAY <option>. This command allows the user to view part or all of the knowledge base or the internal representation of a problem that is being solved. A third example is NEXT <option>. This command tells KMS that the user wants to discuss a new problem (NEXT CASE) or that the system should switch to a new knowledge base (NEXT KB).

In addition to sharing similar commands, the user interfaces of KMS subsystems all ask the user for information about a specific problem in approximately the same way. The name of an input attribute whose value is desired by KMS is printed out and the user simply selects from among its possible values in responding. This multiple-choice format is useful for constraining the inferences made by the system [Rieger, 1978], presupposes no special typing skill of the user, and avoids the need for a sophisticated natural language interface.

From Attribute Hierarchy to Knowledge Base

In the previous section it was stated that, regardless of which subsystem is used, the underlying conceptual structure of a problem to be solved is built around a problem-oriented inference network or, more simply, an attribute hierarchy (see Figure 3). The term 'problem-oriented' indicates that each such network is centered around a specific domain problem. The network specifies both the structure of that problem as well as the "direction" in which inferences are to propagate. Its nodes are the attributes of the problem being solved. Conceptually associated with each of these attributes is a set of possible values which that attribute can have. The links in the network represent the relation "depends on" (or conversely, "determines") indicating that the value of one attribute (drawn in a superior position) is determined by the values of others (drawn in inferior positions). Thus, the lowest level of an attribute hierarchy consists of input attributes whose values are determined by a user; other nodes in the network represent inferred attributes whose values are determined by the system.

Figure 3 gives two simple examples of attribute hierarchies dealing with medical problems. The one-level tree in Figure 3a indicates that the NEUROLOGICAL OUTCOME of a patient with coma following cardiac arrest is determined by the four input attributes shown [Snyder, 1977]. Similarly, the three-level attribute hierarchy in Figure 3b represents the organization for a knowledge base that will stage a patient's lung cancer and generate a two-year prognosis for that patient [Rosenow, 1979]. Since there may be more than one "most superior" node and as a node may have more than one parent, an attribute hierarchy is not necessarily a tree structure as illustrated in these examples.

The development of any knowledge base is organized around an attribute hierarchy similar to those illustrated in Figure 3. For our purposes we will view knowledge bases as consisting of two parts: a knowledge schema and an associations section. The schema consists of a list of the names of the attributes in an attribute hierarchy along with the names of their possible values. For example, the input attribute REGIONAL NODE INVOLVEMENT in Figure 3b and its possible values might be declared as

```
REGIONAL NODE INVOLVEMENT:
        NONE
      / PERIBRONCHIAL NODES
      / HILAR NODES
      / MEDIASTINAL NODES;
```

in the schema of the corresponding knowledge base.

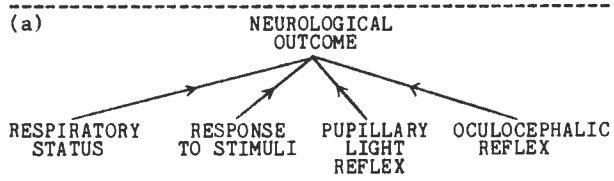
The associations section of a knowledge base, which may overlap in part with the schema, provides information that associates the possible values of different attributes to one another. Such associative links are formed through the use of implicit or explicit KMS statements. An explicit KMS statement has the form

<attribute> <relation> <value>

such as

REGIONAL NODE INVOLVEMENT = HILAR NODES.

Statements are combined to define the partial (non-circular) ordering of the inference network that determines the direction in which information is to propagate (see below).



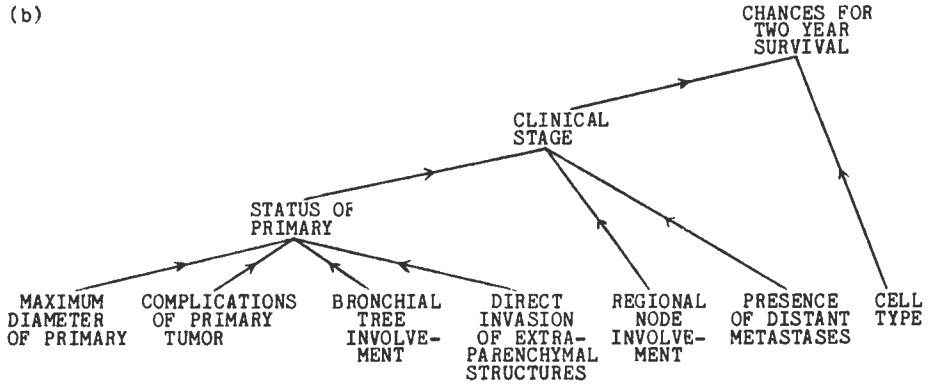




Figure 3: Some simple attribute hierarchies. (a) Predicting neurological outcome following cardiac arrest. (b) Staging a lung tumor and generating a prognosis.

The effect of expressing knowledge in terms of a problem-oriented inference network or attribute hierarchy is to provide a non-procedural framework for knowledge that is understandable to computer-inexperienced individuals. It therefore leads to the formulation of knowledge bases that are easily read and understood by domain experts and users while still being processable by computer.

## The First Generation of Subsystems

The initial version of KMS consists of three subsystems (n = 3 in Figure 2). The first versions of two of these subsystems have been implemented and are described below. The third subsystem is currently being designed. All programming is in LISP and the system is accessed via terminals connected to the University of Maryland Instruction and Research Network. Although domain-independent, KMS is currently being studied in the context of developing CMD systems, especially in neurology.

The first subsystem, KMS.BAYES, requires appropriate information for statistical pattern classification based on Bayes' Theorem (see 'Background' above or [Duda and Hart, 1973]). The prior probabilities of outcomes (e.g., diagnoses, prognoses) and the conditional probabilities of manifestations of each outcome (e.g., signs, symptoms) must be specified in a knowledge base. Bayes' Theorem was selected as an inference method because of its widespread use in CMD systems. In addition, it is suitable for making use of probabilities that are frequently found in journal articles reporting clinical studies or that are produced by medical databases.

Figure 4 illustrates a very small example of a KMS.BAYES knowledge base. This particular example deals with predicting a patient's outcome following cardiac arrest and corresponds to the attribute hierarchy illustrated in Figure 3a. Lines beginning with an asterisk are comments.

```
    *** CPR PROGNOSIS KNOWLEDGE BASE CPR2 ***
  * PREDICTION OF NEUROLOGICAL OUTCOME FOLLOWING
  * CPR BASED ON NEUROLOGICAL EXAMINATION DONE
  * WITHIN ONE HOUR AFTER CPR. ASSUMES NO PRE-
  * EXISTING BRAIN DAMAGE AND NO TOXIC/METABOLIC
  * ENCEPHALOPATHY. ASSUMES 24 HOUR SURVIVAL.
  * PREDICTED OUTCOMES ARE: FUNCTIONAL (NORMAL OR
  * SELF-CARE WITH SUPERVISION), IMPAIRED
  * (SEVERE DEMENTIA OR PERSISTENT VEGETATIVE
  * STATE). REFERENCE: SNYDER ET AL, NEUROLOGY,
  * 27, 1977, 807-811.

  * INPUT ATTRIBUTES
    RESPIRATORY STATUS:
        SPONTANEOUS ACTIVITY /
        ON RESPIRATOR AND NOT TRIGGERING.
    RESPONSE TO STIMULI:
        PURPOSEFUL RESPONSE TO PAIN /
        NON-PURPOSEFUL OR NO RESPONSE TO PAIN.
    PUPILLARY LIGHT REFLEX: PRESENT / ABSENT.
    OCULOCEPHALIC REFLEX: PRESENT / ABSENT.

  * INFERRED ATTRIBUTE
    NEUROLOGICAL OUTCOME (RESPIRATORY STATUS;
        RESPONSE TO STIMULI; PUPILLARY LIGHT
        REFLEX; OCULOCEPHALIC REFLEX):
            FUNCTIONAL(0.62):
              0.67 0.33;
              0.57 0.43;
              0.89 0.11;
              0.69 0.31 /
            IMPAIRED(0.38):
              0.15 0.85;
              0.15 0.85;
              0.42 0.58;
              0.33 0.67 $
```

Figure 4: A simple KMS.BAYES knowledge base.

KMS.BAYES can use this knowledge base to compute a probability distribution for the values of the attribute NEUROLOGICAL OUTCOME when it is given a value for the four input attributes (e.g., RESPIRATORY STATUS). Only implicit KMS statements are illustrated here. For example, the prior probability of NEUROLOGICAL OUTCOME = FUNCTIONAL is 0.62 (indicated in parentheses). The input attributes that NEUROLOGICAL OUTCOME depends on are indicated in parentheses following its name. In general these determining attributes may be a subset of the declared input attributes because there can be more than one inferred attribute. The appropriate conditional probabilities for the values of the determining input attributes are listed in an order corresponding to that of the attribute names in parentheses after the name NEUROLOGICAL OUTCOME. For example, of patients in Snyder's series that ended up in the FUNCTIONAL category, 89% had PUPILLARY LIGHT REFLEX = PRESENT immediately after cardiac arrest while only 11% had PUPILLARY LIGHT REFLEX = ABSENT. Although not shown here, the number of values of an attribute can be greater than two.

Figure 5 demonstrates part of an interaction mediated by KMS between the knowledge base shown in Figure 4 and a user. It demonstrates how questions are generated from a knowledge base. User responses have been underlined for clarity.

The second subsystem, KMS.PS, involves representing knowledge about a problem as a collection of production rules. A rule-based subsystem was selected for inclusion in the first version of KMS because of the demonstrated usefulness and popularity of production systems in knowledge engineering. Production rules have proven especially suitable for capturing some types of "judgemental" knowledge.

```
          ...
>>> ENTER COMMAND:

obtain neurological outcome.

>>> ENTER INITIAL INFORMATION:

none.

>>> RESPIRATORY STATUS:
      (1) SPONTANEOUS ACTIVITY
      (2) ON RESPIRATOR AND NOT TRIGGERING
= ? 1

>>> RESPONSE TO STIMULI:
      (1) PURPOSEFUL RESPONSE TO PAIN
      (2) NON-PURPOSEFUL OR NO RESPONSE TO PAIN
= ? 2

>>> PUPILLARY LIGHT REFLEX:
      (1) PRESENT
      (2) ABSENT
= ? 1

>>> OCULOCEPHALIC REFLEX:
      (1) PRESENT
      (2) ABSENT
= ? 1

>>> NEUROLOGICAL OUTCOME =
      FUNCTIONAL: 0.94
      IMPAIRED: 0.06

>>> ENTER COMMAND:

next case.

>>> READY FOR NEXT CASE
>>> ENTER INITIAL INFORMATION:

          ...
```

Figure 5: Part of a KMS.BAYES session using the knowledge base in Figure 4.

The rules in a KMS.PS knowledge base undergo procedural interpretation using a top-down, depth-first strategy (note the similarities to PROLOG [Futo et al, 1978]). The model of inexact reasoning originally introduced in MYCIN [Shortliffe, 1976] is used to propagate "certainty factors" from a rule's anteoedents to its consequents. Some examples of rules written in KMS.PS are shown in Figure 6. Space restrictions prohibit showing the complete collection of over 130 rules for diagnosing thyroid dysfunotion that are in this particular knowledge base. Each rule consists of a name, one or more antecedents, and one or more consequents. Each antecedent and consequent is an explicit KMS statement. Certainty factors are specified in parentheses following consequent statements with a default value of 1.0 if none is given.

Figure 7 shows two additional rules from another knowledge base for staging a patient's lung cancer and generating a prognosis. These rules demonstrate how disjunctions can be incorporated into antecedents and illustrate the '#' relation (for 'not equal to'). Other possible relations exist for attributes with numeric values (e.g., 'GE' for 'greater than or equal to'). These rules also show how the attribute hierarchy in Figure 3b is implicitly inoorporated into a knowledge base.

An interactive session involving a KMS.PS knowledge base appears very similar to one involving a KMS.BAYES knowledge base (Figure 5) from the user's point of view. Thus this is not illustrated here. One command that is available from KMS.PS but not from KMS.BAYES is JUSTIFY <attribute> = <value>. KMS.PS keeps track of which rules it uses to assign a value to an inferred attribute and this command tells KMS to state the names of those rules for a particular value. This forms the basis for a limited explanation capability.

A third subsystem, KMS.HT, is currently being designed. Although KMS.HT is most similar to cognitive models as described above (see 'Background') the exact resemblance to human reasoning and knowledge organization is not the major concern. The main emphasis in developing KMS.HT is to provide a convienient means for representing and using (medical) problem-solving knowledge as it appears in review articles and textbooks. Knowledge bases associated with KMS.HT will be organized around frame-like structures that will be incorporated by the system into a problem-oriented inference network. A hypothesize-and-test control strategy will be adopted.

```
TEST RULE10
IF FREE THYROXINE INDEX = HIGH
 & T4-RIA VALUE = NORMAL
 & RT3U VALUE = HIGH
THEN SCREENING TEST RESULTS = HYPERTHYROID (0.9);

SCAN RULE6
IF SCAN INTERPRETATION = HOT NODULES
 & TSH SUPPRESSION SCAN = NEGATIVE
THEN SCAN RESULTS = AUTONOMOUS NODULAR GLAND;

CLINICAL RULE24
IF PRELIMINARY CLINICAL EVALUATION = HYPERTHYROID
 & INFERRED THYROID ARCHITECTURE = DIFFUSE GOITER
 & SCAN RESULTS = AUTONOMOUS DIFFUSE GLAND
THEN FINAL CLINICAL EVALUATION = GRAVES DISEASE
 & STATUS OF FINAL CLINICAL EVALUATION
                    = RECOGNIZED PATTERN;

DIAGNOSIS RULE13
IF FINAL LAB TEST RESULTS = HYPERTHYROID
 & FINAL CLINICAL EVALUATION = GRAVES DISEASE
THEN DIFFERENTIAL DIAGNOSIS = GRAVES DISEASE (0.9)
 & STATUS OF DIFFERENTIAL DIAGNOSIS
                    = RECOGNIZED PATTERN;
```

Figure 6: Example KMS.PS rules for diagnosing thyroid dysfunction.

```
STAGING RULE6
IF STATUS OF PRIMARY = LEAST SEVERITY
 & REGIONAL NODE INVOLVEMENT = PERIBRONCHIAL NODES
     / REGIONAL NODE INVOLVEMENT = HILAR NODES
 & REGIONAL NODE INVOLVEMENT # MEDIASTINAL NODES
 & PRESENCE OF DISTANT METASTASIS = NONE KNOWN
THEN CLINICAL STAGE = STAGE ONE;

PROGNOSIS RULE2
IF CLINICAL STAGE = STAGE ONE
 & CELL TYPE = LARGE
THEN
  CHANCES FOR TWO YEAR SURVIVAL = ONE IN THREE;
```

Figure 7: Example KMS.PS rules on lung cancer.

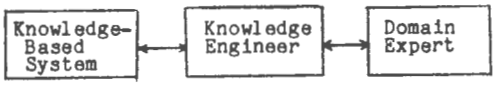
## Related Work

KMS bears a resemblance to some other AI research efforts currently in progress. Its purpose and goals are in some ways similar to those of several general knowledge representation languages (e.g., KRL [Bobrow and Winograd, 1977], KLONE [Brachman, 1979], SBDS [Ohsuga, 1979]). Even more closely related are those systems that have evolved from research into developing expert consultation programs. For example, the rule-oriented software originally developed for MYCIN (called EMYCIN for "Essential MYCIN") has been used to build consultation systems in several different areas of medicine as well as an engineering domain [van Melle, 1979]. A second domain-independent rule-based system is EXPERT, a generalized descendent of the CASNET formalism [Weiss and Kulikowski, 1979]. It is currently being used to develop consultation models in rheumatology, ophthalmology, and endocrinology. Finally, another rule-based system for building consultation programs is AGE [Nii and Aiello, 1979]. AGE has been used to implement a consultation system dealing with pulmonary function test interpretation.

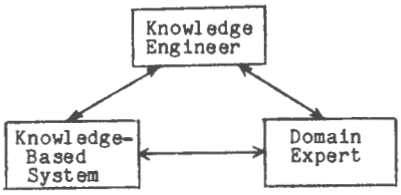
KMS differs from all of these other domain-independent software laboratories in at least two very important ways. First, KMS is decomposed into a collection of subsystems, each of which is based on a different inference method and representation format. This approach to the architecture of KMS reflects the belief that there is no single "best" method for representing and using knowledge. On the contrary, there is a variety of methods that are available, each with certain advantages and disadvantages. The selection of which method to use for a given problem depends on several factors such as the structure of the problem involved and the availability of appropriate problem-solving knowledge. Almost all of the domain-independent systems described above are oriented around a single formalism for managing knowledge, and in this sense approximate a single subsystem in KMS.

A second significant difference between KMS and the systems described above is one of emphasis. KMS is based on the belief that the best way to develop knowledge-based consultants is by permitting a human domain expert to transfer directly his or her knowledge to the computer. This is why KMS emphasizes subsystem-supported languages whose primitive elements are the attributes, values, and associations of a particular domain problem. Many of the other systems described above require a fair knowledge of LISP or other aspects of AI because of the additional expressive power this provides. Individual KMS subsystems sacrifice some of this power in exchange for being directly usable by domain experts after minimal instruction.

Of course, this direct usability by domain experts raises the question about what the role of the knowledge engineer should be in the context of a system like KMS. The more traditional viewpoint has been that the knowledge engineer should serve as an intermediary between domain experts and the computer, helping the human expert to express her or his problem-solving knowledge [Amarel, 1977; Feigenbaum, 1977]. This is depicted in Figure 8a.




(a) The traditional approach




(b) The KMS approach

Figure 8: Alternate views of the knowledge engineer.

The KMS viewpoint is that the knowledge engineer should occupy a different position dealing predominantly with epistemological issues (Figure 8b). Specifically, such an individual would create and modify the KMS subsystems, educate domain experts and users about the system, and be available for consultation as specific knowledge bases are developed.

## Facing the Problems of CMD

It was noted in the 'Background' section above that CMD systems have had relatively little effect on the day-to-day practice of medicine. There are several reasons for this and they have been discussed extensively in the literature (e.g., [Croft, 1972; Friedman, 1977; Mitchell, 1970; Startsman, 1972]). Some of these problems that have inhibited the development and use of CMD systems are not addressed by KMS: the lack of adequate databases with relevant clinical information, the lack of standardization in medical definitions, etc. However, KMS does attempt to alleviate four of the problems that are frequently mentioned.

1) The physician-computer interaction has not been successfully accomplished.

A distinguishing feature of KMS is that it is designed for direct use by physicians, both as users and as domain experts. The growing availability of computer facilities in clinical settings makes this a realistic possibility.

2) The translation of clinical knowledge into a form suitable for computer processing and the implementation of the programs to process it are difficult and time consuming tasks.

Each KMS subsystem includes all the software needed to implement a complete CMD system (inference method, user interface, etc.) once it is given an appropriate knowledge base to work with. Thus no additional programming is necessary. Knowledge acquisition remains a significant problem, but it is at least improved by the direct use of formal representation languages by physicians. In addition, KMS expedites knowledge acquistion by providing a collection of languages suitable for different types of knowledge and by detecting certain classes of errors in knowledge bases.

3) There has been a lack of acceptance of CMD systems by the medical community.

This in part reflects the fact that physicians have not been convinced that CMD systems can be generally useful. Often the emphasis by research workers has been on producing systems that do what the physician does, and not surprisingly this has met with only limited acceptance by physicians. It is hoped to alleviate this problem within the framework of KMS by stressing support for decision-making rather than by producing computer-generated decisions. Also, since KMS is directly usable by physicians, it gives a physician the freedom to implement a knowledge base that is of specific interest to him or her. In the long run this may prove to be one of KMS's most important features. Finally, by providing a library of knowledge bases KMS has the potential for accumulating the "critical mass" of information that would be necessary to justify the time required to learn its use. This is not true with conventional CMD systems that address only one problem with a single knowledge base (see 'Background').

4) Even when successful most CMD systems cannot convieniently be transferred to different installations.

While KMS is theoretically portable to any facility that supports the LISP language, the numerous dialects of LISP make this a less than ideal prospect. However, the real issue is not the portability of KMS itself, but whether or not knowledge bases (not programs) can be made easily transferable from one installation to the next. Since KMS supports machine-independent representation and command languages, any computer facility with an implemented version of KMS would be able to use KMS knowledge bases developed at other sites. While a great deal of work remains to be done, the ideas behind KMS at least have the potential to provide a qualitative improvement in the portability of CMD systems.

## Conclusion

This paper has introduced KMS, a new approach to managing the real world knowledge needed in knowledge-based consultant programs. The major features of the architecture of KMS center on a family of compatible subsystems that are based on different inference methods. These subsystems all address knowledge representation in a similar fashion, requiring the description of a problem to be in terms of its attributes, their values, and the associations between them. They also share a common set of control commands and have similar user interfaces. Associated with each subsystem is a collection of knowledge bases written in the appropriate fashion. A domain expert can use these KMS subsystems to build a library of knowledge bases that deal with one or more domain-specific problems.

At present KMS is best characterized as an experiment-in-progress with many questions remaining to be answered about its ultimate utility. For example, will the use of simple inference methods significantly limit the power of the subsystems? If one accepts the belief that "the problem solving power exhibited in an intelligent agent's performance is primarily a consequence of the specialist's knowledge employed by the agent, and only very secondarily related to the generality and power of the inference method employed" [Feigenbaum 1977] then the ability of KMS to manage libraries of knowledge bases gives reason for optimism. Is the direct domain expert-computer interaction really feasible? The growing diffusion of computer technology throughout society and the increasing computer-sophistication of individuals in various non-computing disciplines at least makes this a possibility.

These and other questions will be examined through KMS in the future. The present plan is to complete KMS.HT and then evaluate the first generation of subsystems by developing a small prototype library of knowledge bases. The use of KMS by computer-inexperienced individuals (medical students, physicians, etc.) will also be evaluated. Hopefully, KMS or similar systems will ultimately help to make the computer a useful tool for a number of individuals for whom it has previously been relatively inaccessible.

Acknowledgements: The research described in this report is funded by an NIH Teacher-Investigator Development Award (5 K07 NS 00348) from the NINCDS. Computer time is provided in part by the Computer Science Center of the University of Maryland. The example rules on thyroid diagnosis are from a rule collection written by Barry Perricone. This is the second report of the NEUREX project.

References

Amarel S et al: Applications of Artificial Intelligence (Panel), Proc. Fifth IJCAI, 1977, 994-1006.

Bobrow D and Winograd T: An Overview of KRL, a Knowledge Representation Language, Cognitive Science, 1, 1977, 3-47.

Brachman R: On the Epistemological Status of Semantic Networks, in Associative Networks, N. Findler (editor), Academic Press, 1979, 3-50.

Croft D: Is Computerized Diagnosis Possible?, Comp. Biomed. Res., 5, 1972, 351-367.

deDombal F: Computer-Assisted Diagnosis of Abdominal Pain, in Advances in Medical Computing, Rose and Mitchell (editors), Churchill-Livingston, 1975, 10-19.

Duda R and Hart P: Pattern Classification and Scene Analysis, John Wiley and Sons, 1973.

Feigenbaum E: The Art of Artificial Intelligence - Themes and Case Studies of Knowledge Engineering, Proc. Fifth IJCAI, 1977, 1014.

Friedman R and Gustafson D: Computers in Clinical Medicine - A Critical Review, Comp. Biomed. Res., 10, 1977, 199-204.

Futo I, Darvas F and Szeredi P: The Application of PROLOG to the Development of QA and DBM Systems, in Logic and Data Bases, Gallaire and Minker (editors), Plenum Press, 1978, 347.

Goldberg R and Kastner J: An Explicit Description of Anatomical Knowledge as an Aid to Diagnosis, CBM-TR-78, Dept. of Computer Science, Rutgers University, Oct. 1978.

Kean P and Morton M: Decision Support Systems - An Organizational Perspective, Addison-Wesley, 1978.

Mitchell J: The Automation of Clinical Diagnosis, Bio-Med. Comp., 1, 1970, 157-166.

Mittal S, Chandrasekaran B, and Smith J: Overview of MDX - A System for Medical Diagnosis, Proc. Third Ann. Symposium on Comp. Applic in Med. Care, IEEE, Oct. 1979, 34-46.

Nii H and Aiello N: AGE (Attempt to Generalize) - A Knowledge-Based Program for Building Knowledge-Based Programs, Proc. Sixth IJCAI, 1979, 645-655.

Ohsuga S: Theoretical Basis for a Knowledge Representation System, Proc. Sixth IJCAI, 1979, 676-683.

Pauker S et al: Towards the Simulation of Clinical Cognition, Amer. J. Med., 60, 1976, 981-996.

Pople H, Myers J and Miller R: A Model of Diagnostic Logic for Internal Medicine, Proc. Fourth IJCAI, 1975.

Reggia J: A Production Rule System for Neurological Localization, Proc. of the Second Annual Symposium on Computer Application in Medical Care, IEEE, Nov. 1978, 254-260.

Reggia J: Computer-Assisted Medical Decision Making - Knowledge Bases, Proc. Third Annual Symposium on Computer Application in Medical Care, IEEE, Oct. 1979, 28.

Rieger C: The Importance of Multiple Choice, TINLAP-II, University of Illinois, July 1978.

Rosenow E and Carr D: Bronchogenic Carcinoma, Ca - A Cancer Journal for Clinicians, American Cancer Society, 29, 1979, 233-245.

Shortliffe E: Computer-Based Medical Consultations - MYCIN, American Elsevier, 1976.

Startsman T and Robinson R: The Attitudes of Medical and Paramedical Personnel Toward Computers, Comp. Biomed. Res., 5, 1972, 218-227.

Snyder B, Ramirez-Lassepas M, and Lippert D: Neurologic Status and Prognosis After Cardiopulmonary Arrest, Neurology, 27, 1977, 807-811.

van Melle W: A Domain-Independent Production Rule System for Consultation Programs, Proc. Sixth IJCAI, 1979, 923-925.

Weiss S, Kulikowski C, and Sapir A: Glaucoma Consultation by Computer, Comp. Biol. Med., 8, 1978, 25-40.

Weiss S and Kulikowski C: EXPERT - A System for Developing Consultation Models, Proc. Sixth IJCAI, 1979, 942-947.

# Knowledge Acquisition and Representation Using Logic, Set Theory and Natural Language Structures

Stewart Bainbridge and Douglas Skuce * †
Departments of Mathematics and Computer Science (resp.)
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5

## Abstract

We present an approach to acquiring qualitative generic knowledge from experts using a language LESK, and show how the semantics of LESK, for both human and machine understanding, can be expressed in a "deep structure" language, ARC. LESK gives a "natural language" surface syntax to ARC, which compactly represents a set-theoretic interpretation of common predicate calculus expressions. A theorem prover for ARC has been implemented in PROLOG.

The design of LESK and ARC have been driven by real examples of such knowledge which describe the conceptual structure of the Canadian census database.

## 1. Introduction

The acquisition of generic knowledge (gk) from experts (not necessarily computer specialists) and its representation in forms well-suited for either analysis by these experts or by deductive knowledge base systems (kbs) is an increasingly important aspect of modern information systems. We describe an approach to expediting these two tasks using two closely related languages, LESK (Language for Exactly Stating Knowledge; Skuce 75-79) and ARC (Algebra of Relational Composition), intended respectively for the two uses referred to above. LESK, a kind of predicate calculus, provides a linguistically natural medium for humans, while ARC provides an efficient formalism for implementing a kbs which can interactively acquire new knowledge and deductively answer questions. Each LESK statement translates readily into ARC, and/or an associated "declaration" language, DARC. (DARC serves a purpose analogous to declaration statements in a programming language.) To the trained reader, ARC and DARC are superior to predicate calculus (PC) and conventional mathematical notation for semantic verification of LESK.

The eventual goal of this research is an interactive kbs which assists in the acquistion of linguistically and logically consistent generic knowledge from persons requiring computer processing of information dependent on this knowledge. After acquiring the gk, the system would serve as an "oracle" in helping humans understand it, or in interfacing to other systems which could use it, such as a conventional database system. By the term "LESK system" we shall mean such a system based on LESK and ARC.

We report here on:
1. the basic ideas of LESK and ARC;
2. the use of LESK in a typical gk acquisition task: capturing the terminology and logical structure of the concepts to be represented in a database system (dbs);
3. implementation of a LESK system.

## 2. Basic Concepts of ARC

LESK is a linguistically natural surface language whose semantics are based on set theory and predicate calculus (PC), which ARC represents more directly without LESK's natural language (NL) sugar. We therefore begin by describing ARC. (A detailed specification of ARC and its deductive structure will appear elsewhere; here we shall sketch the general features of the language and illustrate the inference rules with a simple example.)

* alphabetical order

ARC has terms of two types: terms which denote sets, and terms which denote binary relations (viewed as sets of ordered pairs). For expository purposes, the semantics of ARC will be specified in PC. For example, the ARC set term "person" would be represented in PC by a unary predicate "person(x)", and the ARC relation term "uncle" would be represented in PC by a binary predicate "uncle(y,x)". (Our convention for reading PC expressions "R(y,x)" is "y is a R of x", so in the preceding, y is the uncle, x the nephew/niece.) The examples below will clarify how ARC manages without individual variables. The type of primitive terms is declared by DARC statements such as "set(person)", "rel(uncle)".

Terms of like type may be combined by Boolean operations, and terms of different or like types may be combined by means of operations including those defined below. We give the definitions in PC with the understanding that the ARC relation term "R" denotes the set of pairs (y,x) which make the corresponding PC term "R(y,x)" true, and similarly for ARC set terms.

Composition: for relations R, S, define the relation (R of S) by

(R of S)(y,x) iff $\exists z$ R(y,z) and S(z,x)

Image: for a relation R and a set A, define the set (R of A) by

(R of A)(y) iff $\exists x$ R(y,x) and A(x)

Product: for sets A, B, define the relation (B × A) by

(B × A)(y,x) iff B(y) and A(x)

Inversion: for a relation R, define the relation inv(R) by

inv(R)(x,y) iff R(y,x)

In addition there are certain set and relation terms with fixed meanings such as the set "universe" of which all sets are assumed to be subsets, the empty set "null", the identity relation "id" (id(y, x) iff y = x), greater than "gt" (gt(y,x) iff y > x), and so on. ARC has built-in axioms and/or inference rules for these special terms.

Statements in ARC are equations or set inclusions between ARC terms of like type.

The inference rules for ARC include Boolean inference rules and other rules and axioms for the additional operations, including the following, where X is either a set or relation term:

from X $\subset$ Y infer (R of X) $\subset$ (R of Y)

from R $\subset$ S infer (R of X) $\subset$ (S of X)

associativity of "of": R of (S of X) = (R of S) of X

The operations and inference rules given above are sufficient for the examples which follow, but do not constitute a complete description of ARC.

Example:

The following example illustrates the non-Boolean inference rules of ARC. No attempt is made to indicate how the proof would be discovered by the theorem prover, we simply exhibit it. Please note that the "of" operator conceals existential quantification and that despite the algebraic appearance of the proof, it is not simply propositional logic. We urge the reader not to dismiss this example as trivial without first examining in detail what is involved in the proof in PC.

Axioms:

LESK: the paternalgrandfather of X = the father of the father of X
ARC: pg = f of f
PC: $\forall y,x$ pg(y,x) iff $\exists z$ f(y,z) and f(z,x)

LESK: the age of the father of X > the age of X
ARC: a of f $\subset$ gt of a
PC: $\forall y,x$ if $\exists z$ a(y,z) and f(z,x) then $\exists w$ y > w and a(w,x)

LESK: if Y > Z and Z > X then Y > X
ARC: gt of gt $\subset$ gt
PC: $\forall y,x$ if $\exists z$ y > z and z > x then y > x

(note: the PC given is the literal translation of the ARC, rather than the (logically equivalent) form stated in LESK.)

Proposition:

LESK: the age of the paternalgrandfather of X > the age of X
ARC: a of pg $\subset$ gt of a
PC: $\forall y,x$ if $\exists z$ a(y,z) and pg(z,x) then $\exists w$ y > w and a(w,x)

Proof:

a of pg = a of (f of f)
(substitution using first axiom)

(a of f) of f $\subset$ (gt of a) of f
( (second axiom) of f )

gt of (a of f) $\subset$ gt of (gt of a)

( gt of (second axiom) )

(gt of gt) of a $\subset$ gt of a
( (third axiom) of a )

The result now follows by associativity of "of" and Boolean inferences.

It appears that a large number of the questions one would ask of a LESK system involve deductions not much more complicated than this, using (except for Boolean inferences) only the inference rules given above. It is for this type of deduction that we suggest ARC is especially well suited.

## 3. Criteria for LESK, Language and System

We have the following design goals for LESK as a language:

1. It emphasizes the linguistic aspect of knowledge acquistion. Since the knowledge we are concerned with is virtually all qualitative, it is normally expressed mainly in NL, using terminology which is often not well controlled. LESK enforces a standardized use of terminology upon the user, as a first step in removing semantic errors.

2. It provides statement forms which are unambiguous, and which assist the user in formulating gk in desirable ways, in particular, in conceptual hierarchies. The semantics of these forms should be readily explainable in ARC, or if necessary, PC.

4. Knowledge expressed in LESK should be understandable to anyone capable of learning a programming language such as PASCAL, and someone with a modest mathematical training and ability should be able to write LESK statements.

LESK requires the user to make declarations of the essential terminology (principally nouns, adjectives and various verb forms) in a restricted subject in a linguistically natural yet logically precise manner. By "declaring" terminology we mean a formal process of introducing words in restricted contexts, so that their syntactic and semantic properties are unambiguous. A LESK system would aid the user in making consistent use of this terminology, much as a compiler for a programming language does. With such a system, the process of developing a kbs in this manner would involve a three party communication between user (originating the concepts), other persons trying to understand the concepts in order to implement the kbs for the user (e.g. a database designer), and the LESK system as an "intelligent" assistant.

## 4. An Example: The Use of LESK to Capture Database Concepts

The design of database systems (dbs) usually presumes a previous step in which terminology and logical relations have been clarified. It is felt that this vital first step is often not given sufficient attention, resulting in a dbs which contains errors due to semantic confusion between user and designer. LESK can serve an important role in dbs design by reducing the possibility of such errors.

Having noted this potential, Statistics Canada invited us to attempt to apply LESK to clarifying the conceptual structure of their census database. Although this structure has evolved over many years, and is not about to be drastically redesigned, this application provided a real and challenging example of gk requiring formal explication. Previously, the gk associated with the census was to be found in a variety of forms: narrative in various internal documents, some containing glossaries; in (very limited) formalized rule systems used to drive certain software packages; in "data definition" statements; in code alone, or even only in people's heads. Our goal was to aggregate, condense and make precise in a uniform manner a representative sample of this gk; the appendix displays a fragment of this sample.

The gk of which the appendix is representative was obtained during ten to fifteen hours of dialogue between Skuce and senior personnel of Statistics Canada's System Development Division. During this time each was teaching the other a set of new concepts, using LESK to aid the transfer of census concepts to Skuce. A frequent difficulty was that the logical and lexical precision demanded by LESK caused indecision as to which of several alternatives to choose. Since

sometimes there was no clear authoritative source to resolve these questions, opinions were sought, or a solution was improvised. Thus the process was both one of knowledge acquisition and formulation. It is highly likely that virtually any organization would exhibit similar problems.

We will now discuss the major aspects of LESK illustrated in the appendix. Terminology is introduced either in series of statements termed declarations (ending with a period) or in assertions which are single statements. The terminology here involves either nouns, adjectives, or simple stative sentences denoting binary relations by phrases such as "is related to" or "is an ancestor of". Such terms are declared to be in "is a" hierarchies whenever possible. The primitive "kinds", denoting class partition, statements of the form "x is a y", or the prefixing of an adjective to a NP y all result in subclasses of y. Unless preceded by "iff" or "consists of" (used in a noun declaration to be considered as a record), statements in declarations are necessary conditions. The abbreviation "-" denotes the definiendum. Syntactic details like singular and plural forms are declared elsewhere.

Thus we first introduce (i.e. declare) the noun person; we see that persons are partitioned three ways, first, into the subclasses livingpersons and deadpersons. The resulting ARC statements are:

livingpersons ∪ deadpersons = persons
livingpersons ∩ deadpersons = null

The second partition specifies a partitioning function, age, hence we have:

adults = inv(age) of ge of 17
∩ persons
children = inv(age) of lt of 17
∩ persons
adults ∩ children = null
age of (inv(age) ∩ (persons × numbers))
⊂ id
id ∩ (persons × persons) ⊂ inv(age) of age

Note that:
a.) a relation R is a function iff (R of inv(R)) ⊂ id and id ⊂ (inv(R) of R); the last two axioms thus state that age restricted to persons is a function;
b.) the disjointness of adults and children is made explicit even though it is already implied by properties of age and lt, ge.

The partition of persons by sex is similar.

We next decide to add some constraints to the functions age and sex.

age of persons ⊂ lt of 115
sex of persons = male ∪ female
male ∩ female = null

Here male and female denote atomic sex values, not the sets of all males and of all females. This is indicated in DARC by atom(male) and atom(female).

The functions dwelling and parents are introduced next; the only ARC statements are those expressing functionality as before. The denotes uniqueness, i.e. it is used with (possibly partial) function names.

In the next condition statement, the change of quantifier signals that siblings and children are relations. The fact that children is also a set name causes no difficulty. The persons declaration ends with a period.

One may make other statements about persons; however, only the declaration itself is to be output in response to the question: "what is a person?" Of course a declaration may be changed and recompiled.

The couple declaration illustrates the inclusion of "records" in LESK, signaled by "consists of". An ARC statement results which says that a couple is uniquely specified by its two "components", malepartner and femalepartner. We have:

inv(malepartner) of malepartner
∩ inv(femalepartner) of femalepartner
⊂ id ∩ (couples × couples)

sex of malepartner of couples = male
sex of femalepartner of couples = female

Sometimes a term can be declared with a single statement, termed an "assertion". Thus the function father is given by:

father = malepartner of parents

Similarly, the relation siblings is given by:

siblings = children of father
∩ children of mother ∩ not id

We finally exhibit several relation declarations, each denoted by a simple English phrase: is related to, is an ancestor of, and is married to:

relatedto = ancestors ∪ inv(ancestors)
∪ inv(ancestors) of ancestors
ancestors = father ∪ mother
∪ ancestors of ancestors
marriedto = inv(marriedto)
marriedto = legalspouse
(maleperson × femaleperson) ∩ marriedto
⊂ malepartner of inv(femalepartner)

The phrase the legal spouse of signals that legalspouse is a partial function on the set of adults. We recall that R is a partial function iff (R of inv(R)) ⊂ id, hence the ARC statement is:

legalspouse of (inv(legalspouse)
∩(adults × adults)) ⊂ id

## 5. Implementation of a LESK System

A compiler is partially written which will translate the LESK statement types in the appendix into ARC and/or DARC. We have also developed a theorem prover for ARC which operates subject to specified time and depth limits. These programs are written in DECsystem-10 PROLOG (Pereira, Pereira and Warren, 78), a particularly suitable and enjoyable language. Each program runs in 20K. At present, the theorem prover reads the output of the compiler into its associative database and then can accept questions interactively. An important step will be to couple the compiler with the theorem prover so that semantic errors can be deleted. (These are typically set expressions which turn out to be null, or relation arguments not contained in the sets previously declared to be the source or target of the relation.)

Some additional extensions are:

1. Augmenting the theorem prover so that it uses a "modal" strategy: for all questions q, it first would attempt to prove "q is true" in less than n seconds (e.g. n = 10), failing which, "q is false" in n seconds, failing which it would reply "don't know". We have not yet developed the details of this technique, but believe it to be essential.

2. Adding the special-purpose procedures needed to answer the many question types which are not ARC theorems, e.g. "what is the declaration of person?" or "what is the relation between persons and couples?".

3. Adding an ARC-to-LESK translator for English-like output. This would also allow the variety of synonymous LESK input forms to be reduced to a "canonical" form.

4. Coupling the system to an existing relational database system. (The reasons for wanting to do this are discussed in the articles by Minker (78), Chang (78) and Kellog, Klahr and Travis (78).)

## 6. Related Research

The research described here can be considered as a practical application of a number of developments in AI research; for example, it has drawn ideas from hierarchical knowledge representation systems (e.g. KRL, Bobrow and Winograd, 77); FRL, Roberts and Goldstein, 77); from non-resolution-based deduction systems (Bledsoe, 77) and certainly from PROLOG itself (Kowalski 74; Deliyanni and Kowalski, 79). ARC may be viewed as a synthesis of predicate calculus and semantic network representations; such a synthesis has been proposed by Schubert (76) and Hendrix (75), among others. It is also related to a number of more general attempts to express NL in predicate calculus (e.g. Evens and Smith, 78).

This work may also be considered as a confluence of artificial intelligence and database research (Wong and Mylopoulos, 77) although we see it in a much broader perspective. A majority of the directly relevant research in this area is reported in a single volume: Gallaire and Minker, 1978. There, for example, Reiter discusses the logical problems of coupling a generic knowledge base to an (instantial) database. Several existing systems couple a theorem prover to a relational database: Minker; Chang; Kellog, Klahr and Travis (all ibid). These systems typically emphasize the design of a query language more human-engineered than PC, without considering the same problem in the initial step of a database system: the linguistic and logical specification of the conceptual structure of the database. LESK and ARC are addressed to this more fundamental problem, and yield as a byproduct, a useful query language as well.

If one consults the database literature per se (typified by the ACM Transactions on Database Systems; the International Conferences on Very Large Data Bases; the Proceedings of the SIGMOD Conferences, or Nijssen (76, 77)) little concern is expressed as yet for
1. user-engineered conceptual specification languages (like LESK);
2. coupling extensional databases (e.g. a typical relational database) to (intensional) gkbs;
3. providing practical yet precise semantic specifications.
The common assumption is that to design a database, the problem reduces to specifying a "3-level schema", as in the ANSI/SPARC approach (ANSI/X3/SPARC, 75); we believe there exists a prior step of "conceptual acquisition". Some researchers who share this view are: Bubenko (79), Kent(78), Nijssen (76, 77), Sundgren (79).

A related project which should be compared to LESK is TAXIS (Mylopoulos, Bernstein and Wong, 79). TAXIS is an "AI-inspired" language intended for database systems implementors, and as such is not suitable for use by non-computer professionals. The combination of LESK and TAXIS however is potentially a double-edged tool for implementing systems correctly. Given a conceptual design specified in LESK, a systems designer using TAXIS (possibly with some automatic translation) would have an easier task.

The AI literature, as evidenced by IJCAI79, has just begun to reflect attention to some of the problems we consider. (Dahl, Furukawa, Gallaire and Lasserre, Janas, all ibid.)

## 7. Conclusions

1. A language such as LESK which combines ease of use with logical precision is a necessity in generic knowledge acquisition, hence the design of such a language is a subject of much current research. There are several dimensions along which to rank the design: formality; if it is too mathematical, too few people will be able to use it, whereas if it is too much like NL, its semantics will become unclear to the user; level: too high a level means not enough detail is specifiable; on the other hand most current system design languages are mainly concerned with specifying conceptually irrelevant detail; generality: is it better to be applicable to many tasks but to be replaced in some of these by more particularized languages, or to be very good at doing just one job? The design of LESK attempts to fill what is seen as a considerable hole in this "language space". It has been used in a variety of other applications (Skuce, 75-79) and yet has served a purpose in a typical database application. We envision dialects of LESK for various applications, and intend to experiment with some such as basic gk acquisition in scientific subjects for computer-aided learning, and (overdue) clarification of the concepts involved in a university's academic regulations.

2. As a formalism for knowledge representation in a deductive system, ARC has many potential applications in AI apart from its use as an "deep structure" for LESK. We will investigate the connection between ARC and other knowledge representation systems in a forthcoming paper.

3. The database literature has not reported sufficent attention to the concept acquisition phase (which precedes all others) of database design. Though there has been much work on user-engineered query lanaguages, and system design languages which assume the concepts have already been made clear to the implementors, this latter aspect has not seen a user-engineered tool adequate developed. LESK serves, amongst other uses, to bridge the communication gap between user and implementor which until now has relied excessively upon natural language, with its inherent uncertainties.

## References


Abrial, J. (74) Data semantics. in: Klimbie, K. and Koffeman, K. (eds) Data Base Management. North-Holland, Amsterdam.

ANSI/X3/SPARC: (75) Interim report, Study Group on Data Base Management Systems, FDT Bulletin of ACM, v. 7, no. 2, 1975.

Bledsoe, W. (77) Non-resolution theorem proving. AI Journal, v. 9, pp. 1-35.

Bobrow, D. and Winograd, T. (77). An overview of KRL-0, a knowledge representation language. Cognitive Science, v. 1, no. 1.

Bubenko jr., J. (79) On the role of 'understanding models' in conceptual schema design. (to appear)

Chang, C. (78) DEDUCE2: Further investigations on deduction in relational database systems. in: Gallaire and Minker (78).

Dahl, V. (79) Quantification in a

three-valued logic for natural language question answering. in: IJCAI79.
Deliyanni, A. and Kowalski, R. (79) Logic and Semantic Networks. CACM, v. 22, no. 3, pp. 184-192.
Evens, M. and Smith, R. (78) A lexicon for a computer question-answering system. Amer. J. Comp. Ling., microfiche 83.
Furukawa, K. (79) Relational strategies for processing universally quantified queries to large data bases. in: IJCAI79.
Gallaire, H. and Lasserre, C. (79) Controlling knowledge deduction in a declarative approach. in: IJCAI79.
Gallaire, H. and Minker, J. (eds) (78) Logic and Databases. Plenum Press, New York.
Hammer, M. and McLeod, D. (78) The semantic data model: a modelling mechanism for database applications. Proc. 1978 SIGMOD Conf.
IJCAI79 (79) Proc. of the Sixth International Joint Conference on Artificial Intelligence, Tokyo.
Hendrix, G. (75) Extending the utility of semantic networks through partitioning. Proc. IJCAI75.
Janas, J. (79) How to not say "nil"-improving answers to failing queries in data base systems. in: IJCAI79.
Kellogg, C., Klahr, P. and Travis, L. (78) Deductive planning and pathfinding for relational data bases. in: Gallaire and Minker (78).
Kent, W. (78) Data and Reality. North-Holland, Amsterdam.
Kowalski, R. (74) Predicate calculus as a programming language. IFIP Proceedings, pp. pp. 569-574.
Minker, J. (78) Search strategies and selection functions for an inferential relational system. ACM Trans. on Database Systems, v. 3, no. 2.
Mylopoulos, J., Bernstein, P. and Wong, H. (79) A language facility for designing interactive database-intensive applications. Technical Report CSRG-105, Computer Systems Research Group, University of Toronto.
Nijssen, G. (ed) (76) Modelling in data base management systems. Proc. of the IFIP TC-2 Working Conference. North Holland, Amsterdam.
Nijssen, G. (ed) (77) Architecture and models in data base management systems. North Holland, Amsterdam.
Pereira, L., Pereira, F. and Warren, D. (78) User's guide to DECsystem-10 PROLOG, University of Edinburgh, Dept. of Artificial Intelligence.
Reiter, R. (78) On closed world databases. in: Gallaire and Minker (78).
Reiter, R. (78) Deductive question answering on relational databases. ibid.
Roberts, R. and Goldstein, I. (77) The FRL Primer. MIT AI Lab report 408.
Schubert, L. (76) On the expressive adequacy of semantic networks. Artificial Intelligence 7, pp. 163-198.
Skuce, D. (75) An English-like language for qualitative scientific knowledge. Proc. of the Fourth International Conference on Artificial Intelligence, Tbilisi, pp. 593-600.
Skuce, D. (77) Toward communicating qualitative scientific knowledge between scientists and machines. Ph.D. dissertation, Dept. of Electrical Engineering, McGill University.
Skuce, D. (78) Describing programming language concepts in LESK. Proc. of the Second National Conference of the CSCSI, Toronto, pp. 288-295. Skuce, D. (79a) A language for exactly stating qualitative generic knowledge. submitted to: Journal of Computers in Biology and Medicine.
Skuce, D. (79b) An approach to defining and communicating the conceptual structure of data. Tech. Report No. 79-05, Dept. of Computer Science, Univ. of Ottawa.
Smith, J. and Smith, D. C. P. (79) Database Abstractions: Aggregation and Generization. Tech. Report CCA-79-12, Computer Corporation of America.
Sundgren, B. (78) Data base design in theory and practice. Proc. of the Fourth International Conference on Very Large Databases.
Wong, H. and Mylopoulos, J. (77) Two views of data semantics: a survey of data models in Artificial Intelligence and database management. INFOR, v. 15, no. 3.

## Appendix

A fragment of the generic knowledge underlying an example database.

```
a person:
        kinds : living -, dead -;
        kinds (by age): adults (age > 17), children (age <= 17);
        kinds (by sex): male - (sex = male), female - (sex = female);
        the age of - < 115;
        the sex of - is 1 of: male, female;
        - has 1 dwelling;
        - has 1 couple P called parents of -;
        - has 0 or more persons called siblings of -;
        - has 0 or more persons called children of -.


a couple:
        consists of: 1 adult X called the malepartner of -,
                     1 adult Y called the femalepartner of -;
        the sex of X = male;
        the sex of Y = female.


the father of a person X = the malepartner of the parents of X.

a sibling of a person X = a child of the father of X  and
                          a child of the mother of X, but not X.

a sister of a person X = a sibling of X who is a female person.


a person X is related to a person Y iff:
        X is an ancestor of Y or
        Y is an ancestor of X or
        there exists a person Z such that
                Z is an ancestor of X and
                Z is an ancestor of Y.


a person X is an ancestor of a person Y iff:
        X is the father of Y or
        X is the mother of Y or
        there exists a person Z such that
                X is an ancestor of Z, and
                Z is an ancestor of Y.


an adult X is married to an adult Y iff:
        Y is married to X iff
        X = the legal spouse of Y iff
        Y = the legal spouse of X;
        the sex of X ≠ the sex of Y;
        if X is a male person and
           Y is a female person
        then
           there exists a couple C such that
           X is the malepartner of C and
           Y is the femalepartner of C.
```

# THE REPRESENTATION OF AN EVOLVING SYSTEM OF LEGAL CONCEPTS:

## I. Logical Templates

L.T. McCarty
Faculty of Law, SUNY at Buffalo

N.S. Sridharan
Computer Science, Rutgers University

Although our earlier work on the TAXMAN Project (McCarty, 1977) has demonstrated the basic feasibility of applying artificial intelligence techniques to the field of corporate tax law, the original TAXMAN system was seriously deficient as a model of "legal reasoning". More recently (McCarty, Sridharan and Sangster, 1979), we have proposed an alternative model of conceptual structure, and an approach to the process of conceptual change, in an attempt to remedy these deficiencies. In the TAXMAN II system, which is currently under development, we distinguish between two different kinds of legal concepts. Precise statutory rules are represented as logical templates, a term intended to suggest the way in which a "logical" pattern is "matched" to a lower-level factual network during the analysis of a corporate tax case. But the more amorphous concepts of corporate tax law, the concepts typically constructed and reconstructed in the process of a judicial decision, are represented by a prototype and a sequence of deformations of the prototype. The prototype is a relatively concrete descrption selected from the lower-level factual network itself, and the deformations are selected from among the possible mappings of one concrete description into another. We have suggested that these prototype-plus-deformation structures play a crucial role in the process of legal argument, and that they contribute a degree of stability and flexibility to a system of legal concepts that would not exist with the template structures alone (see McCarty, 1980).

In this paper, we present our curent implementation of the logical template structures of TAXMAN II, but with an eye towards the subsequent implementation of the prototypes and the deformations. In order to construct a deformation of a conceptual prototype, it seems, we must first have availale a clear and coherent representation of the prototype to be deformed. The space of possible concepts must be syntacticaly simple but the corporate tax domain itself is semantically rich. These are the constraints, then, on our initial representation. In Section I of this paper, we will describe the ground-level representation of TAXMAN II, and in Section II we will describe the representation of a hierarchy of higher-level concepts. Section III then describes the pattern-matching procedures which operate in this conceptual hierarchy. For a more detailed version of this paper, see McCarty and Sridharan (1980).

### I. The Basic AIMDS Representation.

We have chosen the AIMDS language (Sridharan, 1978) as the basic vehicle for the implementation of TAXMAN II. Based on the Meta-Description System of Srinivasan (1973, 1976), AIMDS provides extended facilities for the representation of states, events, actions and expectation structures. It was developed primarily for application to the Plan Recognition problem (Schmidt, Sridharan and Goodson, 1978; Sridharan and Schmidt, 1978), but we have discovered that its features are quite general and quite easily adaptable to the needs of the TAXMAN Project as well. AIMDS permits the user to construct a system of templates to describe certain named classes of objects, and a system of relations to express the allowable relationships between these objects. The user can then generate instances of these templates and their associated relations in a particular context, and yet constrain this process of instantiation by a set of consistency conditions written out in a version of many-sorted first-order logic. AIMDS provides a uniform procedure for the instantiation process, called MAKE, and a uniform procedure, called FIND, to retrieve a set of instances from a given context using a partial specification of the network of adjacent relations. Thus, in the spirit of several contemporary frame-based languages, AIMDS resembles at its lowest level a language for processing semantic networks, but it imposes on these networks a higher-level "structure", an organization of knowledge into manageable conceptual "chunks", by means of its interlocking system of template definitions.

For a simple example, consider the OWNership relation in the TAXMAN II system. We construct the template OWN by using the template definition function TDN:

```
(TDN: ((OWN REL)
       ((OWNER FN) ACTOR)
       ((OWNED FN) PROPERTY)))
```

This means that an instance of OWN, which is a template of type REL, must have an "owner" which is an instance of ACTOR, and an "owned" which is an instance of PROPERTY. The flag FN on the relations "owner" and "owned" indicates that a particular instance of OWN can have at most one ACTOR and one PROPERTY standing in these relationships, i.e., the relations are "functional". Suppose we also define a template for STOCK (which is a subset of SECURITY) and a template for SHARE (which is a subset of PROPERTY):

```
(TDN: ((STOCK OBJ)
       ((ISSUEDBY FN) CORPORATION
                      (INVERSE ISSUEROF L))
       ((NSHARES FN) NUMBER)
       ((PARVALUE FN) NUMBER)
       ((VOTING FN) YESNO)
       ((COMMON FN) YESNO
                    (COMPLEMENT PREFERRED FN))))
```

```
(TDN: ((SHARE OBJ)
       ((SHAREOF FN) SECURITY
                     (INVERSE SHARES L))
       ((FRACTION FN) NUMBER)
       ((QUANTITY FN) NUMBER)
       ((VALUE FN) NUMBER)))
```

Then we can write out a fragment of the "security interest" space for a corporation named "NewJersey" by means of the following code:

```
      (MAKE (STOCK (ISSUEDBY NEWJERSEY)
                   (NSHARES &(NUM 294271.0))
                   (PARVALUE &(NUM 29427100.0))
                   (VOTING YES)
                   (COMMON YES)))
      (MAKE (STOCK (ISSUEDBY NEWJERSEY)
                   (NSHARES &(NUM 160686.0))
                   (PARVALUE &(NUM 16068600.0))
                   (NOT VOTING YES)
                   (PREFERRED YES)))
      (MAKE (OWN (OWNER &(MAKE (PERSON PHELLIS)))
                 (OWNED
                  &(MAKE
                    (SHARE
                     (SHAREOF
                      &(FIND (THE STOCK
                                  (COMMON YES)
                                  (ISSUEDBY
                                   NEWJERSEY))))
                     (QUANTITY &(NUM 250.0)))))))
```

This code would first create an instance of "NewJersey common stock" (call it STOCK-1, say), an instance of "NewJersey preferred stock" (call it STOCK-2), and an instance of PERSON named Phellis. It would then retrieve the common stock, STOCK-1, and create an instance of SHARE (call it SHARE-1) which would be asserted to be a "shareof" STOCK-1: (SHARE-1 SHARE-SHAREOF-SECURITY STOCK-1). Finally, it would create an instance of OWN (call it OWN-1), and add to the network the relations (OWN-1 OWN-OWNER-ACTOR PHELLIS) and (OWN-1 OWN-OWNED-PROPERTY SHARE-1). It should be noted here that this network of relations is explicitly created and stored only as a default mechanism in AIMDS. If preferred, the user can write his or her own functions to ADD a relation, to REMOVE a relation, to CHECK the truth value of a relation, and to FETCH all instances which satisfy a relation in a given context.

As described so far, the AIMDS system bears a strong resemblance to several other high-level languages in the AI literature. For example, the "templates" of AIMDS are quite similar to the "units" of KRL (Bobrow and Winograd, 1977; see also Martin, Friedland, King and Stefik, 1977), and the "relations" of AIMDS correspond to the "slots" of these other frame-based systems. One important feature of AIMDS, however, which does not appear prominently in these other systems, is the facility for the partial evaluation of a logical expression. AIMDS provides a general subsystem, called CHECKER, which accepts an expression in a slightly restricted version of first-order logic, evaluates the expression with respect to a given network, and then returns as its result the sub-expression and the set of variable bindings which produced the value of true, false, or unknown, respectively. The expression returned by CHECKER is called a residue, and it plays a major role in the operation of AIMDS. In basic AIMDS, the CHECKER subsystem is used primarily for the evaluation of the consistency conditions, the set of logical expressions which are attached, or "anchored", to the relations of a template. Consistency conditions provide a mechanism for continually monitoring the data base, in some cases simply reporting back an inconsistent instantiation, and in some cases actually updating the network automatically. In TAXMAN II, the consistency conditions are used for several purposes, but the major use of the CHECKER subsystem there is in the pattern-matching procedures, which will discussed in Section III below.

There are other ways, too, in which the ground-level TAXMAN II system has extended basic AIMDS, although the new features we have added are generally well-known in the AI literature. (1.) In addition to the separation of networks into distinct CONTEXTS, which is a feature of AIMDS, the TAXMAN II system provides an additional separation of networks into distinct STATES within each CONTEXT. The states are arranged in a binary-branching tree, so that each assertion in the network in a particular state is visible in each successor state unless it has been explicitly modified in an intervening state. We use this facility primarily to model an historical sequence of states and events, such as the facts of a case, and we use the binary-branching capability to model certain hypothetical variations of the facts of a case. (2.) As our earlier illustration suggests, the AIMDS templates have been organized into several hierarchies of classes and subclasses in the TAXMAN II system. For example: an ACTOR can be a PERSON or a CORPORATION; a PROPERTY can be a PHYSOBJ or a CASH or a SHARE; a SECURITY can be a STOCK or a BOND. We call these hierarchies view hierarchies, because at the level of the instantiated networks they express the various ways of "viewing" a given instance (see Bobrow and Winograd, 1977). Whenever a relatively "specialized view" of an instance is created, the TAXMAN II system automatically creates its more "generalized views", and then propagates upwards a designated set of relations, the so-called "inheritable properties", using the modular network access functions. (3.) Another feature of the TAXMAN II system is its facility for representing meta-templates. The basic idea is to permit each named template (which represents a class of objects) to be itself an instance of a higher-order template (which represents the meta-class). We use this facility in our procedures for matching conceptual hierarcies, as discussed in Section III below, but we also use it in our initial description of the corporate tax domain, where STOCK-1 can be treated as a class of SHARE instances, as well as an instance itself of the higher-order meta-class SECURITY. For a similar discussion of meta-classes, see Levesque and Mylopoulos (1979).

## II. The Conceptual Hierarchies.

The major extension of AIMDS in the TAXMAN II system is the construction of an explicit conceptual hierarchy. Although we have seen already how the AIMDS templates can be arranged in a "view hierarchy", we will see in this section how a template can be defined in terms of a conceptual expansion in a space of descriptions, thus adding another level of organization to the representation. Section II-A describes the syntax and semantics of these new descriptions, and Section II-B shows how they can be arranged in a full abstraction/expansion hierarchy.

### A. Descriptions: DDNs and PDNs

Let us first examine the TAXMAN II descriptions, or, as they will frequently be called, the DDNs. A description has the form: ([CONTEXT cxtvar] [STATE stvar] <template-list> <constraint-list> <bindings-list>), where

```
<template-list>
        ::= ((tname tvar (rname svar) ..) ..)

and tvar ::= <variable-name>
    svar ::= <variable-name>
          or (tname <variable-name>)

<constraint-list>
        ::= (.. <any-logical-expression> ..
             .. <any-arithmetical-constraint>..)

<bindings-list> ::= <any-AIMDS-bindings-list>
```

In this expression, cxtvar and stvar are "context variables" and "state variables," respectively, and they are optional, as indicated by the square brackets. Similarly, tvar is a "template variable" and svar is a "relation variable" or "slot variable". Notice that each item in the <template-list> resembles the input to a MAKE or a FIND, as described in Section I, except that the names of specific instances have been replaced by the names of variables. Also, the unlimited embedding of descriptions which is permitted in the MAKE/FIND syntax is disallowed in the DDN. The "slot variables" here can be constrained, at most, by an expression of the form (tname <variable-name>), where "tname" would generally be a template somewhat further down in the VIEW hierarchy. As an alternative, however, the variables in a DDN can be constrained by the overall structure of the <template-list> itself, and, most significantly, by the <constraint-list>. It turns out that this "flat" description syntax has certain advantages for the specification of a hierarchical matching procedure, as we will indicate later.

Here are some examples. Consider first the "securityholding" description, which we might write out in English as "the OWNership by an ACTOR of a SHARE of a SECURITY which is issued by a CORPORATION":

```
(DDN: (((OWN O1 (OWNER A1) (OWNED (SHARE P1)))
        (SECURITY S1 (SHARES P1) (ISSUEDBY C1)))
       NIL
       NIL))
```

In this example, the relation "shares" is the inverse of the relation "shareof", and we know that A1 must be an ACTOR and C1 a CORPORATION because of the original definitions of the templates OWN and SECURITY, respectively. Note also that we have used the embedded template (SHARE P1) here to indicate that the PROPERTY is restricted to the template SHARE. The <constraint-list> and the <bindings-list> are both NIL in this first example, but consider now how we can specialize the "securityholding" description to represent what might be called "NewJersey-voting- common-stockholding":

```
(DDN: (((OWN O1 (OWNER A1) (OWNED (SHARE P1)))
        (STOCK S1 (SHARES P1) (ISSUEDBY C1)))
       ((S1 STOCK-COMMON-YESNO YES)
        (S1 STOCK-VOTING-YESNO YES))
       ((C1 (NEWJERSEY)))))
```

In this example, the SECURITY template has been specialized to the STOCK template, the STOCK template has been further constrained by the "voting" and "common" relations, and the CORPORATION, C1, has been bound to the instance NEWJERSEY.

In addition to the use of descriptions, or DDNs, in the TAXMAN II system, we will make frequent use of productions, or PDNs. The PDN has a structure very similar to that of the DDN, except that it contains two <template-lists> instead of one, each of which can be localized to a particular STATE:

```
<template-list-1>
      ::= ([STATE stvar]
           (tname tvar (rname svar) ... ) ... )

<template-list-2>
      ::= ([STATE stvar]
           (tname tvar (rname svar) ... ) ... )
```

As the syntax suggests, these PDNs would typically be used to represent a transformation from one state description to another. For example, the transfer of OWNership from one ACTOR to another ACTOR could be represented as follows:

```
(PDN: ((STATE T1 (OWN O1 (OWNER A1) (OWNED P1)))
       (STATE T2 (OWN O1 (OWNER A2) (OWNED P1)))
       NIL
       NIL))
```

For another example, consider the splitting of a SHARE of STOCK into two equivalent SHAREs (and see McCarty, 1977, for a discussion of the need for a device of this sort):

```
(PDN: ((STATE T1
             (OWN O1 (OWNER A1) (OWNED P1))
             (SHARE P1 (SHAREOF S1)
                       (QUANTITY NTOTAL)))
       (STATE T2
             (OWN O1 (OWNER A1) (OWNED P1))
```

```
            (SHARE P1 (SHAREOF S1)
                      (QUANTITY N1))
            (OWN O2 (OWNER A1) (OWNED P2))
            (SHARE P2 (SHAREOF S1)
                      (QUANTITY N2)))
      ((EQUAL NTOTAL (*PLUS N1 N2)))
      NIL))
```

The <constraint-list> in this PDN provides our first example of an "arithmetical constraint": it says that the total quantity of SHAREs owned by A1 must be the same both before and after the split. In our current implementation, the arithmetical constraints are limited to expressions of equality between one variable and an arithmetical LISP function of other variables, as in this example, but we are planning to generalize this syntax in the near future.

### B. Abstraction/Expansion Hierarchies

Now that we have defined the DDNs and the PDNs of the TAXMAN II system, we are ready to link these expressions up to the TDNs of the basic AIMDS representation, in order to construct a full abstraction/expansion hierarchy. For example, consider the DDN which represented the "securityholding" pattern in our original illustration. Suppose we now wanted to define a template of type REL named SECURITYHOLDING, which would represent the relationship between the "security-holder" and the "security-issuer" for any possible SECURITY. We would write:

```
(TDN:
 ((SECURITYHOLDING REL)
  ((HOLDER L) ACTOR (REF: A1))
  ((ISSUER FN) CORPORATION (REF: C1))
  ((STRUCTURE FN)
   &(MAKE
   (DESCRIPTION
    (DLIST
     &(DDN: (((OWN O1 (OWNER A1)
                      (OWNED (SHARE P1)))
               (SECURITY S1 (SHARES P1)
                            (ISSUEDBY C1)))
              NIL
              NIL))))))))
```

Ignoring for the moment the details of this code, let us simply observe that the slots of the TDN have been assigned variable names which correspond to the variable names of the DDN. We refer to the DDN here as the expansion of the SECURITYHOLDING template, and conversely we refer to the SECURITYHOLDING template as the abstraction of the DDN expressing the OWNership of a SECURITY issued by a CORPORATION. Notice that the "securityholding" DDN contains several free variables, O1, A1, P1, S1 and C1, but the SECURITYHOLDING template contains only the variables A1 and C1 in the slots for the "holder" and the "issuer", respectively. This is generally the way abstractions work: an abstraction deletes information which is explicitly represented in the expansion, but it also partially encodes the missing information in the name of the abstraction template itself.

A good example of the power of this kind of structural expansion is the concept of a "B-Reorganization", which was discussed extensively in McCarty (1977). By statute, a B-Reorganization is defined as "the acquisition by one corporation, in exchange solely for all or part of its voting stock ... of stock of another corporation if, immediately after the acquisition, the acquiring corporation has control of such other corporation ..." United States Internal Revenue Code, 368(a)(1)(B). In the TAXMAN II system, we represent the basic structure of this concept as follows:

```
(TDN:
 ((BREORGANIZATION ACT)
  ((ACQUIRINGCORP FN) CORPORATION (REF: C1))
  ((ACQUIREDCORP FN) CORPORATION (REF: C2))
  ((TIME1 FN) TIME (REF: T1))
  ((TIME2 FN) TIME (REF: T2))
  ((STRUCTURE FN)
   &(MAKE
     (DESCRIPTION
      (DLIST
       &(DDN: (((ACQUISITION ACQ
                             (ACQUIRER C1)
                             (ACQUIREDPROP AP)
                             (TRANSPROP TP)
                             (TIME1 T1)
                             (TIME2 T2))
                (CONTROL CON
                         (CONTROLLER C1)
                         (CONTROLLED C2)
                         (TIME T2)))
               ((AP SHARE-SHAREOF-SECURITY S1)
                (S1 SECURITY-STOCKV-STOCK S1)
                (S1 SECURITY-ISSUEDBY-CORPORATION
                    C2)
                (TP SHARE-SHAREOF-SECURITY S2)
                (S2 (SECURITY-STOCKV-STOCK
                     STOCK-VOTING-YESNO)
                    YES)
                (S2 SECURITY-ISSUEDBY-CORPORATION
                    C1))
               NIL))))))))
```

The DDN in this example describes an ACQUISITION by a CORPORATION of CONTROL of another CORPORATION, where the "acquired-property" is related to the "acquired-corporation" in a certain way, and the "transferred-property" is related to the "acquiring-corporation" in a certain way. However, neither ACQUISITION nor CONTROL is a primitive template. Each has a further structural expansion: ACQUISITION expands into a sequence of EXCHANGEs, with constraints, and EXCHANGE expands into a sequence of pairs of TRANSs, with constraints; CONTROL expands into the OWNership of a certain percentage of the STOCK of the controlled CORPORATION.

Given this example, the purpose of classifying the DDN variables into three groups should be apparent. The "template variables" correspond to those templates in the <template-

list> which are potentially subject to further expansion: they act in some ways as if they were existentially quantified, although they often receive multiple instantiations in the network, in which case each instance is analyzed separately in the remainder of the DDN, as if the variable had been universally quantified. The "slot variables" are the variables which can be passed down (or up) with the template variables to (or from) a lower level of the expansion: they are treated as if they were universally quantified. The remaining variables in the <constraint-list> are local to the DDN and purely existential, and they are inaccessible from the lower levels. The basic idea, then, is to restrict severely the channels of communication between the TDNs and the DDNs at different levels of expansion, forcing all information to be carried by the bindings of the slot variables. We conjecture that this is a natural constraint to impose upon a system of conceptual structures, and it has the added virtue of encoding some important control information into the basic semantic representation.

Let us now see how the PDN expressions can be incorporated into an abstraction/expansion hierarchy in much the same way as the DDN expressions. First, the PDN representing the transfer of OWNership (see Section II-A above) could be used for the expansion of a template called DELTAOWN, as follows:

```
(TDN:
 ((DELTAOWN STCH)
  ((OBJECT FN) PROPERTY (REF: P1))
  ((OLDOWNER FN) ACTOR (REF: A1))
  ((NEWOWNER FN) ACTOR (REF: A2))
  ((TIME1 FN) TIME (REF: T1))
  ((TIME2 FN) TIME (REF: T2))
  ((PROCEDURE FN)
   &(MAKE (DESCRIPTION
           (PLIST &(PDN: ((STATE T1
                                 (OWN O1
                                      (OWNER A1)
                                      (OWNED P1)))
                          (STATE T2
                                 (OWN O1
                                      (OWNER A2)
                                      (OWNED P1)))
                          NIL
                          NIL))))))))
```

The expansion here is called a procedural expansion, since the PDN can actually be used as a "procedure" for transforming a network from state T1 to state T2. A template with a procedural expansion can then be conjoined with other templates inside a DDN expression to produce a more complex "procedure". For example, DELTAOWN could be conjoined with SPLITPROP and JOINPROP to produce the procedural expansion of TRANS, at least for those cases in which the PROPERTY transferred is "divisible":

```
(TDN:
 ((TRANS ACT)
  ((AGENT FN) ACTOR (REF: A0))
  ((OBJECT FN) PROPERTY (REF: P1))
  ((OLDOWNER FN) ACTOR (REF: A1))
  ((NEWOWNER FN) ACTOR (REF: A2))
  ((TIME1 FN) TIME (REF: T1))
  ((TIME2 FN) TIME (REF: T2))
  ((PROCEDURE FN)
   &(MAKE
     (DESCRIPTION
      (DLIST
       &(DDN: (((SPLITPROP SP1
                           (OLDOWNER A1)
                           (OLDPROPERTY OP1)
                           (NEWPROPERTY P1)
                           (TIME T1))
                 (DELTAOWN DEL1
                           (OBJECT P1)
                           (OLDOWNER A1)
                           (NEWOWNER A2)
                           (TIME1 T1)
                           (TIME2 T2))
                 (JOINPROP JP1
                           (NEWOWNER A2)
                           (OLDPROPERTY OP2)
                           (NEWPROPERTY P1)
                           (TIME T2)))
                ((P1 PROPERTY-DIVISIBLE-YESNO YES)
                 (T1 TIME-PRECEDES-TIME T2))
                NIL))))))))
```

The basic rule here is that any DDN expression appearing in a procedural expansion must eventually expand down to one or more PDN expressions, so that the procedural expansion can actually be carried out. But the exact ordering of the procedures would normally be specified only by the <constraint-list>, as in this example: (T1 TIME-PRECEDES-TIME T2). Thus many of the planning techniques of Sacerdoti (1977), which involve the progressive tightening of the partial-order constraints, could be accommodated within the TAXMAN II formalism.

## III. The Pattern-Matching Procedures.

Although the preceding discussion has suggested some of the semantic properties of the abstraction/expansion hierarchies, the full "meaning" of these expressions depends on the way they behave with respect to our pattern matching procedures. Basically, the DDNs and PDNs are abstract patterns which can be matched to any number of concrete networks, but when they have been arranged into abstraction/expansion hierarchies the pattern matching procedures must operate across multiple levels.

Let us first consider the matching of a single DDN expression, without any possibility of a further expansion of the <template-list>. The function to call in the TAXMAN II system is (RMATCH <Dname> <Bindings>), which takes as its arguments the name of a DDN expression and an initial AIMDS bindings list. A single-level RMATCH is basically a call to the CHECKER subsystem of AIMDS: the templates in the <template-list> are converted into a conjunction of network relations and combined with the expressions in the <constraint-list> to form the

logical expression which we wish to have evaluated. The result, as described in Section I, is a residue expression telling us whether the match is true, false, or unknown, and why. If the result is unknown, for example, the residue sub-expression and the unknown bindings list will tell us which part of the DDN expression was responsible for the unknown match, an important piece of information to have available. Sometimes the unknown residue can simply be asserted true, as it is in the GMAKE functions of TAXMAN II. GMAKE also takes the name of a DDN expression and an initial bindings list as arguments, and it also calls CHECKER for an initial match result, but it then asserts into the network all the relations which were unknown but uniquely bound in the initial match. RMATCH and GMAKE are thus complementary operations: RMATCH, somewhat like the FIND procedure of the basic AIMDS system, takes a network as given and attempts to construct a set of variable bindings which produce a successful match of the DDN expression; GMAKE, like MAKE, takes the variable bindings as given and attempts to construct a set of assertions within the existing network which satisfy the DDN expression. But it is interesting to note that both functions do their work by manipulating residues, and their control structures are similar.

For the PDN expressions of TAXMAN II, the expressions with two <template-lists>, we use an RMAP function which combines the features of both RMATCH and GMAKE. Suppose we are using the PDNs to express a change of state, such as the "transfer of OWNership" or the "splitting of SHAREs" illustrated in Section II-A above. To carry out this change of state, we would basically do an RMATCH of <template-list-1> and then a GMAKE of <template-list-2>, thus transforming the network from a successful match with the first state description to a successful match with the second state description. Again, the necessary operations can be performed by the manipulation of residues, and this adds considerably to the clarity and uniformity of the implementation.

Let us now investigate how the RMATCH procedure would operate if given a full abstraction/expansion hierarchy, such as the definition of a BREORGANIZATION in Section II-B above. The first step is to try a single-level match of the DDN expression, using RMATCHDDN. If this match succeeds, i.e., if it returns a true residue for every template in the <template-list> subject to the constraints of the <constraint-list>, then the RMATCH procedure terminates with a successful result. If the initial match is false or unknown, however, RMATCHDDN cycles through each template in the <template-list> in an attempt to match the lower-level expansions. Specifically, RMATCHDDN passes down the top-level bindings for each "slot variable" in each template in the <template-list> to a function called RMATCHTDN, which is then responsible for establishing a successful match of the template expansion, if this is at all possible. After an initial analysis of the form of the expansion (is it "structural" or "procedural"?), RMATCHTDN passes the variable bindings down one level further with a second call to RMATCHDDN, and this process continues recursively until the match either succeeds completely, or else terminates from a lack of further expansions. The residue expressions returned from the bottom of the expansion would then be analyzed and combined until the result of the top level by level match could finally be assembled. Although the rules for combining lower-level residues into upper-level match results are complex, the idea is simple: each template in the <template-list> could either be explicitly matched to the network, or implicitly matched by virtue of its lower-level expansions, and the explicit and implicit instances must all be combined according to their truth values, with true residues preferred to unknown residues, and unknown residues preferred to false residues.

In an earlier version of the TAXMAN II system, we attempted to collect all of the combined residue expressions at the top level of the RMATCH, but this approach would be unworkable for a large hierarchy. Our current version stores the residue expressions locally at each TDN and at each DDN of the expansion, and returns to the top level a meta-residue which traces out the path of the RMATCH functions as they proceed recursively down and back through the hierarchy. This approach turns out to be useful, also, for the coordination of several more diverse styles of hierarchical pattern matching. To understand it, we need to understand the notion of a meta-domain for an abstraction/expansion hierarchy. Look again at our code for the SECURITYHOLDING template in Section II-B above. Although it appears here that the DDN has simply been attached to the STRUCTURE slot of the TDN, we actually represent this abstraction/expansion pair by a set of meta-templates and meta-relations: SECURITYHOLDING is an instance of a meta-template called TEMPLATE, which is connected by a "structure" relation to an instance of a meta-template called DESCRIPTION, which is in turn connected by a "dlist" relation to an instance of a meta-template called DDN. Suppose now that this particular instance of a DDN, say DDN017, has been matched to the network to produce a residue expression: we would then create an instance of DDN017,say DDN017-1, to stand for the result of the match. We would store the residue expression attached to this newly created instance. If these match results later justified the creation of a new instance of SECURITYHOLDING (call it SECURITYHOLDING-1), we would connect SECURITYHOLDING-1 to DDN017-1 by instantiating the "structure" and the "dlist" relations in the appropriate context. In this way, we would preserve a record of the dependencies between the DDN instantiations and the TDN instantiations, while distributing the actual residue expressions throughout the network.

Using the meta-residue expressions, it now becomes possible to integrate several other pattern matching strategies into the TAXMAN II system. The RMATCH procedure performs a top-down

goal-directed match of a given abstraction to an existing lower-level network: in short, a "recognition match". The GMAKE procedure (a "generative make") can also be extended to the full abstraction/expansion hierarchy: given a top-level abstraction GMAKE instantiates all the lower-level expansions, if this can be done consistently in the existing network. We can implement this quite easily, it turns out, by first running RMATCH to test for the consistency and the uniqueness of the proposed expansion, and then tracing back down along the meta-residue to generate all of the unknown templates and relations. Perhaps even more significant is the possibility of defining a GMATCH procedure (a "generative match") using meta-residues. In its most general form, GMATCH would do a bottom-up data-driven generation of all the abstract descriptions which could be inferred from an existing lower-level network, a process which seems impractical if the hierarchy is complex and if the inference rules can tolerate partial matches and partial mismatches, as in the TAXMAN II system. However, GMATCH can be defined more reasonably within the framework of an existing set of RMATCH results: GMATCH would then monitor the data base and generate only the updates for the top-level instantiations. Note that the meta-residue expressions record all the dependencies between the DDN instantiations and the TDN instantiations which have resulted from an application of RMATCH. We can implement this version of GMATCH by tracing along the path of the meta-residues. (In this connection, the meta-residue expressions are similar to the "footprints" proposed by Woods, 1978, and the "detlists" of MDS and AIMDS.) In some situations, it may also be useful to have available an RMAKE procedure: this is a goal-directed search for a specific component of a lower-level expansion, and it is related to GMATCH in the same way that GMAKE is related to RMATCH. We should note here, however, that only the RMATCH and the GMAKE procedures have so far been implemented in the TAXMAN II system. GMATCH and RMAKE are in their design stages.

## IV. Future Work.

One of the main goals of the TAXMAN project is to represent a legal concept as a prototype-plus-deformation structure and to analyze the role that this kind of structure plays in the process of legal argument. In this paper, we have examined only the logical template structures of the TAXMAN II system, but we have established the foundation for a subsequent examination of prototypes and deformations. To construct a deformation, we need a clear and coherent representation of the prototype. But we now have a highly structured conceptual space to work with: the DDN space. The DDN expressions can be arranged in a generalization/specialization hierarchy, as we have seen in some earlier examples; they can be stored as AIMDS networks, using variable names in the place of instance names; and they can be transformed by a system of PDN expressions, like any other AIMDS network. We thus have available the basic mechanisms we need to represent the mappings of a conceptual space, as proposed in McCarty (1980) and McCarty, Sridharan and Sangster (1979). We will develop these ideas further in subsequent papers.

Acnowledgement: The National Science Foundation has generously funded our research through Grant SOC-78-11408 from the Law and Social Sciences Program (1978-79) and Grant MCS-79-04091 from the Intelligent System Program (1979-81).

## References

Bobrow, D.G. and Winograd, T., "An Overview of KRL-0, a Knowledge Representa ion Language," 1 Cognitive Science 3-45 (1977).

Levesque, H., and Mylopoulos, J., "A Procedural Semantics for Semantic Networks," in N.V. Findler, ed., Associative Networks 93-120 (Academic Press, 1979).

Martin, N., Friedland, P., King, J., and Stefik, M., "Knowledge Base Management for Experimental Planning in Molecular Genetics," Proceedings, Fifth International Joint Conference on Artificial Intelligence 882-87 (1977).

McCarty, L.T., "Reflections on TAXMAN: An Experiment in Artificial Intelligence and Legal Reasoning," 90 Harvard Law Review 837-93 (1977).

McCarty, L.T., Sridharan, N.S., and Sangster, B.C., "The Implementation of TAXMAN II: An Experiment in Artificial Intelligence and Legal Reasoning," Report LRP-TR-2, Laboratory for Computer Science Research, Rutgers University (1979).

McCarty, L.T., "The TAXMAN Project: Towards a Cognitive Theory of Legal Argument," in B. Niblett, ed., Computer Logic and Legal Language (Cambridge University Press, forthcoming 1980).

McCarty, L.T., and Sridharan, N.S., "The Representation of Conceptual Structures in TAXMAN, Part One: Logical Templates," Report LRP-TR-6, Department of Computer Science, Rutgers University (1980).

Sacerdoti, E.D., A Structure for Plans and Behavior (Elsevier North-Holland, 1977).

Schmidt, C.F., Sridharan, N.S., and Goodson,, J.L., "The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence," 11 Artificial Intelligence 45-83 (1978).

Sridharan, N.S. and Schmidt, C.F., "Knowledge-

Directed Inference in Believer," in D.A. Waterman and F. Hayes-Roth, eds., Pattern-Directed Inference Systems 361-79 (Academic Press, 1978).

Sridharan, N.S., ed., "AIMDS User Manual, Version 2," Report CBM-TR-89, Department of Computer Science, Rutgers University (1978).

Srinivasan, C.V., "The Architecture of Coherent Information Systems: A General Problem-Solving System," Proceedings, Third International Joint Conference on Artificial Intelligence 618-28 (1973); also published in vol. C-25 IEEE Transactions on Computers 390-402 (1976).

Woods, W.A., "Taxonomic Lattice Structures for Situation Recognition," Proceedings, Second Workshop on Theoretical Issues in Natural Language Processing 33-41 (1978).

# PROVIDING AUTOMATIC GRAPHIC DISPLAYS THROUGH DEFAULTS *

Sakunthala Gnanamgari
N.I.Badler, H.L.Morgan, Bonnie L. Webber

Department of Computer and Information Sciences
Moore school of Electrical Engineering
University of Pennsylvania, Philadelphia, Pa. 19104, USA

## Abstract

This paper addresses the problems in providing graphic displays automatically to serve a user naive with respect to computer graphic devices. It identifies the properties of data that affect graphic representation and presents a formalism in which to view them. It also discusses and illustrates the selection of various graphic formats based on the data to be represented, its properties, and graphic device characteristics.

## I. Problem Statement

Broadly speaking, there are three phases of using computers: acquiring, processing and presenting information. As to the first two, many years of research and development have led to the availability of efficient ways of collecting and processing data. However, methods of presenting information are by and large limited to variations of tabular form. Reading a sequence of lines and understanding their import is a tedious job though, reminding people of the old proverb, 'A picture is worth a thousand words.' As a result, efforts are now being directed towards presenting such data graphically. Unfortunately, using graphic devices can be a complex process, requiring days or even weeks of training. Up to now, it has been almost impossible for a naive user to create a graphic display to view information.

Our long range goal is to have an intelligent system helping users in the graphical display of data, performing the task of a graphic artist. Our objective, at present, is to facilitate automatic display of information by providing reasonable defaults for graphical representations and easy user modification of the resulting displays.

The major problem in developing such a system is that there is a gap between the way a user conceives of a graphic display and the way the machine does. For the user, it is a meaningful picture made up of certain particular pieces; for the machine, it is the sequence of operations needed to create such a display. A second problem is that a user will not think to make explicit what s/he does not care about or what s/he believes the system already knows or is able to infer. What is needed is a graphic expert system that, on the one hand, is at an appropriate conceptual level for user to state things that s/he cares about, but, on the other, provides appropriate defaults to take care of everything else.

Research has proven that graphic presentation of information is better than tabular form. Tabular form merely presents raw data without interpretation [Gene Zelazny, 1972], whereas pictorial form conveys the relationship between the data items.

* This research is partially supported by DARPA grant #MDA903-80-C-0093.

To illustrate this contrast between tabular and graphic presentation, consider the following example. Using the Harvest system [Harvest, 1979], a database query system, a naive user can type in

```
WHERE YEAR = 1980 DISPLAY BUDGET
```

and get a formatted output as shown below:

BUDGET FOR 1980

| ITEMS | AMOUNT* |
|---|---|
| 1. SALARIES | 35 |
| 2. TRAVEL | 10 |
| 3. EQUIPMENT | 25 |
| 4. MAINTENANCE | 18 |
| 5. MISCELLANIOUS | 12 |
| TOTAL | 100 |

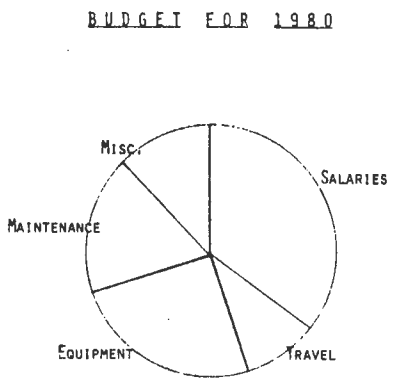
* Thousands of dollars

For tabular form output, systems such as HARVEST can provide default formats. This relieves a naive user of the need to provide detailed format specifications, a burdensome task especially when the user may not care more about the format than it be easy to read.

However, it is not currently possible to request a graphic display in the same easy terms - i.e., to type

```
WHERE YEAR = 1980 DISPLAY BUDGET GRAPHICALLY
```

and get a graphic display as shown here:




No existing system provides the default graphical formats needed to provide such a service.

There are some "high level" software packages commercially available, such as PLOT-10 and DISPLA [ISSCO], that allow an applications programmer to use a graphic device at a programming language level. Interactive systems like Tell-a-Graf [ISSCO] requires users to enter data and specify their preferences completely. But none of these systems can provide default displays for either completely or incompletely specified choices. What is needed is, highly automated graphics systems to meet the needs of naive users who either do not want to specify any preferences about the graphic display or give incomplete specifications.

This paper discusses appropriate defaults for those aspects of a display the user has failed to specify and how those defaults depend on three factors: the data to be displayed, the device on which it is to be displayed and the users it is displayed for. Two different types of defaults are considered: defaults affecting the choice of graph through which to display the data and defaults affecting the choice of "attributes" for that graph, such as color, size, orientation, order and other factors. These defaults are used to provide a naive user with the ability to see his or her numeric data (which would otherwise be presented as a table of numbers) in the form of a pie chart, bar graph or trends graph.

## II. Definitions

Before introducing the system and basic assumptions for the system, we shall define the concepts we will be using:

1. CONTINUITY: a boolean value that represents whether or not the members of an ordered set represent an interval of a continuum with respect to the given ordering. Example: A set of days, {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday} could be defined to represent a WEEK, an interval of time, and have the property continuity, while {Sunday, Tuesday, Saturday} may not, and {Sunday, Tuesday, Friday, Wednesday, Monday, Saturday, Thursday} may not.

2. TOTALITY: is a boolean value that represents whether or not the members of a set represent ALL the component parts of an object or an abstract concept. Example: the set of items {Salaries, Travel, Equipment, Maintenance, Miscellaneous} could be defined to represent the parts of which BUDGET is composed,and have the property totality. The subset {Salaries, Travel, Maintenance} would not have totality.

3. CARDINALITY: is the number of elements in a set. Example: the cardinality of range the set of days is 7.

4. MULTIPLICITY: is the number of values assigned to each element in a domain set by a mapping. Example: the mapping "square root" from real numbers into complex numbers has the multiplicity of 2.

5. UNITS: is the set of labels specifying the unit of measurement associated with each numerical value. Example: Thousands of dollars, Hundreds of tons, etc.

One of the factors upon which effective automatic data display depends comprises particular characteristics of the data itself. By abstracting out these characteristics, one can form a well defined bijection mapping that can help one to understand the complex phenomenon of data and its manipulations.

Let this abstract form of data be represented by the word title, a mapping from the domain set of labels into the range set of quantities. That is,

$$\text{TABLE: } \{l_1, l_2, \ldots, l_m\} \longrightarrow \{(q_{11}, q_{12}, \ldots, q_{1n}), (q_{21}, q_{22}, \ldots, q_{2n}), \ldots, (q_{m1}, q_{m2}, \ldots, q_{mn})\}$$

where, for every i=1 to m, $l_i$ is the ith element in the domain set and for every i=1 to m and j=1 to n, $q_{ij}$ is the jth component of the ith tuple in the range set. Each column also has an entity called units and another ENTITY called column-label.

In other words, the data in the range set is a matrix of size m rows and n columns.

The cardinality of row-labels and multiplicity of the mapping can be derived from input data. However, two additional properties of this mapping, that are necessary to select a display format are not directly derivable from the input data itself. These are:
(i) whether elements of either row-labels or column-labels form component parts of some whole with respect to the quantities represented by each member of the column-labels and row-labels respectively: that is, whether either set has totality.

(ii) whether elements of either row-labels or column-labels denote to a continuum with respect to the quantities represented by each member of the column-labels and row-labels respectively: that is, whether either set has continuity. For example,

NET INCOME PER SHARE

| | COMPANY-1 | COMPANY-2 |
|---|---|---|
| 1972 | 0.86 | 0.60 |
| 1973 | 1.01 | 0.90 |
| 1974 | 1.22 | 1.15 |
| 1975 | 1.35 | 1.45 |
| 1976 | 1.60 | 1.80 |
| 1977 | 1.93 | 2.24 |
| 1978 | 2.44 | 1.90 |
| 1979 | 2.70 | 2.01 |

In this example, the row-labels are 1972, 1973, 1974, 1975, 1976, 1977, 1978 and 1979 and the column-labels are COMPANY-1 and COMPANY-2. The continuity of row-labels could be true or false with respect to each column-label. If the comaparison of incomes for two companies over the period of time is prefered, then the continuity of row-labels would be {true, true} with respect to each of the column-labels. If an absolute comparison of incomes is prefered then the continuity of row-labels would be {false,false}. The totality of row-labels could be true or false. If a relative comparison of each year's income with respect to the total income of each company is prefered, then the totality of row-labels would be {true,true}; otherwise, it would be {false,false}. Similarly, the continuity and totality could be defined for column-labels. The cardinality of row-labels is 8. The multiplicity of the mapping is 2. The units are dollars for each column-label.

## III. Examples

Having defined the concepts that we will be using, to demonstrate how the above mentioned ideas can be used to provide a graphic display, consider the BUDGET FOR 1980 example given earlier. Here the mapping is BUDGET FOR 1980, the set of row-labels is {Salaries, Travel, Equipment, Maintenance, Miscellaneous}, the range set of quantities is {(35), (10), (25), (18), (12)}, the set of column-labels is {Amount} and the set of units is {Thousands of dollars}. Let the totality and continuity of row-labels be {True} and {False} respectively. Given this information and no preferences on the user's part, the system's task is to observe the data and its characteristics, decide what type of graphic format is both suitable and feasible with respect to the graphic device that is available, decide its attributes and then display the picture. (Although it should also allow the user to modify the resulting display, this aspect of the user interface will not be discussed.) For this example, the system selects a pie chart representation to express the totality of the row-labels. This pie chart representation is an appropriate choice as confirmed in the literature:

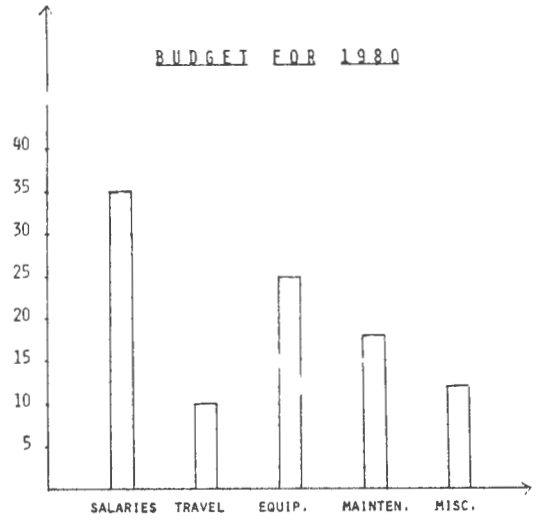
"Because a circle gives such a clear impression of being a total, a pie chart is ideally suited for the one purpose it serves - showing the relative sizes of the components of some whole." - [Zelazny, 1972]

"...the separation of a whole amount in terms of its component quantities. In the graphic figure, a circular form can be used to represent a whole amount, and can be divided into segments which represent proportional quantities, or percentages, of the whole." - [Bowman, 1968].

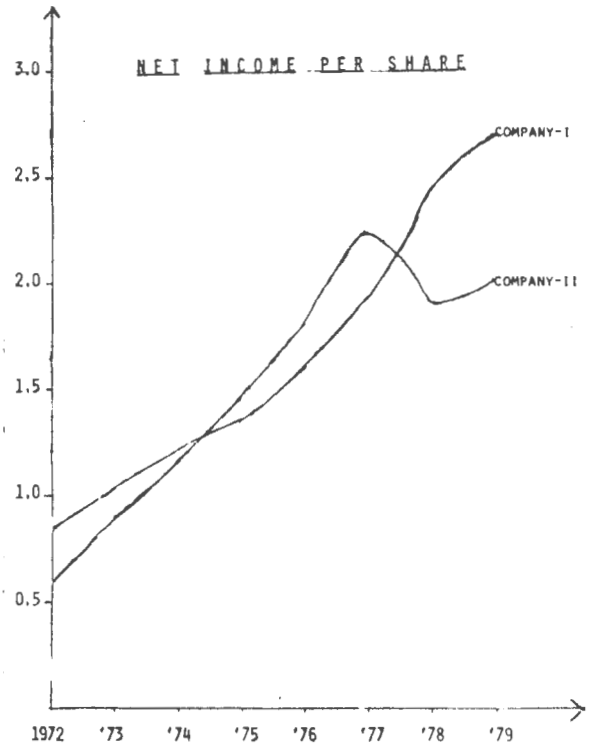
As we noted above, the user has not stated any preferences regarding the display. This being the case, the choice of whether or not to color

the different segments of the pie (and if so, what colors) is left as another set of defaults. These choices/defaults depend partially on device capabilities but also on whether colors would be an effective way of communicating information to the user. For a device such as the printed page, the choice of colors is black and white.

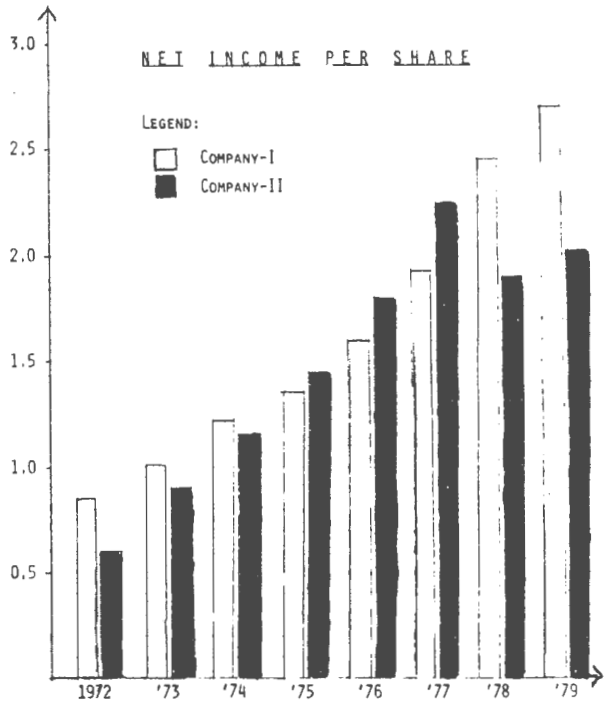
In this example, suppose the totality is {False}, the system whould have opted for a bar chart. The reasons for this option are: (i) continuity being false, a line graph is not selected, (ii) totality being false, a pie representation is not selected, and (iii) "In a graphic figure, quantity can be shown in comparative relation to other quantities, through the extension of abstract parallel bar forms." - Bowman [1968]. The resulting figure is shown below:



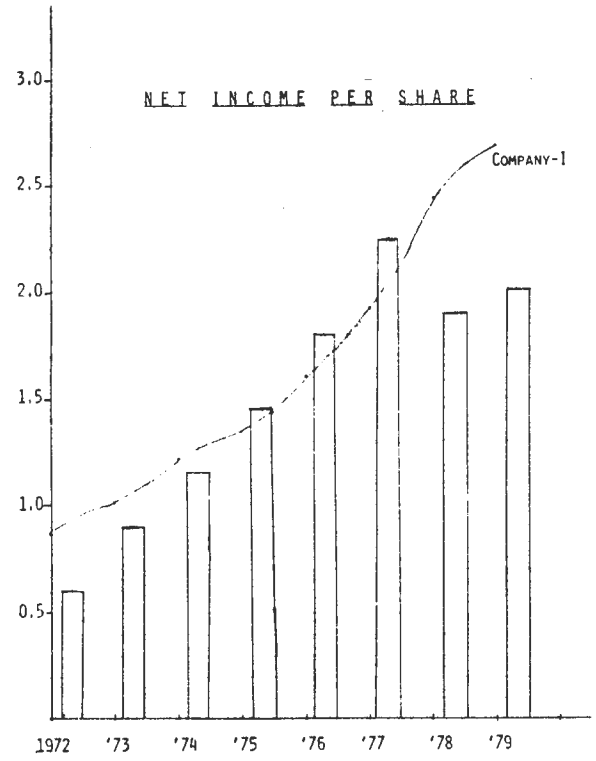

As another example consider the mapping INCOME PER SHARE. We will look at five cases. case 1. If the continuity of row-labels is {true, true}, the totality of row-labels is {false, false} and the units is {dollars, dollars} the graphic format selected would be a LINE graph. That is,




Case 2. If the continuity of row-labels is {false, false}, the selected graphic format would look

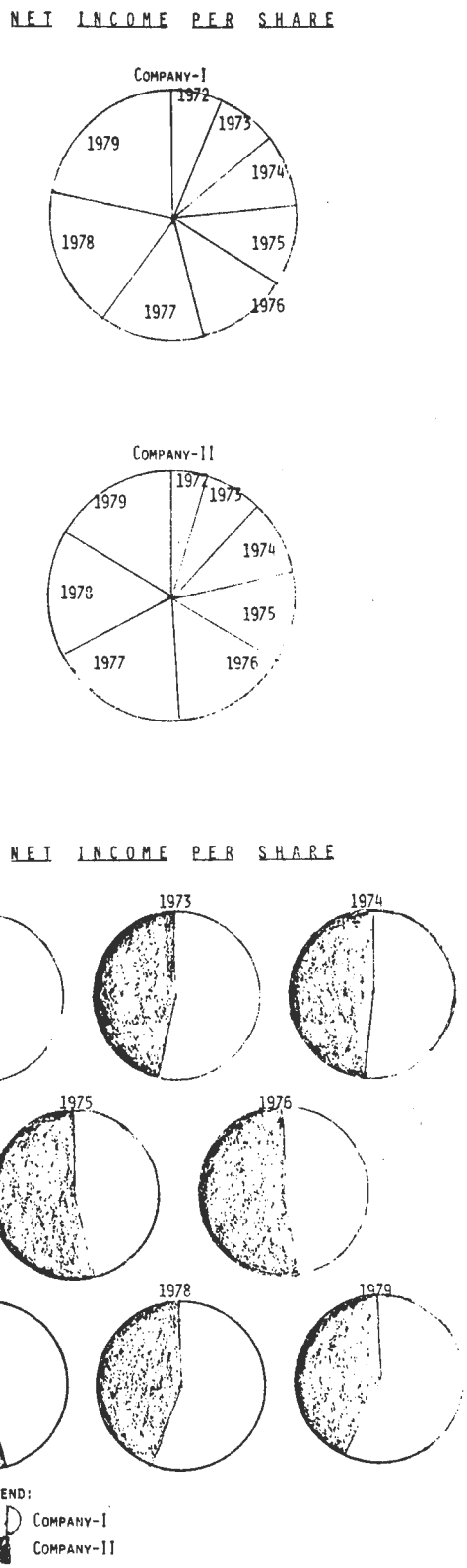


Case 3. If the continuity of row-labels is {true, false}, the graphic format selected would be




Case 4. If the continuity and totality of row-labels are {false,false} and {true, true} respectively, then the data would be presented in the form on the top right.

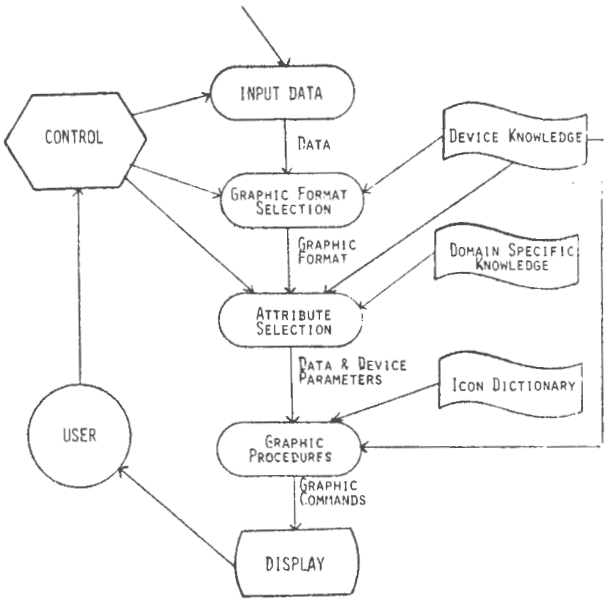
Case 5. If the continuity and totality of row-labels are {false, false} and {false, false}; and the totality of column-labels is {true, true, true, true, true, true, true, true}, then the graphic format on the bottom right represents the input data.



## IV. System Overview

The overview of the proposed system currently under development is given in the following figure.

OVERVIEW OF THE PROPOSED SYSTEM




We are making the following three assumptions with respect to this system design:

(i) DATA is expected from an existing database. The system expects a table of information which has both row-labels and column-labels. Either of these sets may be tagged with the properties of continuity and/or totality. These two properties of the mapping are expected as input to the system along with the data mapping and information on units of measurement for the quantities in the range set.

(ii) DEVICE is expected to have a set of routines for drawing and erasing points, lines and characters, and for setting colors or grey values.

(iii) USER is expected to be able to type in the request for a graphic display.

The information from a database enters the system at the node INPUT DATA. The data is passed to the next node FORMAT SELECTION.

Depending on the characteristics of input data such as multiplicity, cardinality, units, continuity and totality; and of graphic device such as device type, spatial and intensity or color resolution; a default graphic format (such as a pie chart) will be selected to display information. These rules of selecting a particular display format are defined after consulting Bertin [1973], Bowman [1968] and Gene Zelazny [1972 and 1980] and studying various graphic representations.

Once the appropriate graphic format has been selected, the format and the information to be displayed are passed to the next node, the ATTRIBUTE SELECTION. This state consults the device knowledge and domain specific knowledge to determine the attributes of the display such as color and icons. The output of this state consists of data and device parameters.

Depending upon these parameters, the next node, GRAPHIC PROCEDURES, generate the graphic commands to a particular device that realizes the display.

DISPLAY is the actual display of information, the final output of the system, in the graphic format.

The graphic display is obtained by simply requesting the system to present tabular information graphically. If the display is not satisfactory to the user, it may be modified. The modifications are provided at three levels: (i) input data could be modified by selecting or grouping the row-labels to be displayed, (ii) the properties such as totality or continuity could be changed thereby changing the format of the display and (iii) attributes of display could be changed.

## V. Summary

In summary, this paper has discussed the system [Gnanamgari, 1980] which we have designed to provide appropriate defaults for those aspects of the presentation of a user's data that s/he either does not care about or assumes the system would "obviously" infer. The underlying structures of input data have been studied and abstracted and relavant properties of data have been recognized. A reasonably large set of graphic formats have been defined for presenting data. Currently we are working on, knowledge representation issues of the system.

## BIBLIOGRAPHY


1. Bertin, J.: "La Graphique et Le Traitement Graphique de L'Information",Flammarion, Paris, 1977.

2. Bowman, W.J.: "Graphic Communication", John Wiley, 1968.

3. DISPLA: a software product of Integrated Software Systems Corporation, San Diego, Ca. 92121.

4. Gnanamgari, S.: "Automatic Generation and Presentation of Graphic Information Displays", Ph.D. thesis, Department of Computer and Information Sciences, University of Pennsylvania, 1980 (forthcoming).

5. HARVEST Reference Mannual, International Data Base Systems, Philadelphia, Pennsylvania, 1979.

6. TELLAGRAF: a software product of Integrated Software Systems Corporation, San Diego, Ca. 92121.

7. Zelazny, G.: "Choosing and Using Charts", copyright 1972, Gene Zelazny, McKinsey and Company Inc., New York.

8. Zelazny, G.: Director, Visual Communications, McKinsey and Company Inc., New York, Personal Communication, 1980.

# USING COMPUTER PERCEPTION FOR GRAPHICAL TYPE CHECKING*

Nadia Magnenat-Thalmann
Département des Méthodes
Quantitatives
Ecole des Hautes Etudes
Commerciales
Montréal, Canada

Daniel Thalmann
Département d'Informatique et
de Recherche Opérationnelle
Université de Montréal
Canada

## ABSTRACT

The concept of graphical type is a basis of structured and reliable graphical processing. However, it requires the way of checking the compatibility and legality of operations. This paper presents methods of graphical type checking using computer perception. Algorithms have been implemented for MIRA-2D, a graphical PASCAL extension based on graphical types.

## 1. INTRODUCTION

A data type, as it is defined by Wirth [1], determines the set of values to which a constant belongs, or which may be assumed by a variable or an expression, or which may be generated by an operator or a function. The concept of data type is very fundamental, but it requires the way of checking the compatibility and legality of operations. For example, the assignment of a real value to a logical variable has no meaning. Such an error may be detected without executing the program. However in some cases, the compiler cannot detect errors. For example, in a language such as PASCAL which admits subrange types, the result of a calculation may be out of range and it may be only found during the execution. In this case, the problem may be solved in incorporating tests in the object code produced by the compiler or in the runtime library.

Because the concept of data type is very important and because we are concerned with structured computer graphics, we have designed and implemented abstract graphical data types as an extension of the PASCAL language. This extension gives the user a way of defining and using specific graphical types, which can be used as other PASCAL types. For example, the programmer may define and use variables of type triangle, square, circle and so on. Type checking may be perform in most cases at the compile time. For example, we know that a rotation does not alter the type of a variable. But, the language gives the user the possibility of defining image transformations which may change the type of a variable (e.g. a shear does not preserve a circle!). A user may also enter interactively a figure and there is a union operation which allows the user to build a new figure from two existing figures. In these different cases, type checking is very difficult to perform.

It is necessary to recognize the structure of the figure, but at the runtime all figures are implemented as linked lists and it is not possible to have access to the original structure. The only way of checking if the figure has the good type is based on pattern recognition. The process of pattern recognition is dependent on the complexity of the graphical type. In a first step, we create a figure which is a model for the data type that has to be checked. This figure is then translated in such a way that its center is the same as the center of the figure to recognize. Afterwards, the size is adjusted, and two typical vectors of both figures are computed, then the model is rotated until the typical vectors match. At this time, a characteristic function is computed for both figures and it is decided if the figure is analog to the model. Different types of characteristic functions may be choosen and the choices will be discussed. In the case of more complex figures,

* This work was supported by Natural Sciences and Engineering Research Council Canada.

it may be necessary to decompose each figure into simpler figures and apply topological analysis. In some cases, type checking is not possible because the type is too general.

## 2. THE CONCEPT OF GRAPHICAL TYPE CHECKING

### 2.1 The concept of type checking

A data must possess a type; this concept is one basis of structured programming as stated by Hoare [ 2 ]. Let us have an example: we would like to write a function which calculates the factorial of a number. Although it is not the most efficient definition, the factorial may be written in PASCAL as:

```
function fact (n:integer): integer;
begin
if n=0 then  fact :=  1
       else  fact :=  n*fact(n-1)
end;
```

It is clear that such a function will have some trouble when it is invoked as:

```
y := fact(-10)  or  y := fact(10000)
```

A better version of this function will be:

```
const  max = 10;
type   subrange = 0..max; positive=0..maxint;
   function  fact(n:subrange):positive;
   begin
   if  n=0  then  fact := 1
            else  fact := n* fact(n-1)
   end;
```

A statement as y := fact(-10) will produce a diagnosis at compile time, because it is very easy to find that -10 is not included in the subrange 0..10. But the two following statements may be only checked at the runtime:

```
read (val);
y := fact (val);
```

In fact, if the read value is -10, a runtime error has to be detected.

### 2.2 The concept of graphical type

We are concerned with structured computer graphics [ 3 ]. That is the reason why we have designed and implemented MIRA 2D [ 4 ], a graphical PASCAL extension. This extension is based on abstract graphical data types [ 5 ]. Such graphical types allow the programmer to define and use graphical variables which have a specific type.

e.g. var s: square; t: triangle;

User may define their own graphical types as it is shown in the following example:

```
type  losange = figure (c: vector; a,b: real);
                var  x,y: vector;
                begin
                x := << a,0 >>; y := << 0,b>>;
                connect (c+x, c+y, c-x, c-y, c+x)
                end;
```

Standard types have been defined: square, circle, triangle, ellipse, line, segment and fig, which is a universal but unstructured type Graphical variables may be manipulated by procedures, assignments and image transformations as they are defined in [ 5 ].

### 2.3 The concept of graphical type checking

Assume that a programmer defines two variables of circle type;

```
var  c1, c2: circle;
```

then, he performs the following operations:

```
create  c1 (<< 2,3 >>, 10);  c2 := c1;
rotation (c1, origin, pi/3, c1);
translation (c1, << 2,3 >>, c1);
draw  c1;
```

These operations have the following meaning: creation of a circle c1 (with center < 2,3 > and radius 10), copy of this circle, rotation around < 0,0 > with an angle $\pi/3$, translation of < 2,3 > and drawing.

All these operations may be checked at the compile time, because it is well-known that a rotation and a translation preserve a circle.

However, it is possible to define image transformations which alter some graphical types. As an example, we define a shear along the x-axis.

```
transform  xshear (size: real);
var  y: real;
begin  y:= projy(oldfig);
newfig := << projx(oldfig) + size*y, y >>
end;
```

A transformation is similar to a procedure; the statements define the transformation to obtain a vector (called "newfig") of the new figure from a vector (called "oldfig") of the old figure. The transformation is implicitly done on each vector of the figure.

If we declare the following variables:

```
var  t: triangle,  c: circle; f: fig;
```

The following sequence of statements will produce the figure which is shown in appendix.

```
window (<< -10,-10 >>, << 10, 10 >> );
create  t (<< 1,2 >>, << 3,4 >>, << 2,6 >>);
create  c (<< -3,-2 >>, 2);
draw  t,c;
xshear (t, 0.5, f);  draw  f;
xshear (c, 0.5, f);  draw  f;
```

If we replace Xshear (t, 0.5, f) by xshear (t, 0.5, t), there is no problem, because a shear transforms a triangle in another triangle. However, a shear does not preserve a circle and xshear (c, 0.5, c) has to cause a runtime error.

Other operations require a runtime graphical type checking:

a) the standard procedure readgraph (f1, f2,...fn) which allows the user to enter interactively figures.

b) the standard procedure union (f1, f2, f3) which allows the user to build the figure f3 from the existing figures f1 and f2. If the type of f3 is not the fig type, it has to be checked.

c) the assignment; for example, if we declare:

```
var  t: triangle;  f: fig;
```

the assignment f:=t is always correct but the assignment t:=f is only legal if f is a triangle.

## 3. HOW TO CHECK GRAPHICAL TYPES

### 3.1 Philosophy of implementation

At the runtime, all figures are implemented as linked lists and it is not possible to have access to the original structure. The only way of checking if the figure has the good type is based on pattern recognition.

Our extension has been implemented by a preprocessor which produces a "standard" PASCAL program. Runtime type checking has to be performed by adding tests in the object code.

For example, if we declare

```
var  t: triangle;  f: fig;
```

f:=t will be translated into:

```
copy (t, f)
```

t:=f will be translated into:

```
if  istriangle  (f) then copy (f, t)
                    else  runtimeerror(4)
```

In the same way, the translation of the "xshear" statements that we have discussed in paragraph 2.3 is the following:

source code (MIRA 2D)

```
xshear(t, 0.5, f);
xshear(c, 0.5, f);
xshear(t, 0.5, t);
xshear(c, 0.5, c);
```

object code (PASCAL)

```
xshear(t, 0.5, f);
xshear(c, 0.5, f);
xshear(t, 0.5, t);
if not istriangle (t) then runtimeerror (4);
xshear(c, 0.5, c);
if not iscircle (c) then runtimeerror (4);
```

We show that the fig type is not checked in the first two statements.

### 3.2 Pattern recognition of very simple objects

Objects like triangle or segment may be easily identified, as it is proved by the following function definitions:

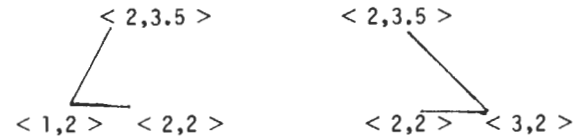
```
function  istriangle (f:fig): boolean;
var  cons, closed: boolean; n: integer;
begin  features (f, cons, closed, n);
istriangle:= cons and closed and (n=3)
end;

function  issegment (f: fig):  boolean;
var  cons, closed:  boolean; n: integer;
begin  features (f, cons, closed, n);
issegment:= cons and (not closed) and (n=2)
end;
```

Both functions use the procedure features (f, cons, closed, n) which checks if the figure f has consecutive visible segments, if it is closed and how many vectors compose the figure.

### 3.3 Problems of structure

The function istriangle which has been shown in last paragraph will cause problem if a triangle is built by the union of two half triangles as shown:




The new figure will have 6 vectors; that is the reason why we have implemented a procedure "restructure" which builds a new version of a figure by deleting all redundancy and unuseful vectors.

### 3.4 Comparison of objects

In case of figures like a circle, we create a figure which is a model for the graphical type that has to be checked. The figure is then translated in such a way that its center is the same as the center of the figure to recognize. Afterwards, the size is adjusted, and two typical vectors of both figures are computed; then, the model is rotated until the typical vectors match. At this time, a characteristic function is computed for both figures and it is decided if the figure is analog to the model.

This algorithm is only possible when the type defines a class of similarity; the corresponding function is the following:

```
function  sameshape (f1, f2: fig): boolean;
var ct1, ct2, pt1, pt2: vector;
    alpha1, alpha2, dmax2, dmax1: real;
begin ct1:= center(f1);  ct2:= center(f2);
translation (f1, ct2-ct1, f2);
distmax (ct2, f1, pt1, dmax1);
distmax (ct2, f2, pt2, dmax2);
homothety (f1, ct2, dmax2/dmax1, f1);
alpha1:= arctan ((projy(pt1)-projy(ct2))/
                 (projx(pt1)-projx(ct2)));
alpha2:= arctan ((projy(pt2)-projy(ct2))/
                 (projx(pt2)-projx(ct2));
rotation (f1, ct2, alpha2-alpha1, f1);
sameshape:= eqfig (f1, f2)
end;
```

The boolean function eqfig determines if both figures are "equal". In fact, eqfig calculates a few characteristics of both figures and compares them.

Many choices are possible for these characteristics but they may be classified into two classes:

i) statistical analysis
   e.g. comparison of averages, variances, correlations
ii) topological analysis as:
   a) correspondence of point types as defined by Nagao [ 6 ]
   b) comparison of relative positions as it has been used in recognition of hand-printed [ 7 ] and handwritten [ 8 ] text.
   c) comparison of line drawing analysis; this kind of analysis may be based on structural units as it was developed by Morofski and Wong [ 9 ] in PPS.

In the case of standard and simple graphical types, we have chosen a statistical analysis.

We compare averages and variances of vectors of both figures. However, we add, in the statistics, the middle vectors of all visible segments.

## 4. USER GRAPHICAL TYPE CHECKING

### 4.1 Problems with user graphical type checking

As users may define their own graphical types, type checking method is dependent on the kind of type. For example, the two following types can not be checked in the same way:

```
type
   regularpolygon = figure(center: vector;
                           length: real; nside: integer);
   regularhexagon = figure(center: vector;
                           length: real);
```

The first type is very difficult to check, because a square or a regular hexagon are regular polygons and it is not possible to use the algorithm described in paragraph 3.4. For the second type, there is no problem.

### 4.2 Directives to the preprocessor

The only way of providing type checking is to give the user the possibility of specifying his type checking method. As we consider that type checking is not a characteristic of the language but an implementation feature, we prefer that type checking specifications are introduced as directives to the preprocessor. These directives have to be given in a PASCAL comment (as directives to the PASCAL compiler). Such a comment has to begin with a character '!' . Each directive consists of a letter followed by a character '+' if the method has to be used, or a character '-' if the method is not used. Ten directives are available:

C: check if the figure has consecutive visible segments

D: decompose the figure into simpler parts and checks if the different parts are compatible with the type definition.

I: (see paragraph 4.3)

K: check if the figure is closed.

L: check if all sides have the same length

N: check if the figure has the same number of vectors than it is defined in the type

R: in case of type error, the figure is restructured and checked again

S: type checking by a statistical comparison

T: type checking by a topological comparison

Z: suppress all type checking.

Different directives may be used simultaneously.

e.g.

```
   type
     (*!N+, K+, L+*)
     equilateral = figure (a,b,c: vector);
                   begin connect (a,b,c,a)
                   end;

     (*!L-*)
     quadrilateral = figure (a,b,c,d: vector);
                     begin connect (a,b,c,d,a)
                     end;
```

A variable of type equilateral must be closed, must have all sides with the same length and the number of vectors will be checked. For a variable of type quadrilateral, there is no restriction on the side length.

### 4.3 Discussion of some directives

Topological comparisons are obtained by a technique which is similar to the method of Morofsky and Wong [10]. It means that the analysis of patterns is based on the recognition of junctions:

T- junction 

X- junction 

K- junction 

Y- junction 

Comparisons are also based on angle measures. The directives 'D' should be used when a graphical type is defined by inclusion of simpler figures, which is possible with the include statement.

e.g. type (*!T+,D+*)
doublecircle=figure(c:vector; r:real);
var c1, c2: circle;
begin
create c1(c-<< r,0 >>,r);
create c2(c+<< r,0 >>,r);
include c1, c2
end;

In the case of an operation like union (f1, f2, f3), if the type of f3 is "doublecircle", the type checking algorithm searches for two circles.

The directive 'I' has to be followed by a real positive number less or equal to 1. This value is a tolerance factor; the higher is this value, the more exact pattern is required. Such a technique was already introduced in ESP$^3$ by Shapiro [11]. The default tolerance factor has been fixed to $1-10^{-10}$.

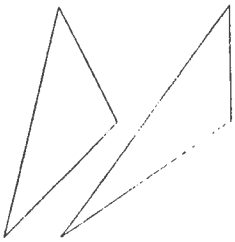
## 5. CONCLUSION

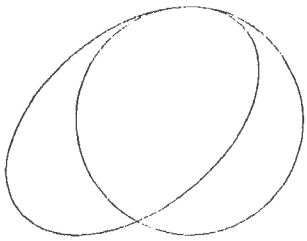
Graphical type checking is a new concept, because graphical types have been introduced only recently. Methods have to be developed using techniques in pattern recognition. Our approach is sometimes empirical and types like polygons with n sides (where n is a parameter) can not be checked. Further investigations in this domain have to be done.

## 6. REFERENCES


1. Wirth, N. "Algorithms + Data Structures = Programs", Prentice-Hall, 1976.
2. Hoare, C.A.R. "Notes on Data Structuring", in Structured Programming, Academic Press, N.Y., 1972.
3. Magnenat-Thalmann, N. and Thalmann, D. "A Structured Approach to Computer Graphics", Proc. 6th Man-Computer Comm. Conf., NRC, Ottawa, 1979, pp. 139-150.
4. Magnenat-Thalmann, N. and Thalmann, D. "A Graphical PASCAL Extension Based on Graphical Types", Software-Practice and Experience, 10, 1980.
5. Thalmann, D. and Magnenat-Thalmann, N. "Design and Implementation of Abstract Graphical Data Types", Proc. COMPSAC'79, Chicago, IEEE Press, 1979, pp. 519-524.
6. Nagao, M. "Picture Recognition and Data Structure", Proc. IFIP Working Conference on Graphic Languages, North-Holland, 1972, pp. 48-68.
7. Groner, G.F. "Real-time Recognition of Handprinted Text", Proc. Fall Joint Computer Conference, 1966; pp. 591-601.
8. Wolff, H.J.G. "Recognition of Handwritten Capitals Based on the Use of a Line Follower, Proc. Seminar on Pattern Recognition, Liege, 1977.
9. Morofsky, E.L. and Wong, A.K.C. "Computer Perception of Complex Pattern", Proc. 2nd Intern. Joint Conf. on Artificial Intelligence, British Computer Society, 1971, pp. 248-257.
10. Morofsky, E.L. and Wong, A.K.C. "Isolating and Identifying Objects in Line Drawings", Proc. 4th Intern. Joint Conf. on Artificial Intelligence, 1975, pp. 656-663.
11. Shapiro, L.G. "Inexact Pattern Matching in ESP$^3$", Proc. 3rd Intern. Joint Conference on Pattern Recognition, Coronado, 1976, pp. 759-763.

# APPENDIX





# ON THE DESIGN OF AN INTELLIGENT TERMINAL FOR VOICE OUTPUT IN PROGRAMMING

T. Radhakrishnan and C. Labrador

Computer Science Department
Concordia University
Montreal, Canada
H3G 1M8

ABSTRACT

This paper discusses the application of voice-output in programming as well as important techniques and problems intrinsic to voice-input. A prototype voice response system for PASCAL programming is presented, and applications to alternate programming languages are considered. The introduction of microprocessors to implement the voice response unit as well as the software modules involved are covered and the applications and advantages of such "speaking terminals" with respect to the programmer who is visually handicapped are discussed.

## I. INTRODUCTION

In the general area of man-machine communications by voice, there are three major areas of interest for researchers [1] : (a) voice response systems, (b) speaker verification and speaker identification, and (c) recognition of spoken utterances. These major areas may be subdivided into a large number of sub-areas, depending on such factors as the vocabulary size, static or dynamic vocabulary, speaker population, speaking conditions, requirements of the end-user etc. Central to a voice response system are, a vocabulary store and a set of rules for message formation. Figure 1 shows a block diagram of such a system. When a message request is received from an external source, the message composition program composes the required message by referring to the vocabulary store and the message formation rules. The composed message is in an acceptable form to the voice synthesizer which produces the voice response. An adaptive system may allow dynamic changes in the vocabulary, in message formation rules, or in both.

Several applications of voice response systems have been considered in the literature. For example, directory assistance systems, stock price quotation systems, flight information systems, and voice response systems for wiring communication equipment are discussed in [2]. Consider an interactive process, such as computer programming, carried out by a visually handicapped person. There are two directions of information flow: (a) Man->machine, such as program input, data input, or input for the correction of a program statement in error; (b) Machine->man, such as the errors in input, diagnostic messages from compilation or execution of the programs submitted, or the results generated by the program. A visually handicapped person who has experience in typing, faces no serious problems in man->machine communications, however for machine->man communications, he needs assistance, since he cannot read by himself. Conventionally, such assistance came from Braille terminals [3] or from the readings of a sighted person. Though braille terminals are helpful, they are not always satisfactory. For instance, they require "feeling by fingers", produce voluminous hard-copy output, and they are expensive. Furthermore, working with these special kinds of terminals, distinguishes a visually handicapped programmer from his fellow programmers and this might be undesirable in some cases. It is in this context a voice response system for programming by a visually handicapped computer programmer, is discussed in this paper.

## II. VOICE INPUT AND VOICE OUTPUT IN PROGRAMMING

The following stages may be noted in the development of a program:

(1) Program Design
(2) Coding
(3) Program Input
(4) Compilation
(5) Correction of Syntax Errors
(6) Execution
(7) Verification

Some of these stages may be repeated more than once in the program development cycle. Both man->machine and machine->man communications are involved in the stages cited above.

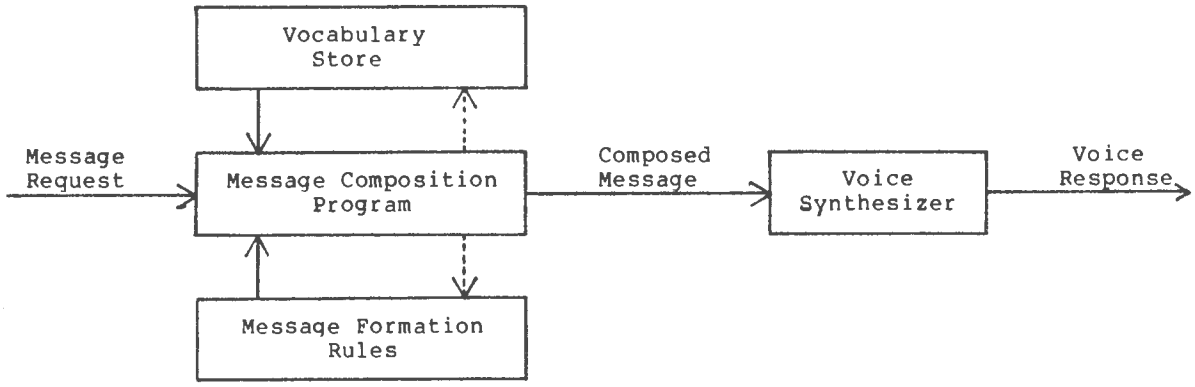



Fig. 1 Block diagram of a Voice Response System

Use of a programming language for communication has certain notable differences when compared to the use of a natural language. The syntax and grammer of a programming language are much more rigidly defined than those of a natural language. A program consists of several "words" or names and operation symbols such as +, -, *, /, =, ||, etc. Some of the words used in a program are reserved words and others are defined by the programmer. For a given programming language, say PASCAL, the set of reserved words are known a priori and they constitute a fixed vocabulary for a voice response system. The set of programmer-defined names is not invariant and it changes from program to program. However, while programming in a language like PASCAL, the set of such programmer-defined names are defined in the declaration part, before they are used in the program. Thus, for voice-input in programming, it might be possible to train the speech recognizer unit on the programmer-defined names. The knowledge obtained through such training may then be applicable for the duration of that program with that particular programmer or speaker.

It is well known that recognition of isolated words or discrete speech is a simpler problem than the recognition of connected or continuous speech. Occurences of isolated words are more intrinsic in the statements of a programming language than in the sentences of a natural language. Consider speaking the statement COUNTER = COUNTER + 1. As an extreme case of this situation, some names in programs may not be readable as a word. For example, the name "SQZT" can not be read as a word and hence has to be read letter by letter. These factors lead to less stringent conditions for speech recognition in the case of voice-input for programming. An experimental man->machine voice communication for programming in the BASIC language has been reported in [4,5]. The results reported have been encouraging, though not totally satisfactory. Initial tests have shown that utterances can be recognized in approximately one quarter of the real time with an error rate and rejection rate of 9.6 and 6.0 percents respectively.

Applications of syntax directed techniques for pattern recognition are well known [6]. Similar approaches may be useful in speech recognition, especially in voice-input programming. Advanced programming techniques such as interactive graphics, involve higher dimensional data structures than simple linear strings. Syntax directed techniques will be appropriate for voice-input of such higher dimensional data types.

Since voice response systems produce synthesized speech utterances that are used for communications with humans, intelligibility is of paramount importance. Also, subjective factors such as quality and naturalness have been found to have a great effect on the acceptance of a voice response system [1]. However, while designing a voice response system for the visually handicapped programmers, one could trade the subjective factors for the desirable system parameters like low cost and compatibility with other programmers for example.

Browsing is one of the activities that a human eye (with the mind) can do more efficiently than others. One form of browsing occurs in program debugging. Suppose the compiler of a program reports an error in the J-th line of the program. It is possible, in some cases, that the error is actually in the (J-1)-th line; but the compiler has detected the error

while processing the J-th line. A human eye looking at the J-th line is capable of "browsing" through the neighbouring lines and often it locates such errors. A voice response system designed for visually handicapped programmers should have facilities to cater to such needs. In essence a good text editor with voice-ouput will be of help in this direction. A blind person, using a terminal equipped with such an editor, may wish to hear a line repeatedly for intelligibility. Thus, for example, "REPEAT CURRENT LINE" would be a desirable feature in the speaking text-editor.

## III. A VOICE RESPONSE SYSTEM FOR PASCAL PROGRAMMING

For the experiment discussed in this paper, the popular programming language PASCAL has been chosen [7]. The experimental voice response system can speak-out the error messages detected by the PASCAL compiler when compiling a PASCAL source program, as well as the lines or statements selected by the programmer from the program. The commercially available voice synthesizer Votrax [8] has been used in the present experiment. It is a phoneme based synthesizer that employs analog methods for voice synthesis. Votrax accepts input in its own code which will henceforth be referred to as votrax-code. Besides the Votrax, the experimental system consists of the following software modules which will eventually be transferred to a micro-computer (Section IV):

Compiler Interface Module: From the compiler output file, this module selects the error messages or the error codes and the statements in error.

Editor Interface Module: This provides an interface between the conventional text editor and the visually handicapped programmer.

Voice Interface Module: This module is an interface between the voice synthesizer and the other parts of the system. Its functions are the same as that of the voice response system shown in Fig. 1. It makes use of a set of vocabulary, VSET, and a set of rules, MSET, for message composition. Also it produces an output that is acceptable to the voice synthesizer.

There are 128 error messages corresponding to associated error numbers in the PASCAL compiler used in our experiment. The number of words in a message varies from 2 to 10. There are 182 distinct words, each of which occurs with different frequency in the set of error messages. The word that occurs most frequently is TYPE (32 times). It may be remarked that the distribution of the frequencies of the phonemes, which compose these words, follows the well known Zipf's Law [9] closely enough to warrant interest. Given the law as being RANK * FREQUENCY = CONSTANT, C, the average value for C derived from the raw data is 0.202. Assuming that Zipf's Law is adhered to, the predicted value for C is 0.193. Although this is an empirical law, it forms a useful formula for prediction of approximate phoneme frequencies and is a useful tool in the design of appropriate storage and retrieval schemes for a large phoneme/word database.

The VSET in our experiment consists of two parts; VSET-1, the set of distinct words in the error messages and VSET-2, the set of single characters such as A, B, ..., Z, 0, 1, ..., 9, etc. Each member of VSET is stored in a table along with the votrax-code for speaking that member.

Generation of votrax-codes for the words in VSET-1 can be made automatic through a program. The approach used in the automatic translation of English text to phonetics in [10] has been used in our experiment as the starting point. But the initial results obtained from [10], for the set of words in VSET-1, did not give good quality sound output. The output obtained from [10] for each word of VSET-1 has been manually tuned to improve the sound quality by changing the phonemes, the inflections, or both.

For each error message in the compiler, there is a corresponding member in MSET which gives the concatenation rules for message composition. The rules in MSET refer to the words in VSET. The message is composed by the voice interface module and then passed to the voice synthesizer. The software modules have been designed so that they may be easily adapted to support different programming languages. However, VSET and MSET have to be generated a priori for each of the programming languages to be supported.

## IV. AN INTELLIGENT TERMINAL FOR VOICE OUTPUT

With the advent of microprocessors and LSI technology, it has become possible to distribute some processing power to the otherwise dumb terminals. Also digital representation and processing of speech

signals have become a reality [1]. Commercial products such as the Texas Instrumants SPEECH MODULE, TM 990/306 [11], are now widely available. For instance, this speech module has a fixed vocabulary of 179 words whose digital representations are stored in electrically Programmable Read-Only Memories (PROM). Combining the advances in microprocessors and digital speech processing, it is possible to design an intelligent terminal that will have a voice response system. Yet, another approach would be to augment a conventional terminal with a voice-output unit as shown in Fig. 2. This system is well suited for use by visually handicapped programmers, since the voice-output unit can be "easily" plugged into a conventional terminal.

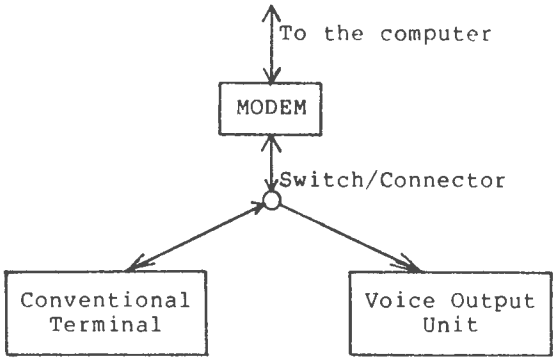



Fig. 2 Block diagram of a Voice Response Terminal

The voice-output unit of Fig. 2 will consist of a microprocessor, some local memory (ROM) for program storage, some local memory for data storage (RAM), and a speech synthesizer. With the availability of single board microcomputers, and the single board speech modules like the TM 990/306, it is possible to design a compact portable voice-output unit. Use of the components and subsystems available "off the shelf", would render such a system to be inexpensive and hence affordable to individual users. However this scheme is not without its limitations.

The microprocessor in the voice-output unit of Fig. 2 will implement the functions of the software modules discussed in section III. The information or the data flow between the conventional terminal and the computer is monitored by the voice-output unit; and if desired the monitored information is also spoken out. The voice-output unit has its own limits on the output rate which are determined by such factors as the processing rate of the voice synthesizer, and the acceptable input rate of the receiver, the human ear. If there are no buffers and inter-locking mechanisms, the voice-output unit should function at least as fast as the flow rate between the terminal and the computer. The experimental system discussed in section III, for example, functions at a rate of 300 baud.

It is possible to extend the storage and processing capabilites of the microcomputer. Then, programs and the results of their processing may be stored in the local memory of the voice-output unit for ready access. Even if the rotating memories such as floppy disks are not preferred from the portability point of view, new memory technologies like magnetic bubble memories [12] or charge coupled devices may be used to extend the storage system. A single board reported in [12] provides up to 64K (K = 1024) bytes of non-volatile memory with a 4 millisecond access time and a 50 kilo-bit per second data transfer rate. Higher storage volumes, up to 768K bytes, are expected to be available in the first quarter of 1980.

Translation of computer produced results to voice-output is not always a trivial task. Presentation of results in the form of tables, graphs, or charts is well suited for sighted persons; but reading of such data to communicate to a visually handicapped programmer is non trivial. Besides voice synthesis, there are other problems to be solved in this context. Algorithms for solving such problems may be implemented on the "microprocessor" and the data stored in local memories may be refered to as often as needed.

## V. CONCLUSION

A prototype voice response system for programming in PASCAL has been constructed. It has been used by a visually handicapped programmer and has been found to work satisfactorily. In this paper, two approaches to the design of an intelligent terminal for voice-output have been discussed. The prototype system is being implemented on a Motorola 6809 microprocessor. Research is continued in the areas of reading complex data types like graphs, tables, and charts; and in the area of syntax directed recognition of "spoken programs".

Acknowledgements:

The financial support provided by the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged. The authors are also grateful to Professors C. Y. Suen and A. K. Menon of the Computer Science Department of Concordia University for their helpful discussions.

REFERENCES

1. Rabiner, L.R. and Schafer, R.W., "Digital Processing of Speech Signals", Prentice Hall 1978 (Chapter 9).

2. Rabiner, L.R. and Schafer, R.W., "Digital Techniques for Computer Voice Response : Implementation and Applications", Proc. IEEE 1976, Vol. 64, No. 4, pp. 416-432 (special issue on Man-Machine Communication by Voice).

3. _____A Low Cost, Interactive Braille Computer Terminal for the Blind, ISG-1. Information Systems Inc., 3132 S.E. Jay Street, Florida 33494.

4. Niimi, Y. and Kobayashi, Y., "A voice-input programming system using Basic-like language", IEEE Int. National Conference on Acoustics, Speech, & Signal Processing, 1978, pp. 425-428.

5. Niimi, Y.; Kobayashi, Y.; Asami, T.; and Miki, Y., "The speech recognition system of "Spoken-BASIC", second US-Japan Computer Conference, 1975, pp. 375-379.

6. Fu, K.S., "Syntactic Methods in Pattern Recognition", Academic Press 1974.

7. Jensen, K. and Wirth, N., "PASCAL: User Manual and Report", Springer-Verlag, 1975.

8. _____Votrax Audio Response System - VS 6.0 Vocal Interface Division, Federal Screw Works, 500 Stephenson Highway, Troy, Michigan 48084.

9. Zipf, G.K., "Human behaviour and the principle of least effort", Reading, Mass., Addison Wesley, 1949.

10. Elovitz, J. et al, "Automatic Translation of English Text to Phonetics by means of letter to sound rules", Naval Research Report, NRL 7948.

11. _____TM 990/306 Speach Module Data Manual
Texas Instruments Inc.

12. _____TM 990/210 Magnetic Bubble Memory Systems
Texas Instruments Inc.

# An Adaptive Sorting Program

Oliver G. Selfridge
Valerie I. Congdon
Stephanie R. Davis

Computer and Information Science
University of Massachusetts
Amherst, Massachusetts

ABSTRACT

This paper discusses the design, construction, and use of an adaptive sorting program, which selects and tunes the sorting algorithm according to its recent experience with the algorithms available to it. That is, the program adapts its behavior to try and minimize a cost function specified by the users.

The point of this exercise is to explore ways in which the computer program can carry some of the responsibility of optimizing its performance, instead of relying on a user to set rigid specifications. The purpose of choosing a good sorting algorithm is to minimize some kind of cost; the cost function used here is "virtual CPU time," computed by the program; we use that instead of measuring real CPU time because of the difficulties and unreliabilities of measuring it in a time-sharing environment.

Some of the adaptive programs discussed here perform better on some populations of lists than the standard workhorses found in many computer centers.

## 1.0 Introduction and Overview

There are a number of different algorithms that can be used to sort lists. Each has advantages and disadvantages that depend on the nature of the lists. This paper discusses an adaptive sorter, which selects the particular algorithm it uses according to the efficiency it has previously found. The sorter restricts itself to three algorithms: Straight Insertion, MergeSort, and QuickSort. It should be noted that each has operating parameters that have to be set before it can be considered well specified.

The task domain is the selection of algorithms to sort lists. For this paper, the lists were generated by a program. For some of the exercises, the characteristics of the lists changed slowly in time -- that is, there is not merely a single optimum sorting algorithm that is to be searched for, and, once found, maintained. Rather, the program must be capable of changing its selections as its environment changes.

In Artificial Intelligence (AI), there is another approach to this problem, namely the use of experts: find out from them the rules they follow, the diagnostics they use, and so on, and design those rules into a program. Such a technique has been very successful in some cases.(1) But sometimes the domains are too large to have rules of the necessary precision; or the rules seem to involve human judgment in a profound way, as if in artistic selection; or the best experts are just not very good; and so on. Sometimes the criteria for an "optimum" change -- in our example, the evaluation or cost function might be changed to include some measure of the cost of storage. That is, our underlying interest here is the use of adaptive techniques where there may be no experts, or where the problems may be too hard to understand or even to state. We use sorting as a domain, where there are experts, so that we can compare the efficiency of the adaptive techniques with that of the experts.

Our purpose here is to see to what extent we can give the program the responsibility of choosing the "optimum" algorithm, by providing it the experience of trying several algorithms and remembering how

(1) For example, see Feigenbaum and Lederberg (1974), with Dendral and Metadendral.

they performed (with respect to the cost function that defines the optimum).

Any program that can track a moving optimum must spend some extra effort deciding what the current optimum is, just as the experts spend effort on their diagnostic analyses. Our program is constantly checking the current best algorithm against its competitors; it does so less often when, whenever it does, the comparison is very one-sided, and more often when it isn't.

In general it is not possible to compare two algorithms with just two trials, because each has parameters that must be set for it to do best, so that the ideal settings have to be tracked. In the first set of experiments, however, we set the parameters by hand. A second set tries to optimize the parameter settings as well. A third set tries to find a good combination of useful diagnostics to help the program determine a good threshold function to help make a good selection of algorithm.

The general adaptive approach that the program exemplifies is discussed at length, considering especially its inherent limitations and the underlying assumptions about its application. For many domains of AI, we suggest that AI cannot afford the time spent to tap experts, and ought to try giving the program some responsibility in improving its behavior; that would be even truer if the experts were scarce or not very good. This program is, we hope, a beginning exploration of ways to do that.

At least one of the programs shown here seems to do better on the average than some of the workhorses used at computer installations; while we cannot be certain that it would be profitable to substitute them, it would certainly be worthwhile to check them with a broader range of sorting problems and algorithms, perhaps referring to human interaction, and to the expert work of, for example, Knuth (1974).

An example of the kind of problem that might be susceptible to the approach is the scheduler on a time-sharing system, where it is hard to decide what is the best way to satisfy user requirements, especially as they and the system change in unknown and unpredictable ways; another example is the control of a communication net with large swings in the volume and nature of the traffic, subject to changing priorities and channel capacities.

We use three sorting algorithms, Straight Insertion, Mergesort, and QuickSort, each of which is good for some applications, and which are described in section 2. The adaptive techniques are not particularly subtle, and are described in section 3. The structure and functions of the program as a whole are described in section 4, and section 5 presents the results of the several experiments. The final section discusses the results and draws conclusions from them.

## 2.0 **Background: Sorting**

Sorting is the task of arranging items in some desired order, like alphabetical. It was one of the first tasks assigned to automatic data processing machines, and was one of the first such problems to be thoroughly analyzed.

We chose sorting as a task domain for several reasons:

1. It is a well-known problem with several commonly used algorithms, whose relative costs vary widely with the sorting problems presented to them.

2. Sorting algorithms have been thoroughly analyzed, so that experts can be reasonably sure about the rules they follow; in that way the performance of the program can be properly evaluated.

3. We felt that it was likely that if the program worked as well as we hoped, it could lead to profitable improvements over some of the standard work horses now being used in computer installations.

The program deals with 3 sorting algorithms: Straight Insertion (I), MergeSort (M), and QuickSort (Q). These are described in the following subsections.

### 2.1 STRAIGHT INSERTION (I)

Insertion adds items one at a time to a previously ordered list. Since each insertion leaves the list ordered, a list n long takes merely (n-1) insertions. In the worst case, I makes i-1 comparisons to insert the ith item, so that the number of comparisons is O(n*n). Note that if the list is highly ordered to start with, then each insertion may be done with very few

comparisons. It is also usually a fast method for lists with few items, say, fewer than 15.(1) The program itself is short and easy to understand. Some other sorting methods may use it with short lists or sublists.

### 2.2 MERGESORT (M)

Merging is the technique of combining lists (in this case, two) by sequentially comparing the first elements of sublists, and moving them in the correct order into the merged list. M combines pairs of single elements into sublists, then the pairs themselves, and so on, each time dealing with sublists twice as large, until the process terminates.

M is an efficient sort, and is currently the system choice at the computer center at UMASS. It does, however, require a lot of storage. In running time, it takes O(n*log n). Its worst case is never much worse than that average.

### 2.3 QUICKSORT (Q)

Q is on the average the best sorting algorithm according to the experts.(2) Q is based on the notion that exchanges of items should be made over large distances in order to minimize the number of exchanges -- it is the diametric opposite of bubble sort, for example, which exchanges items out of order only with their neighbors. Q makes an arbitrary partition of the list into two parts, comparing items from both parts, and interchanging them when needed.

The running time for Q is on the average O(n*log n), but its disadvantage is that its worst case performance can be O(n*n), which is not true of M.

## 3.0 Background: Adaptation and Computer Learning

There is a long and rich history of attempts to make the computer (program) learn in the sense that children learn and grow. Nearly twenty years ago, there was a great deal of interest in 'self-organization,' by which the computer was to organize its data and restructure itself so as to perform better. Some of us remember the perceptron of the late Frank Rosenblatt(1) and its numerous companions. Recently such activity has waned, perhaps because of a more or less conscious decision by the AI researchers that it was not very productive. One of the questions that we raise here is whether the simple control mechanisms that we discuss can be applied to hard problems so as to make a beginning of an attack on the larger area of learning by computers.

The advantages of adaptation and learning, if any, are not had for nothing: it will always cost extra resources to make checks on the efficiency of the particular algorithms being used.

One method of improving a strategy is to try small changes in it, observing the changes in performance. If they are positive, continue to make such changes; if negative, undo them, and try other ones. This is generally known as hill-climbing, and it has a venerable history. Much of the power and difficulty of hill-climbing depends on the particular representation of the strategy, so that the changes are in some way related to the changes in performance. Indeed, AI researchers have long considered that the problem of finding good representation is one of the truly central ones in AI.(1)

## 4.0 The Adaptive Sorter

We deal with three different kinds of adaptive mechanisms. In the first, the program is given the cost of the algorithm when it tries it; its goal is to minimize the total cost of a long series of sorting problems. In order to make sure that the algorithm it is using is the best (that is, the cheapest), it must occasionally try the other algorithms, which costs it more resources. The underlying assumption behind this strategy is that the sorting problems in the series vary their characteristics only slowly, so that the best algorithm stays best for some long time.

---

(1) See Knuth (1974).
(2) Ibid, and Wirth (1976).

(1) For example, see Rosenblatt (1960). For the best discussion of perceptrons, see Minsky and Papert (1969).

(1) See Winston (1976) for a good general discussion.

The characteristics of the lists to be sorted that are relevant here are two: the length L, and the degree of randomness R. The latter is interesting -- Q, for example, takes nearly as long to sort a list that is already sorted as to sort one that is in random order. Straight insertion, on the other hand, takes but L - 1 comparisons to establish that a list is well-ordered.

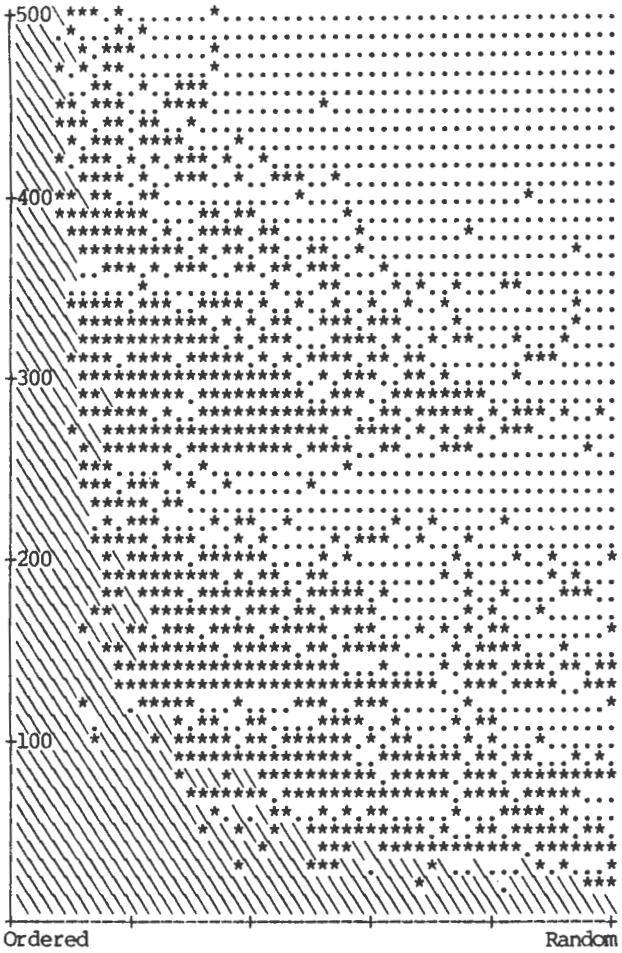



Figure 4.1 Optimum Method for Sorting
X-axis: Randomness R
Y-axis: Length L

\ : I . : M * : Q

Figure 4.1 is a length-randomness plot, with the length shown vertically, and randomness horizontally. The figure shows the regions where the three algorithms are superior to the others. The figure was constructed by generating 2500 lists using a random number generator. In general, small lengths or low randomness suggest I; M and Q are in fact fairly close over the remaining region, and the superiority of one over the other is usually but slight. It is possible to observe the extra time taken by M as the length rises above each power of 2. Clearly, if the program could detect the best method cheaply enough, perhaps it could make significant savings in resources.

In the second kind of adaptation, the program takes advantage of knowing, either by experiment or by our having told it, the contents of figure 4.1. The program's task is then to make good estimates of length and randomness. In our sorting, the length is provided as a given with the submission of the list; the question is then how to estimate the randomness cheaply enough to make it profitable.

A third experiment in adaptation selects a good simple combining function of L and R.

There is a kind of zeroth order adaptation, in which the best overall method is used consistently; given the population of lists that we presented to the machine, and presuming a uniform distribution of lists over the variables L and R, that method was in fact M. the first job of any adaptive scheme, obviously, must be to do better than M.

## 5.0 Experiments and Results

We ran a number of experiments with some interesting results. The first task was to generate the lists; how we did that is described in section 5.1 below. Each list was in fact evaluated separately with each of the three methods, and what was presented to the program was merely the characteristics of each list and the resources it would take according to the three methods. In that way, it was possible to compare different adaptive strategies without actually running the algorithms over and over again, which would have consumed considerable computer time.

### 5.1 Generation of Lists

The lists had two controllable attributes -- length L and randomness R. The value R set the fraction of the list that was constructed **at random**, the other elements being generated **in order**. For example, for a list 100 long, the ith element defaults

to just i itself, for R = 0, the perfectly ordered case. If R is 0.5, then in exactly half the elements, the value chosen is just 100 times a random number uniformly distributed between 0 and 1.

Note that lists can be generated with **negative** ordering, that is, backwards, but we did not use such lists in our experiments.

For each experiment we generated lists that form the basis of the data used in the adaptation. Each list was then sorted by all the three methods. The data is contained in an array whose columns were:

1. The length of the list
2. The randomness of the list
3. How long it took to sort with I
4. How long it took to sort with Q
5. How long it took to sort with M

For each test of the adaptive strategy, we computed the costs by merely referring to the array, instead of generating lists and sorting them. In this way the individual adaptive run could be tried with very little CPU time.

Inspection of figure 4.1, generated in this way, will reveal a certain noisiness in the data. That arises from the use of the random number generator in making the lists.

## 5.2 First Adaptive Scheme

In the first scheme, the program initially tries the three algorithms, and then continues by using only the best one. Best is defined by an estimate of the CPU time used by the algorithms. The program checks the validity of its choice by trying the other ones occasionally; if one of the other algorithms proves to be shorter, the program switches. The underlying assumption for this scheme is that the attributes of the lists do not change very fast.

In fact, the lists were selected from a population whose characteristics followed the trajectory shown in figure 5.1, which covered 1000 lists. Note, by comparing that figure with figure 4.1, that the trajectory runs through all three regions where the different algorithms were optimum.

The program worked by computing the cost of the preferred sorting algorithm. Some fraction of the time, it also tested by trying the other algorithms; that meant adding their costs to the costs already incurred. The cost of the tests on the non-preferred algorithms was minimized by ceasing the test whenever its cost exceeded that of the preferred one.

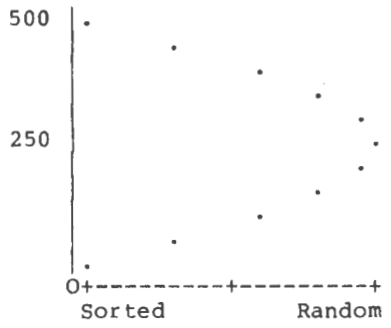



Fig. 5.1 Trajectory for a Population of Lists

The behavior of the program is about equal to the best single algorithm, M. In this case, then, adaptation does not provide any real profit. The extra expense of making checks causes the program not to outperform the best single algorithm in a significant way.

This kind of scheme is efficient only when the population characteristics change slowly, of course. In that sense, it is unrealistic to expect that it can provide a vast improvement in sorting efficiency in an operational environment. By analogy, however, we might hope that such a scheme applied to tasks harder to analyze, like certain kinds of scheduling in time-sharing systems, could lead to really worthwhile savings.

## 5.3 Second Adaptive Scheme

The second scheme used the data shown in figure 4.1. The program has but to know where on the figure the new list is, and it can choose the best sorting method. The length L is given; since the program is not given R, it has to estimate it. The adaptation here is how much resources to put into estimating R. If the whole list is examined, that represents a fairly massive expense; and if, say, but ten items are examined, then there is a fairly large

chance of making a bad estimate. The parameters of the procedure for estimating R are tuned so as to minimize the cost. The difficulty with such a scheme is the length of time needed to be sure of the results of tuning. It is clearly cheaper to use short samples, except that then the probability of picking the wrong selection by chance increases; that may lead to the selection of a much more expensive algorithm.

We decided that the parameter of estimation that ought to be optimized was the fractional size X of the sample that gave the estimate of R. That is, if X were 0.10 then a list of length 500 would be tested for R with a sample of length 500*0.10=50. The estimation does not pretend to be an accurate measure of randomness, which is in any case undefined except insofar as it is defined by the generation process itself.

If Y is the fraction of successive differences between successive elements in the list that are negative, then the program makes the randomness estimate R´ = 2*Y. The process is illustrated

| LIST | | | | | | | | | | Y | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | | | | | 0.0 | 0.0 |
| 0 | 6 | 2 | 3 | 4 | 5 | | | | | 0.2 | 0.4 |
| 6 | 10 | 1 | 5 | 4 | 9 | 2 | 3 | 8 | 7 | 0.5 | 1.0 |

and these agree obviously with the generation process in a gross way. The cost of the estimate is some approximately linear function of the number Ll=L*X of the items in the sample. There is no a **priori** reason why the optimal value of X should not be a function of L, and in truth it may be; furthermore, the optimum is not even well defined in any absolute sense, and must depend on the distribution of the population. This point is considered further in section 6.

Once the (R,L) was established for the given list, the method to be selected was found by examining the region around the point (R,L) in figure 4.1. Since that figure is drawn from noisy data, as discussed in section 5.1, we averaged the region around (R,L), using a 5X5 window, and the program chose the most frequent best algorithm in that window.

We selected a population of lists so as to exaggerate the effects and success of the scheme, by picking lists where the differences in performance are marked. The results for various values of X, that is, the fractional sample size for estimating S, are shown in the table:

| | COSTS of Sorting | Estimating | Total |
|---|---|---|---|
| X | | | |
| .05 | 2930 | 16 | 2946 |
| .10 | 2817 | 33 | 2850 |
| .15 | 2856 | 50 | 2906 |
| .20 | 2833 | 68 | 2900 |
| .30 | 2808 | 102 | 2910 |
| .50 | 2823 | 171 | 2994 |
| 1.00 | 2812 | 344 | 3156 |

The COSTS are in arbitrary units. This shows a shallow but definite minimum at X=0.10.

Using this value of X, then, let us compare this program with the single algorithms separately, and with the best possible selection (BP):

| Program | M | Q | I | BP |
|---|---|---|---|---|
| 2821 | 3457 | 3476 | 17991 | 2744 |

So it is clear that the program is not quite the best possible, but is still some 20% better than M or Q, even allowing for the extra resources used in making the estimate.

## 5.4 **Third Adaptive Scheme**

The third scheme illustrates the selection of a usable adaptive method from a set of possibilities. Each method is not precisely prespecified, but must be adaptively improved before it can be reasonably evaluated. This does not demonstrate the full construction of a possible complex processing scheme from a **tabula rasa**, but it does show how easy it is to test possible combinations of simple subprocesses to generate useful processes.

The supposition is that the program knows, from previous experiences that we do not specify, that the values R and L are significant parameters in the choice of the best sorting algorithm. What the program does not know is how best to combine those two parameters to get a reasonable function that will help to make the decision about the algorithm to be selected.

Again, we used the data that produced figure 4.1. The figure was divided in regions bounded by smooth curves, so as to smooth over the irregularities caused by the random number generation that made the figure. The boundary of the region where I is the best policy is like a hyperbola, for example, separating it from where Q is

best. The boundary between Q and M is somewhat more complicated, and shows clearly that M (if we didn't know) is a binary merge, losing a little efficiency relative to Q every time the length L rises above another power of 2. For the sake of simplicity, this scheme selects only between Q and I, ignoring M: that is because the boundary between Q and M is not easily describable.

The point of this scheme is not to have to store the relatively large amount of data in figure 4.1, but to approximate its contents with a simple formula. We suppose that the program does not initially have the concept or idea of hyperbolas, and cannot make algebraic inferences from pictures like the figure. What it can do is to try schemes and pick the best performing one. This conceptually simple plan is complicated by the fact that each scheme will be seen to have parameters of operation, just like the one in section 5.3 above. Since we need to compare the best examples of each scheme, that means that we must optimize each one before we compare them all and choose the best.

We decided to do that in parallel. The schemes we tried were all to use a function of L and R in combination with a threshold; if the function was greater than the threshold, select Q, and otherwise I. The functions were simple arithmetic combinations of L and R:

(L) (R) (L*R) (R/L)

There are obviously others; and those can be generalized in obvious ways. But perhaps they can be considered a fair sample. Remembering the observation two paragraphs above that the boundary between Q and I was approximately hyperbolic, the reader will suspect that the best function ought to be (L*R), since L*R=constant is a family of hyperbolae.

The program ran all of those schemes, represented by the different function forms, in parallel, keeping track of the costs; after adapting the thresholds to somewhere near optimum, the costs were compared and the best one chosen. The experiment used 2500 lists, the ones that were used to make figure 4.1. Using L as the function, the best threshold turned out to be 40; using R, the best threshold was .08 - .10. This is shown in the following table, using arbitrary units for the costs:

| LENGTH THRESHOLD | TOTAL COST |
|---|---|
| 10 | 6185 |
| 20 | 6182 |
| 30 | 6181 |
| 40 | 6179 |
| 50 | 6180 |
| 60 | 6187 |
| 70 | 6194 |

| RANDOMNESS THRESHOLD | TOTAL COST |
|---|---|
| .03 | 5995 |
| .04 | 5944 |
| .05 | 5944 |
| .06 | 5916 |
| .07 | 5916 |
| .08 | 5912 |
| .09 | 5912 |
| .10 | 5912 |
| .11 | 5933 |
| .12 | 5990 |

Using L*R, the best threshold was 30, and the cost was better than for either L or R by themselves.

| L*R THRESHOLD | TOTAL COST |
|---|---|
| 10 | 5957 |
| 20 | 5881 |
| 30 | 5877 |
| 40 | 5942 |
| 50 | 6066 |
| 60 | 6268 |

Using the other functions produced no usable threshold at all. The differences in performance are not enormous, it must be remarked, but they are all in the right direction.

## 6.0 Discussion

What we have tried to show here is that some conceptually very simple methods of adapting certain parameters that govern the selection of an algorithm in a computer program can produce profit for the system. Learning what is the best thing to do, and when to do it, always entails more work than merely doing what is the standard; sometimes, however, it more than pays for itself. If one is in a situation that is hard to model -- like a fickle and changing set of computer users -- the default procedure has to be to see experimentally what works best, and then to take advantage of what one finds out.

In our paradigm, sorting lists in an adaptive way, testing to see which is the best algorithm is expensive; usually more expensive than doing the task. It is as though we are slogging through mud of varying depth on submerged wooden tracks, but we do not know which is the track that is nearest the surface. To test the other tracks may require complete submersion, but it may result in finding a track that is only a couple of inches deep. So how often should one take the plunge?

Adaptation at another level is shown by the second scheme. Here we are provided with a good map of the terrain, showing the depth of the tracks. It's just that we do not know where we are unless we swim around in the mud getting bearings -- the more we swim the better the estimate.

The third scheme handles two adaptations at once. One of them is a simple tuning of a threshold, the other a simple discrete choice. The important aspect is that the second choice depends on having done the first adaptation well. It is an easy example of a hierarchy of adaptations. We use it not so much as to produce a useful program by itself, but to illustrate the kind of adaptations that must be used in the development of systems that may be hard to model or even understand.

It will be clear that a crucial role in this attack is played by the representation of the possibilities, the different functions in our case. As we have mentioned already, Winston (1976) lays much stress on that point. Behind it lie some other ideas. Before the representation can help, there must be the possibility of searching for help in the first case. L and R are merely two attributes of a situation where a program has a task to do. In some way the program seeks to use the information in R and S; and furthermore, it seeks to optimize the diagnostic functions of the observables R and S, by improving them. Typically in current system design, the possible realm of modifications is sharply restricted and tightly delineated, so as to make it less likely that bugs will arise. Systems, it is claimed, should always work in known ways with tried and true algorithms. What we are talking about here is a system that ought to be able to **notice**, for example, that small values of L should mean to use I; and from noticing, to make infe ences about good rules of behavior. Good rules of behavior -- in our case, that L can be used as an indicator or diagnostic, and that so can R -- can be combined or modified to make better ones. That process of course does not stop in a single application, but continues, guided by the enlarging set of tasks that the system is faced with. Indeed, the rules of combination and modification themselves ought to be considered as modifiable in the same way, but perhaps that is more ambitious than we are dealing with here.

The efficient functioning of the adaptations described here does not depend on having an accurate model of what sorting is or how the individual algorithms work. Rather, we claim, the program tries to learn from various kinds of experience. This is far from saying that good models, mathematical or otherwise, should be avoided. If we can get good models, we should use them. But there are many instances when it is difficult to produce and deal with accurate models of the tasks ahead; in such cases, some of the adaptive techniques shown here, or ones like them, may be useful.

That may be true, for example, in the control of a complex communications net, or in setting or tuning the scheduling algorithm of a time-sharing system. Typically such models have to assume certain kinds of random distributions in order to be mathematically tractable; and all too often those assumptions are grossly incorrect.

## 7.0 BIBLIOGRAPHY


Feigenbaum, E., and Lederberg, J., "Heuristic Programming Project," in Earnest, L., "Recent Research in Artificial Intelligence, Heuristic Programming, and Network Protocols," Stanford AI Lab Memo AIM-252, July 1974.

Knuth, D., **The Art of Computer Programming; Volume 3: Sorting and Searching**, Addison-Wesley, Reading, Mass., 1973.

Minsky, M., and Papert, S., **Perceptrons; an Introduction to Computational Geometry**, M.I.T. Press, Cambridge, Mass, 1969.

Rosenblatt, F., "The Perceptron - a Theory of Statistical Separability in Cognitive Systems," Cornell Aero. Lab., Report VG 1196, G1 & G2, 1958.

Winston, P.H., **Artificial Intelligence**, Addison-Wesley, Reading, Mass., 1977.

Wirth, N., **Algorithms + Data Structures = Programs**, Prentice-Hall, Englewood Cliffs, N.J., 1976.

# Some Observations on Problem Solving†

Hans Berliner
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213

## Abstract

In this paper, we make the case that problem solving based on immutable goals, selection of operators to bring these goals closer, and discrete logic to both select operators and evaluate outcomes is effective only in very small domains. Instead, methods using search and continuous evaluation functions do well in any sized domain as long as the evaluation functions have a certain structure. Discrete reasoning systems manipulate discrete valued entities. However, serious errors can occur when the value of a continuous variable is discretized, especially if this is done before the value is needed for final output. Because of the need to prevent this source of large errors during the evolution of problem solvers that must survive while they master their domain, we infer that the generality-specificity dimension of problem solving runs from ends-oriented to means-oriented, and from continuous to discrete. Finally, we conjecture about the structure of computing machinery for problem solvers that must evolve from general to specific.

## I. Introduction

Means-oriented problem solving requires a method of selecting a sequence of operators that may lead to a goal. This involves knowing the potential of available operators, and possibly the closeness of non-goal states to goal states. It is widely held that this type of activity is a major part of human problem solving, and that the selection of suitable operators is achieved using rule-based or pattern-based knowledge.

It is also possible to have simply an ends-oriented problem solving approach. This involves generating a set of alternatives (by generate and test procedures such as searches) and then evaluating the leaves of the test set to find the best path to pursue. Usually one attempts to generate the largest set of alternatives that can be processed with the resources available. This gives a brute-force aspect to the method; it attempts to discover the best path by investigating the maximum number of alternative paths, rather than by attempting to apply knowledge to guide the investigation into those areas that appear most promising.

The two methods can best be distinguished in that the means-oriented method must have knowledge of the potential of operators so that it can choose wisely among the available ones. This has lead (in GPS [Newell, et. al., 1960]) to the "table of differences" that gives a clue as to which operator is most likely to produce maximum progress. To date, the generation of data to guide the selection of operators has been done almost exclusively by humans (programmers). Thus, it appears unlikely that data of this type can be generated mechanically for domains of (say) $10^{12}$ states, yet humans are able to make good decisions in such large domains. Means-oriented methods and ends-oriented methods both will require knowledge of how good a current state is; in the first instance to decide which branch to pursue (as being closest to the goal) and in both instances in order to identify the goodness of leaf nodes that are reached.

Evaluation can be thought of as being done by a function that assigns a scalar value to a state, thus making it possible to compare its goodness to that of another state. In small domains this process may be little more than the identification of goal-states, or the identification of states that have some salient feature that must elevate it above any state not having such a feature. This dominance type of reasoning is usually quite adequate in small domains, thus giving evaluation a discrete character; yes/no or a sorting into a small number of equivalence classes. However, in larger domains the full power of a polynomial function, with its ability to trade-off the value of one term of the polynomial against the value of another, may be required. At its full potency, the polynomial can take on a (more or less) continuous set of values, and should (if totally effective) be able to correctly order all states in the domain with respect to nearness to goal-states. In practice, such effectiveness is not achievable in interesting domains, so it is desirable that the ordering, if not totally effective, at least not produce large decision errors (such as sending the solver off in the opposite direction, or leaving it stranded on a hill-top). We shall show that the structure of the evaluation polynomial has a great deal to do with its effectiveness.

†This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The selecting of effective operators at a node, apart from being governed by decision rules, could also be done by evaluating the state that each available operator produces and selecting the best. It should be noted that, while there is a sequential flavor to a reasoning process that moves from one premise to the next to achieve its aims, the evaluation polynomial is essentially a parallel construction, with each term independent of all others. Thus reasoning, discrete, and sequential appear to go together, while judgement (evaluation), continuous, and parallel go together also.

## II. Two Examples

Consider the problem of mating with a King and a Rook versus King (KRK) at chess which is a medium size problem with a state-space of about $10^5$. All instruction books for humans will indicate that the correct *procedure* (thus *means-oriented*) is to use the rook to build a fence around the black king (see lower left of Figure 1), and gradually constrict the fence until the mate is there. It is rather interesting that this advice suffices for humans. Clearly, they have enough structure to interpret these instructions and produce the correct effect. I have never heard any beginner complain about the adequacy of these instructions, although I remember being temporarily at a loss the first time I tried the exercise because fence constriction, taken to the ultimate, results in stalemate. Thus, a last-minute change of strategy is required.

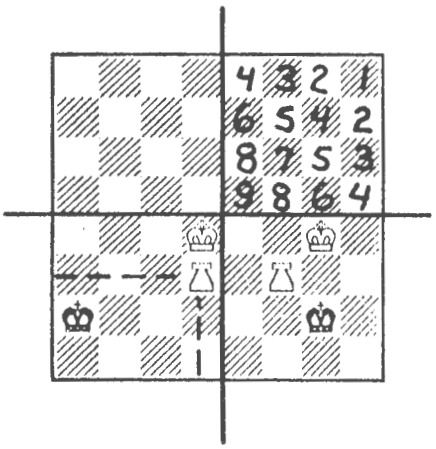



Figure 1

However, when one attempts to implement the same instructions for a computer program, some very vexing problems occur. They deal with exactly how to go about constricting the fence, since at times no constricting move is possible without letting the opposing king out (lower right of Figure 1). Also, there is the problem of not intersecting the fence by placing one's own king on the fence line and thus allowing the opposing king to cross (for instance if K-B3 in lower left, then K-R6 and black's king has escaped). These problems are so prolific that one author [Zuidema,1975] has complained that if such a simple problem be that difficult to program, then chess itself must be impossible. If one wishes to program chess using only the means-oriented (rule based) approach, then Zuidema is probably right in that no set of humans will be able to write all the required rules.

Actually, the ends-oriented approach for doing KRK had already succeeded several years earlier, although with a great deal of structured knowledge together with very small searches [Huberman, 1968]. But the real power of evaluation functions when combined with search was demonstrated with great simplicity and elegance as follows [Atkin, et. al., 1971]: Consider the upper right of Figure 1. Here a gradient exists from the center to the corner. Let the major term in the evaluation function be "how decentralized the black king is". Since the same evaluation function is used by both sides, Black will resist being decentralized. Thus, it is sufficient for White to choose the sequence of moves that decentralizes Black the most until the task is completed. This measure would be sufficient for terminal nodes of a 9-ply search. For shallower searches, a subsidary term which values keeping White's king near the Black one, and a still less significant term that values keeping White's rook near the White king, allow the mate to be performed by a 3-ply search. This search need only use the evaluation function, know the value of material (so as not to lose the rook) and the rules of chess, so as to mate and not stalemate. The resulting program could be written and debugged by a second year undergraduate in about 5 or 6 hours.

Finally, the same problem is capable of ultimate solution. A data base can be created for all possible positions of KRK. Then working backwards from those positions that are mates, one can assign a number to all other positions. That number represents the minimum number of moves that are required to produce a mate. Positions that are draws (stalemate or lost rook), will be assigned a value of infinity. Using this data base it will always be possible to produce the shortest mate in any situation, by merely generating all legal moves and selecting the one that moves to the position of lowest value. This task has actually been performed by a number of researchers, first by M. R. B. Clarke [Clarke, 1977], and has produced the first (though trivial) computer-produced chess knowledge. Clarke showed that it is possible to mate in at most 16 moves from the most difficult position, whereas it always had been thought to require 17.

That there is a trade-off between the amount of knowledge and the amount of search is very clearly shown above. A data base of $10^5$ suffices to produce optimal play as does a search to 31 ply (intractable unless a dynamic programming approach is used to identify identical nodes and turn the tree into a graph). The most desirable solution though, for problems of this level of complexity is a heuristic one, in which only a satisficing, rather than an optimal solution is obtained. A shallow, ends-oriented search serves well here. A simple construction (the gradient) puts a key measure onto the evaluation process. At certain depths of search this suffices for successful performance of the task. At

shallower depths of search some small amounts of additional knowledge are required. To select an adequate move, without search or a complete data base, requires large amounts of carefully structured knowledge. The optimum trade off between search and knowledge in the above example appears to be in the area of a 5-ply search (1 second duration) with a 2-term polynomial. It is interesting to note that one can characterize this problem (as is possible of all mates of a lone king) as simply a decentralization problem. This characterization is simple, precise and very useful. Yet humans characterize it differently, possibly because of the effect of culture on the primitives available for perceiving the problem.

The above is a typical medium sized problem as judged by the size of its state space. Let us now examine a small problem: i.e. the Monkeys & Bananas (M&B) problem [McCarthy, 1964] with at most a couple of hundred states. As usually stated, the M&B problem has a few operators: move monkey (X), move box (X), climb box, and reach (x), each of which may have some applicable pre-condition and effect some tranformation on the state. Then depending on what your favorite paradigm is, you can solve the problem using GPS, predicate calculus, etc. However, these formulations all require a number of restrictions on the real problem (to make it tractable), together with machinery specifically designed to make the solution proceed in the necessary direction. Even then, solving the problem produces a formidable challenge to the solving system.

Let us now pose M&B as a search problem. We can complicate the problem to the point where many techniques would find it next to impossible to solve by increasing the number and scope of the operators. The operator for moving the box produces movements of exactly one foot in one of the 8 compass directions. The same is true for moving the monkey. Also, allow the monkey to climb down from the box as well as climbing up. In addition, allow the monkey to throw the box against the cage (making it unclimbable), to tear a lath off it (making it unclimbable), and to reach in each of the eight compass directions. A goal state is one in which the monkey touches the bananas. In this problem statement, quite a few operators may be applicable at any one time. The state description specifies the 3-dimensional location of monkey, box, and bananas, and the condition of the box.

Now, as a search problem, we would be willing to allow (say) a 4-ply search and evaluation of terminal nodes. Our evaluation function will value primarily the closeness of the monkey to the box, secondarily the closeness of the box to the floor and to the bananas, and thirdly the closeness of the monkey's hands to the bananas. It is rather clear that such a paradigm will succeed with extreme efficiency and ease in discovering a very good solution. The monkey will approach the box to satisfy the primary term, push it under the bananas to satisfy the secondary term, and climb the box and reach for the bananas to satisfy the tertiary term. I realize, of course, that M&B is only used as a pedagogical tool to demonstrate problem solving paradigms. In fact, that is exactly what I am using it for.

## III. Problem Solving Performance in Large Domains

Let us now examine the experience of various problem solving methods in large domains. There are a few efforts to apply means-oriented methods to checkers [Samuel, 1959 & 1969]; and chess [Baylor & Simon, 1966; Berliner, 1974; Pitrat, 1977; and Wilkins, 1979]. Samuel's program was a marvel for its time, but has more recently been soundly trounced by a full-width searching (ends-oriented) program with much less knowledge [Samuel, 1977]. The Baylor-Simon MATER program worked only in very restricted situations. Thus this was more a case of exposing the power of a useful move selection heuristic (the move that allows the fewest replies) than an attempt to cover the domain of mating combinations, not to speak of the realm of combinations in chess. The Berliner program did reasonably well at doing chess combinations, but was inept when no combinations were to be found or when its knowledge was not quite up to finding them. Pitrat introduced the notion of plans to select moves that, deeper in the search, were compatible with an initial goal. He also introduced methods for patching a plan when it ran into difficulties, but his approach relied heavily on brute force searching and very simple plans. The Wilkins program considerably improved on the last two efforts above with knowledge comparable to the Berliner program and plans of great sophistication that effectively controlled the plausible actions deeper in the search. This program was able to attain very high solution rates on chess combination positions, once its knowledge base had been built up to an appropriate level.

In all these efforts one paramount fact has intruded itself upon us: a very small change in the problem environment can make a large difference in what is the correct action, and what, therefore, the problem solver may or may not be able to do. Thus, the way means-oriented programs are improved is by the writing of ever more exception rules. In the end, the search is supposed to catch those exceptions that were not explicitly programmed in.

However, there is a great deal more to chess than executing combinations. This has been shown dramatically by the Northwestern University chess group, whose program CHESS 3.0 (and up) has been the perennial winner of almost all important computer chess events. While means-oriented programs wallow in trying to solve relatively small sub-domains of chess, CHESS 4.6 (and up) [Slate & Atkin, (1977)] has in the last 3 years moved up to challenge good human players, some of whom it has defeated. This program relies heavily on a full-width search with iterative deepening† which is made more

†A full-width search at each node looks at all alternatives in the tree that have not been logically eliminated by alpha-beta pruning. Iterative deepening involves doing first an N ply search, then an N+1 ply, etc., until the allocated time resources have been expended.

efficient by the installation of a hash table that:

1) Guides the search into the promising sub-trees discovered by the previous iteration, and

2) Terminates the search at positions that are identical to those already searched in the current iteration.

Ken Thompson of Bell Telephone Laboratories has shown that organizing the above method for parallel computation and using special purpose hardware produces further significant speed-ups. Thus, in chess and checkers the hand-writing is clearly on the wall. Brute force searching with relatively little knowledge will soon be able to beat almost all the players in the world. Whether knowledge oriented programs will be required for the World Champion level in chess is a moot point; however, in this writer's opinion the programs will play with so much greater consistency, that with just small amounts of additional knowledge, they will perservere to the World title.

The situation is quite different in GO, however, where the magnitude of the task would appear to make the use of ends-oriented methods quite difficult because of the large number of alternatives. In fact, no one has tried such methods, and the best program to date [Reitman and Wilcox, 1979] makes heavy use of specially designed GO constructs to guide its play. However, its play in this most difficult of games is far from being able to give even intermediate players a decent game.

At backgammon, a program that uses no search but relies solely on evaluation of all possible moves emanating from the current position [Berliner, 1980], has recently defeated the reigning World Champion by the lop-sided score of 7-1; a result that must be somewhat discounted due to the stochastic nature of the game. Again, this was the result of an ends-oriented approach, that we describe to some degree in the next section.

Apart from games, experience with speech understanding systems has shown that discrete reasoning systems do not do as well as a brute force searching system using a fraction of the knowledge [Lowerre & Reddy, 1979].

It should be clear from the above that, if at all possible, ends-oriented methods should be employed. They are easier to implement, succeed better, and may be the only realistic way in certain domains. Further, the methods have been shown to be applicable to many domains that were thought to be too complex to ever be subjugated by brute-force searching.

We hope the above has made clear our first thesis: that means-oriented problem solving has proven robust only in small domains. Some shallow searching plus some simple terminal evaluation has in an overwhelming number of cases been shown to be superior to the business of solving problems by operator selection and reasoning. This is definitely true for machine oriented problem solving, and the evidence is so strong that one wonders how living organisms get along without using this, if, in fact, they do.

## IV. The Structure of Evaluation Functions

The principal usefulness of evaluation functions is for guiding a problem solving process that is unlikely to reach a domain defined goal (i.e. a complete solution) during its present probe, and must thus settle for a step in what is considered to be the right direction toward a solution. A number of reasons now appear to favor using evaluation functions where possible over the reasoning methods that have been considered fundamental in the past:

1) It is possible to simultaneously pursue several goals with this method. Each term (or a small set of terms) in the polynomial could be considered a possible sub-goal to be pursued. Thus the degree to which each has been achieved may be ascertained. This is extremely difficult to do under reasoning paradigms, as one goal will be paramount in such procedures. Such a goal, in turn, determines the valid sub-goals, and all others are ignored. Such methods will prefer success at a primary goal to success at a number of secondary goals that may, in fact, be superior. Interactions between sub-goals may be taken care of in the evaluation function by the use of non-linear terms.

2) Two major problems with evaluation functions have been that they were thought to be lacking in context sensitivity, and it was possible for a hill-climbing process using such evaluation functions to get stuck on a sub-optimal hill and not be able to get off. However, in the pursuit of goals and sub-goals, proper construction of the evaluation function will produce smooth transitions from one state to another, even if the first state represents a major goal that has just been achieved, and the focus must now shift to a new goal. Below, we demonstrate how to construct evaluation functions properly.

The essential DO's of constructing evaluation functions are embodied in my SNAC method that was used in the backgammon program that beat the World Champion last year [Berliner, 1980]. SNAC stands for Smoothness, Non-linearity, and Application Coefficients.

Non-linearity is extremely important for expert performance. A constant coefficient can at best portray the *average* usefulness of the term associated with it. There will be times when this average value will be at considerable variance with what expert judgement will consider correct, and this is where systems using linear functions will fail. Non-linear functions can produce the necessary context by combining the action of several variables into one term. However, the key to using non-linear functions is smoothness. This is where Samuel made a serious methodological error when he found that his non-linear functions did not perform better than his earlier linear ones [Samuel, 1969]. Smoothness relates to the rate of change of a function for adjacent parts of the domain. Samuel, for several of his variables, subdivided their natural range into a compressed range, so that the variable could only take on a few values and thus the Signature Table would be smaller. However, this was a fundamental error as can be seen in Figure 2.

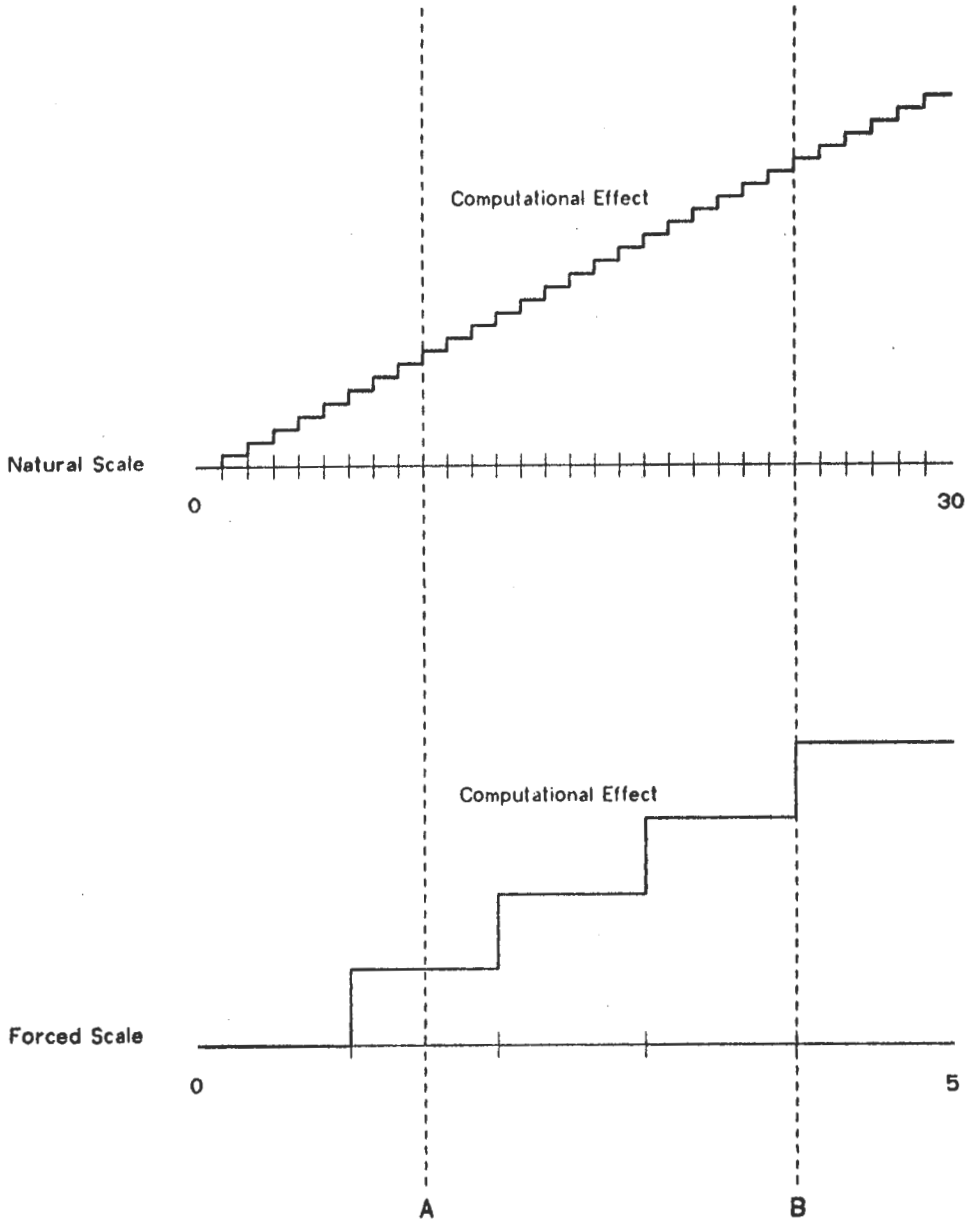



Figure 2

If a variable has a value near vertical line A, then in both the lower (large grain) and upper (smooth) situations, a small change in the value of the abscissa will produce only a small change in the value of the ordinate. However, near vertical line B the situation is quite different. Here, for the lower situation, a small change in the value of the abscissa can produce a very large change in the value of the ordinate. Such a construction will provide opportunities for a program to manipulate the value of the ordinate to an extent unwarranted by its actual utility, and this may cause the program to make serious errors.

This type of behavior can occur whenever there are sharp boundaries in the evaluation space. Assume a chess program has a different method for evaluating middle-game situations than it does for evaluating end-game situations. Experts agree that such disparate types of positions should be evaluated differently. Further, assume such a program has a middle-game position that it likes, but this position would receive a poor evaluation if seen as an end-game. If swapping material would cause the position to be evaluated as an end-game, then the program would go to great lengths to avoid swaps. This could well cause it to encounter severe

and unnecessary problems in the play. The converse of this problem also occurs: the program hurries into the end-game because the center control situation is unfavorable in the middle game.

Smoothness in functions is the answer to this problem. There is a slow metamorphosis of middle-game to end-game, and during this phase the values of both phases must be recognized, although the middle-game is waning and the end-game waxing. Any attempt to draw a sharp line between these is doomed to failure because it will result in occasional unwarranted attempts to stay on one side of the boundary or cross it too quickly.

The key to accomplishing smooth transitions is Application Coefficients. An application coefficient is a variable that measures something global, yet varies very slowly in the current context. It multiplies a term in a polynomial, thus providing context about the importance of the term under current conditions. We have investigated a number of domains and found good application coefficients in all of them. Their character is that they measure some trend or change of phase. Because they vary slowly and smoothly, the program will not be trying to manipulate them over a significant range (as by deliberately staying in the middle-game because it has good control of the center, and this is not valuable in the end-game). For chess, material on the board is a good application coefficient, and this will produce a smooth metamorphosis between phases and there will be no boundaries near which catastrophes can occur.

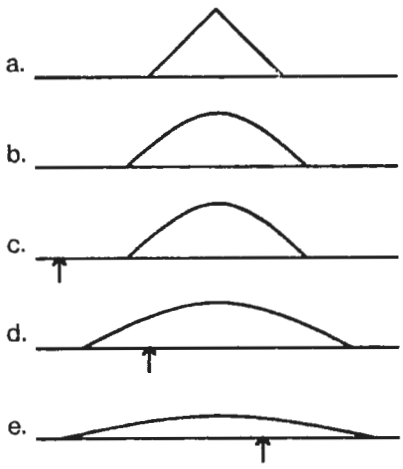



Figure 3

Application coefficients can also prevent the previously mentioned problem of a hill-climbing program getting stuck on a sub-optimal hill. Figure 3 shows the problem. With linear polynomial evaluation functions, the hills in the evaluation surface will have pointed peaks and this will make it quite likely to get stuck on such a hill (3a). With non-linear functions, the peak is less pronounced so that it may be easier to descend a once-climbed hill, if some other high ground is in view of the searching process (3b). However, with application coefficients it is possible to change the contour of the hill even as it is being climbed. This is shown in 3c through 3e; the arrow showing the proximity of the current state to the hill. At a distance, the hill looks as in 3c. This allows the height of the hill to be compared to that of other landscape features that may be achievable. However, as the hill is climbed, it begins to flatten (3d), making the achievement of the summit less desirable (since we are almost there anyway), and resulting in the program looking for the next set of goals before even fully achieving the current set. As it sets out for the next goal, the hill flattens still further (3e). This flattening is controlled by application coefficients that detect the degree of progress in achieving the goal, and reduce its importance as it comes closer to being achieved. This paradigm recalls the situation in which a football player begins to run with the ball before he has caught it. The point is: if the controlling human processes solved problems sequentially rather than in parallel, such behavior would be unlikely to occur.

Thus application coefficients can change the program's view of what it should be doing, even as it is doing it. For instance, in my backgammon program one of the major goals is to blockade some of the opponent's men. However, if such a maneuver succeeds, the blockade must eventually be lifted in order to bring one's own men into the homeboard to proceed with the win. In order to have the program understand the desirability of the blockading goal, there are application coefficients that gauge the overall situation and raise or lower the desirability of blockading based upon global considerations. Such constructions can be tuned to give truly amazing performance: perceiving (as it appears) when blockading is appropriate and when it is not.

That is not to say that a domain should never be partitioned into sub-domains for evaluation purposes. Sometimes, that is the only sensible thing to do, but it must be done judiciously. For instance, in backgammon there will come a time in the game where the two sides are no longer in contact and both are racing for home with no further impeding of each other. In such a phase it is senseless to consider such features as blockading potential, board control, etc. Since the coefficients of such terms would be zero during the running game, evaluations generated for such running game situations will differ considerably in magnitude from those generated for competing non-running game positions. Yet it may be necessary to choose between such positions. What is needed in such a situation is a common measure along which both types of positions may be evaluated. This is attained by computing the win probability for each type of position. The best of each position type can first be chosen using the evaluation function appropriate to each sub-domain. Then the best position in each sub-domain can be compared to select the best over-all course of action. New sub-domains should only be created when there is a clear basis for doing this, as the number of potential comparisons grows with the square of the number of sub-domains, and each comparison is a potential source of decision error.

The use of SNAC functions in my backgammon program turned it from a mediocre competitor to an expert level program, with only a small influx in additional backgammon information, as is documented in [Berliner, 1980].

## V. Why Discrete Systems Fail in Large Domains

To here, I have tried to make two points:

1) That the combination of search and good evaluation functions produces a very fine problem solver for many domains, and

2) That the evaluation functions must be carefully constructed and are more powerful when non-linear.

Now it is time to take cognizance of the rather apparent degradation that takes place when problem solvers relying on boolean decision making are applied to large domains. The degradation takes place in the process of goal selection, the process of operator selection, and the process of evaluating closeness to goals in non-terminal nodes of the domain. From the variety of evidence presented, we consider it apparent that this is not the fault of researchers or lack of effort, but rather of the nature of the problem and the method. It appears that the idea of applying boolean decision rules to a large domain just will not work unless the domain is quite regular (as is mechanics, where a few principles have ultimately been shown to account for all macro behavior). Thus, as the exponential explosion prevents any attempt to produce satisfactory decision functions based on predicates (too many predicates required), any attempt to subdivide the domain without true basis in fact succumbs to the problems we have described in Section IV.

Let us try to determine the reasons why it is so difficult to improve a discrete problem solver. The effectiveness of a problem solver is measured by the nearness of the system proposed solution to the best or an adequate solution. If an algorithm for a particular domain is not known, then it is likely that effectiveness will be achieved gradually through increases in sensitivity to what a correct solution is. By sensitivity, we mean the number of states in the domain that are now ordered correctly, ignoring how far off misordered states are.

Given that the effectiveness of a problem solver is to be improved, sensitivity can be increased by having two states, that formerly had the same value, no longer have the same value. If two such states have similar state descriptions, it is possible to think of them as being somewhat adjacent in some mapping of the domain onto a multi-dimensional surface. To produce the new sensitivity, it is possible to place a partition between the two states so that states on each side of the partition will now be treated differently. However, this will result in many other adjacent states now being on one side or the other of the new partition, thus possibly altering their treatment too.

As earlier sections of this paper have shown, there is a definite risk associated with increasing the sensitivity of a discrete problem solver. The risk stems from the fact that introducing a partition, while it may improve the sensitivity of the problem solver, may also result in radically misordering certain states. Partitioning will only work properly if:

1) There really is a discrete difference between identifiable sets of states in this part of the domain, and

2) The partition is drawn absolutely correctly so as to not have any state on the wrong side of the partition.

Apart from partitioning domains, increases in sensitivity can also be achieved by creating a smooth gradient between the two states that are now to be treated differently. This will affect the values of other adjacent states, but not in such a severe manner as to cause misclassifications.

A critical observation here is that drawing a partition, irrevocably fixes the value of some variable at some discrete interval (as in Figure 2, lower scale). If such a variable is quasi-continuous, and if its value is to be used later for other computations, then it is certainly preferable to postpone the discretizing process as long as possible and retain its value in quasi-continuous form.

Among large systems, MYCIN-like systems are considered to exercise their expertise very well. These systems apparently avoid the partitioning problem in large domains by the use of probabilistic indicators [Shortliffe & Buchanan, 1975]. Because of this, they can hardly be considered to reason in the boolean manner, but rather one gets the flavor of evaluation with summation of likelihoods.

## VI. Models and Sensitivity

The effectiveness of a problem solver depends on how well its domain is being modelled. Most domains can be modelled at many levels of detail. Consider that the morning weather forecast predicts a 40% chance of showers, when it could conceivably produce a cumulative precipitation curve over time for that day for each acre in the metropolitan area. If the domain is discrete and the model also, then a correctly formulated discrete model can be very effective. This is the case in most small domains and in domains that are "regularized", e.g. the game of NIM for which a simple rule can find the winning move even though the size of the domain is infinite. If the nature of the domain is not completely understood, opting for a quasi-continuous model appears preferable. This applies both to operator selection in a problem solver that applies knowledge to this process (because misordering operators can also have a very deleterious effect), and to evaluation.

Based on the action requirements and the accuracy and availability of input data, a model is chosen. It is desired to have that model function near the top of its effectiveness. When a given model does not perform near that level, then either the input data are insufficient, or the model is insensitive to certain things. The selection and improvement of a model appears to governed by the following principles:

1) For each model of each domain there is an optimum sensitivity. If the model utilizes a greater degree of sensitivity, it wastes computational resourses; if it uses lesser sensitivity, it will fail to "understand" or react to certain things. However,

2) Each increase in sensitivity in the problem solver is accompanied by an increase in risk of incorrect interpretation or action.

Consider the chess middle-game, end-game situation mentioned in Section IV. In a particular implementation, a program may consider that trying to control the center in the end-game is important, even though it is really not. This would produce occasional ordering errors because the program would value center control in the end-game more than is warranted by reality. Thus, it would occasionally fail to achieve a more worthwhile goal.

Now, assume the space is partitioned so that middle-game and end-game are no longer on the same side of the partition, and control of the center is valued only in the middle-game. This will result in better ordering of most end-game situations, but will occasionally cause serious problems akin to myopia when transitions from middle-game to end-game are involved. This is the risk involved, and as we have shown earlier, it can produce serious problems that would render the value of the increased sensitivity questionable.

Another way of looking at the problem is the following: Assume a system is capable of only two responses and a partition in the domain determines which response is given. The naive probability of response error is 0.5. However, assume an ordering of the states of the domain exists such that all states above a certain state in the ordering are on one side of the partition and all the remaining states are on the other side. Under such conditions, errors are much more likely to be made in the vicinity of the partition than elsewhere.

If a variable is to be used to produce a *final* boolean decision, then there is no difference between using a partition and using some distance function of the place in the ordering to produce the answer. However, if this is an *intermediate* result that may later be combined with other data, then there is a great deal to be gained by retaining some fuzzy representation of the result; i.e. the distance from the partition. Thus, it is frequently more useful to know that an event occurred at sunset, than that it occurred during daytime. Consider the importance of distance from high noon when evaluating the ability of an observer to see an event accurately. When it may be important to carry forward some of the properties of the original measurement, a quasi-continuous measure serves better. In such cases, where a gradient measures the property in question, the likelihood of error would be equally distributed throughout the domain. Since under such conditions, general remedies exist for reducing the error in any state, such a paradigm would seem preferable, when the value may be interim or when partitioning cannot be justified by the intrinsic properites of the domain.

## VII. The Evolution of Problem Solving Systems

There is no doubt that a highly discretized problem solving structure is the most effective one possible when such a model is applicable and the data it requires are available. After all, that is what science is all about. However, if a model produces some errorful responses then care must be taken in achieving discretization. In a sequential problem solver, a number of small errors is preferable to one large error. Research in game playing programs has shown again and again that such a system is no better than its weakest link. Further, the very ability to discriminate the condition of small error as against no error at all, is the hallmark of the expert. Each presently surviving organism considers itself to have adapted adequately. However, an expert organism of a particular species may be able to distinguish errorful acts in a somewhat inferior (but currently surviving and self-confident) specimen of its species. This, again, supports the view that small errors are tolerable, and are done away with gradually over time.

Thus, for large domains (and in real life almost everything is large) problem solvers must first and foremost be able to produce reasonable decisions (ones that are not too far off the mark). To do this, fuzzy methods are much more satisfactory than those that reason. Because highly discretized problem solving is so difficult to achieve, it is almost certainly preceded by other less exact decision methods in the ontogeny of any evolving problem solver.

Thus, it would seem that starting with smooth, continuous functions and gradually discretizing them would be a good strategy for achieving increased sensitivity. As increased sensitivity is achieved over time, most of the previously effective problem solver must still be in place. Thus, there will be a mixed bag of problem solving techniques, ranging from the use of continuous functions to discrete logic. In such an environment, it appears extremely likely that many intermediate variables will retain their original fuzzy character because higher level constructs are presently made from them. These notions would apply equally well to animate and inanimate problem solving systems.

Assuming the above ideas are valid, there must be a way for the problem solving systems of living organisms to evolve in this direction, both during the life of the organism and the life of the species. One possible solution to this problem is the variable coefficient. Such coefficients, as they vary between 0.0 and 1.0, have several known uses:

1) As a characteristic function in fuzzy set theory, indicating to what extent the element to which it applies is a member of the set.

2) As an application coefficient in SNAC that controls applicability of a concept.

3) For controlling truth value in certain belief systems.

When such a value has gravitated as close to an extremal value as can be detected by the system, then we no longer have smooth variation between the limits, but a

boolean entity. We conjecture that this paradigm accounts for the behavior of Piaget's pre-conservation children, where the physical extent of a set of objects is taken to be the best criterion of the *amount* of the set, until it is learned that conservation (when applicable) dominates "extent".

Thus, an essential element of any evolving problem solver would appear to be computing elements capable of graded response.† Beyond that, we do not want to propose here that human problem solvers use full-width shallow searches and evaluation procedures as those that have been so successful in computer programs. However, we do consider it likely that processes based on constraint satisfaction (as first implemented in Waltz's vision system [Mackworth, 1977]) or tightly controlled knowledge directed searches (as in the B* tree search algorithm, [Berliner, 1979]) are developed to play the role that brute-force searching does in the previously references programs. Both methods could be used to screen out obvious misfits in the solution process, in the first case through a low level combinatorial analysis and in the second case through estimation of the limits of usefulness of each alternative. Evaluation would be done using SNAC-like methods at each level of the solution process.

Finally, let me briefly address an issue that may be brought up by some. The theory of computation decrees that any continuous system can be simulated to any desired degree of fidelity by a Von Neumann machine. That is not the issue here. The issue is one of complexity. Certain computing elements perform certain tasks more efficiently than others, and in this case the required elements are such that they can provide graded acceptance of signals, and graded response. To use boolean circuits to provide the response required by the complex domains that are encountered every day would appear to be so difficult that (we hold) even evolution would not have been able to build a satisfactory system out of such components. The real question is how did a system that has graded response come to evolve into a system that can manipulate symbolic entities. It may be that, in our desire to simulate the highest levels of human behavior, we have been overlooking the fundamental information processing that is required to produce the variables that support such performance.

†We are well aware of previous work in the field of perceptrons and the demonstration of the limitations of *linear* perceptrons in [Minsky and Papert, 1969]. However, we are here proposing a special class of *non-linear* perceptron.

## BIBLIOGRAPHY


Atkin, L. R., Gorlen, K., and Slate, D. (1971), "Chess 3.0 - An Experiment in Heuristic Programming", (Northwestern University), *Unpublished*, 1971.

Baylor, G. W., and Simon, H. A. (1966), "A Chess Mating Combinations Program," *Proceedings of AFIPS 1966, SJCC*, Vol. 28, 1966, pp. 431-447.

Berliner, H. J. (1974), *Chess as Problem Solving: The Development of a Tactics Analyzer*, Ph. D. Dissertation, Computer Science Department, Carnegie-Mellon University, March, 1974.

Berliner, H. (1979), "The B* Tree Search Algorithm: A Best-First Proof Procedure", *Artificial Intelligence*, Vol. 12, No. 1, 1979, pp. 23-40.

Berliner, H. J. (1980), "Backgammon Computer Program beats World Champion", *Artificial Intelligence*, Vol. 14, No. 1, 1980.

Clarke, M. R. B. (1977), "Appendix. King and Rook against King" in *Advances in Computer Chess 1*, M. R. B. Clarke (Ed)., pp. 116-118, Edinburgh University Press, 1977.

Huberman, B. J. (1968), "A Program to Play Chess Endgames", Technical Report No. CS 106, Computer Science Dept., Stanford University, 1968.

Lowerre, B., & Reddy, R. (1979), "The Harpy Speech Understanding System", in *Trends in Speech Recognition*, W. A. Lea (Ed.), Prentice-Hall, 1980.

Mackworth, A. K. (1977), "Consistency in Networks of Relations", *Artificial Intelligence*, Vol. 8, No. 1, pp. 99-118, 1977.

McCarthy, J. (1964), "A Tough Nut for Proof Procedures", *Stanford University*, AI Project Memo, No. 16, 1964.

Minsky, M., & Papert, S. (1969), *Perceptrons*, The MIT Press, 1969.

Newell, A., Shaw, J. C., and Simon, H. A. (1960), Report on a General Problem Solving Program for a Computer, *Information Processing: Proceedings of International Conference on Information Processing*, pp. 256-264, UNESCO, Paris, 1960.

Pitrat, J. (1977), "A Chess Combinations Program Which Uses Plans", *Artificial Intelligence*, Vol. 8, No. 3, 1977.

Reitman, W., and Wilcox, B. (1979), "The Structure and Performance of the INTERIM.2 Go Program", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, 1979, pp. 711-719.

Samuel, A. L. (1959), "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, Vol. 3, No. 3, 1959, pp. 210-229.

Samuel, A. L. (1969), "Some Studies in Machine Learning Using the Game of Checkers, II - Recent Progress", *IBM Journal of Research and Development*, Nov. 1967, pp. 601-617.

Samuel, A. (1977), "The Duke vs. Stanford Computer to Computer Checker Match", *SIGART Newsletter*, No. 63, June, 1977, pp. 86-87, ACM Publications.

Shortliffe, E. H., & Buchanan, B. G. (1975), " A Model of Inexact Reasoning in Medicine", *Mathematical Biosciences*, Vol. 23, pp. 351-379, 1975.

Slate, D. J., & Atkin, L. R. (1977), "CHESS 4.5 -- The Northwestern University Chess Program", in *Chess Skill in Man and Machine*, P. Frey (Ed.), Springer Verlag, 1977.

Wilkins, D. (1979), "Using Plans in Chess", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, 1979, pp. 960-967.

Zuidema, C. (1975), "Chess, How to Program the Exceptions", *Afdeling Informatica*, Amsterdam, 1975.

## AUTHOR INDEX