SOCIÉTÉ CANADIENNE DES ÉTUDES D'INTELLIGENCE PAR ORDINATEUR

Proceedings of the
SECOND NATIONAL CONFERENCE

Compte-rendu de la
DEUXIÈME BIENNALE

University of Toronto
TORONTO, ONTARIO
July 19-21, 1978
19-21 juillet, 1978

PROCEEDINGS OF THE SECOND NATIONAL CONFERENCE OF THE CANADIAN SOCIETY FOR COMPUTATIONAL STUDIES OF INTELLIGENCE

COMPTE-RENDU DE LA DEUXIEME BIENNALE DE LA SOCIÉTÉ CANADIENNE DES ÉTUDES D'INTELLIGENCE PAR ORDINATEUR

presented in cooperation with
présenté en coopération avec

ASSOCIATION FOR COMPUTING MACHINERY
SPECIAL INTEREST GROUP ON ARTIFICIAL INTELLIGENCE

UNIVERSITY OF TORONTO
TORONTO, ONTARIO

July 19-21, 1978
19-21 juillet, 1978

© 1978

by Canadian Society for Computational Studies of Intelligence
par Société Canadienne des Etudes d'Intelligence par Ordinateur

Copies of these Proceedings can be obtained
at a cost of $12 per copy from

Professor C. Raymond Perrault
Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 1A7

PREFACE

At the first conference of the Canadian Society for Computational Studies of Intelligence held in Vancouver in 1976, it was decided that the organization would hold a national meeting every other year. The objectives of the conferences are to foster the development of Artificial Intelligence and Cognitive Science in Canada and to provide a forum for the North American AI Community during years when no International Joint Conference is held.

Many people have helped to make the Second National Conference possible. Ted Elcock was chairman of the Programme Committee which also included J.S. Brown, Gordon McCalla, Jerry Hobbs, Steven Zucker, and Ray Reiter. Alan Mackworth kindly agreed to give the keynote address. The Department of Computer Science at the University of Toronto and its chairman, J.N.P. Hume, provided us with a base of operations. Pam Linnemann, Teresa Miao and Winnie Green got us over administrative hurdles. Gordon McCalla prepared the proceedings. The cover design is by Mario Carvajal through the courtesy of Gage Educational Publishing Ltd. Alex Borgida and the Department of Computer Science students J. Allen, R. Cohen, M. Horrigan, H. Levesque, C. Reason, P. Schneider, J. Tsotsos and H. Wong contributed ideas and peoplepower without which this conference could never have been held. Many warmest thanks to all of them.

C. Raymond Perrault
General Chairman

PRÉFACE

La Société Canadienne des Etudes d'Intelligence par Ordinateur décida à son premier congrès qui eut lieu a Vancouver en 1976 d'organiser de telles rencontres tous les deux ans. Le but de ces réunions est de promouvoir l'épanouissement de la recherche en l'Intelligence Artificielle et en Sciences Cognitives au Canada, ainsi que de fournir à cette communauté un forum nord-américain chaque année ou n'aurait pas lieu une conférence internationale.

La tenue de ce deuxième congrès aurait été impossible sans la coopération d'un grand nombre de personnes. Ted Elcock a coordonné les efforts du comité responsable du programme. J.S. Brown, Gordon McCalla, Jerry Hobbs, Steven Zucker, et Ray Reiter en ont aussi fait partie. Alan Mackworth a gentiment consenti à servir de conférencier invité. Le département d'informatique de l'Université de Toronto par la voix de son directeur, J.N.P. Hume, nous a fourni une base d'opérations. Pam Linnemann, Teresa Miao et Winnie Green nous ont aidés surmonter une kyrielle d'obstacles administratifs. Gordon McCalla est responsable de la préparation de ce volume. Le graphique de la couverture est de Mario Carvajal, gracieuseté de Gage Educational Publishing Ltd. Alex Borgida ainsi que les étudiants J. Allen, R. Cohen, M. Horrigan, H. Lévesque, C. Reason, P. Schneider, J. Tsotsos, et H. Wong ont contribué leur imagination, leur temps, et leur labeur. Je les en remercie tous.

C. Raymond Perrault
Directeur

Officers of the CSCSI/SCEIO

President

Professor Richard S. Rosenberg
Department of Computer Science
University of British Columbia
Vancouver, British Columbia

Vice-President

Professor Zenon Pylyshyn
Department of Computer Science
University of Western Ontario
London, Ontario

Treasurer

Professor Wayne Davis
Department of Computing Science
University of Alberta
Edmonton, Alberta

Secretary

Professor John Mylopoulos
Department of Computer Science
University of Toronto
Toronto, Ontario

Officers of Second National Conference of the CSCSI/SCEIO

General Chairman

Professor C. Raymond Perrault
Department of Computer Science
University of Toronto
Toronto, Ontario

Programme Chairman

Professor E.W. Elcock
Department of Computer Science
University of Western Ontario
London, Ontario

Editor of Proceedings

Professor G.I. McCalla
Department of Computer Science
University of Toronto
Toronto, Ontario

Programme Committee

Ted Elcock (Western)
John Seely Brown (Xerox PARC)
Jerry Hobbs (SRI)
Gord McCalla (Toronto)
Ray Reiter (UBC)
Steve Zucker (McGill)

CONTENTS

Problem Solving I (Wed., 9:00)

Sleuth: An Intelligent Noticer
S. Rosenberg 1

A Test-bed for Developing Support Systems for Information Analysis
J. Stansfield 11

Locating the Source of Unification Failure
P.T. Cox 20

An Analysis of Theorem Proving by Covering Expressions
L.J. Henschen, W.M. Evangelist . . 30

Natural Language I (Wed., 10:40)

A Simultaneously Procedural and Declarative Data Structure and Its Use in Natural Language Generation
D.D. McDonald 38

Knowledge Identification and Metaphor
R. Browse 48

Representing and Organising Factual Knowledge in Proposition Networks
R. Goebel, N. Cercone 55

Capturing Linguistic Generalizations in a Parser for English
M. Marcus 64

Representation I (Wed., 4:40)

Semantic Networks and the Design of Interactive Information Systems
J. Mylopoulos, H.K.T. Wong, P.A. Bernstein 74

Organization of Knowledge for a Procedural Semantic Network Formalism
P.F. Schneider 81

On Structuring a First Order Data Base
R. Reiter 90

The Genetic Graph: A Representation for the Evolution of Procedural Knowledge
I.P. Goldstein 100

Vision I (Thurs., 9:00)

Low-Level Vision, Consistency, and Continuous Relaxation
S.W. Zucker 107

Photometric Stereo: A Reflectance Map Technique for Determining Object Relief from Image Intensity
R.J. Woodham 117

Image Segmentation and Interpretation Using a Knowledge Data Base
S.I. Shaheen, M.D. Levine . . . 125

The Extraction of Pictorial Features
A.H. Dixon 135

Representation II (Thurs., 10:40)

Aspects of a Theory of Simplification, Debugging, and Coaching
G. Fischer, J.S. Brown, R.R. Burton 139

Knowledge Structuring: An Overview
M.S. Fox 146

Re-Representing Textbook Physics Problems
J. McDermott, J.H. Larkin . . . 156

Representing Mathematical Knowledge
E.R. Michener 165

Learning (Thurs., 3:00)

BACON.1: A General Discovery System
P. Langley 173

Learning Strategies by Computer
Y. Anzai 181

A Computer Program that Learns Algebraic Procedures by Examining Examples and by Working Test Problems in a Textbook
D.M. Neves 191

Natural Language II (Fri., 9:00)

Co-operative Responses: An Application of Discourse Inference to Data Base Query Systems
S.J. Kaplan, A.K. Joshi . . . 196

A Progress Report on the Discourse and Reference Components of PAL
C. Sidner 206

Participating in Dialogues: Understanding via Plan Deduction
J.F. Allen, C.R. Perrault . . . 214

Analyzing Conversation
G.I. McCalla 224

Vision II (Fri., 10:40)

Hypothesis Guided Induction: Jumping to Conclusions
E.C. Freuder 233

Explorations in Visual-Motor Spaces
Z. Pylyshyn, E.W. Elcock, M. Marmor, P. Sander 236

Using Multi-Level Semantics to Understand Sketches of Houses and Other Polyhedral Objects
J.A. Mulder, A.K. Mackworth . . 244

A Procedural Model of Recognition for Machine Perception
W.S. Havens 254

Problem Solving II (Fri., 3:00)

Approaches to Object Selection for General Problem Solvers
P. London 263

Experimental Case Studies of Backtrack vs. Waltz-Type vs. New Algorithms for Satisficing Assignment Problems
J. Gaschnig 268

Distributed Problem Solving: The Contract Net Approach
R.G. Smith, R. Davis 278

Programming (Fri., 4:20)

Describing Programming Language Concepts in LESK
D.R. Skuce 288

TONAL: Towards a New AI Language
D.J.M. Davies 296

Examples of Computations as a Means of Program Description
M.A. Bauer 304

SLEUTH: AN INTELLIGENT NOTICER[1,2]

Dr. Steven Rosenberg

Massachusetts Institute of Technology
Artificial Intelligence Laboratory

Abstract

Traditionally, programs in AI have used micro-worlds with completely defined semantics. Most real world domains differ from these micro-worlds in that they have an incomplete factual database which changes over time. Understanding in these domains can be thought of as the generation of plausible inferences which are able to use the facts available, and respond to changes in them.

A traditional rule interpreter such as Planner can be extended to construct plausible inferences in these domains by A) allowing assumptions to be made in applying rules, resulting in simplifications of rules which can be used in an incomplete database; B) Monitoring the antecedents and consequents of a rule so that inferences can be maintained over a changing database.

The resulting chains of inference provide a dynamic description of an event. This allows general reasoning processes to be used to understand in domains for which large numbers of Schema-like templates have seemed the best model.

Part I

Introduction: Suppose that you were a farmer, and each week you must make decisions about your wheat crop based on the the price you expect to get for it. You formulate the hypothesis to yourself: "will the price of wheat rise?" In making this decision you have available a stream of constantly changing information ranging from weekly reports on global demand to your own knowledge about the weather and state of your crop. As an aid in making your decision you would like to know whenever these facts can be organized into a descriptive scenario which supports your hypothesis.

1. This research was supported in part by ARPA contract N00014-75-C-0643.

2. This research was aided by many fruitful conversations with the members of the AI lab, particularly Ira Goldstein and Jim Stansfield.

This is a task which I call Intelligent Noticing. It is the sort of thing that we all do whenever we follow reports of developing events such as election campaigns. It is a type of task which can be difficult to do when the amount of information available is large and changes frequently. For instance, if you were charged with preparing the President's daily news summary, you would have to read hundreds of reports each morning, and analyze them in sufficient detail to summarize only those parts which were relevent to the president's concerns.

In this paper I will discuss the design of a program, Sleuth, for performing intelligent noticing. The program's intended role is to serve as a consultant to a decision maker, by serving as an intelligent noticer. Sleuth's purpose is to recognize events in commonsense domains which support the hypotheses it is given, and to keep these scenarios current while the information changes. Ideally, Sleuth should be smart enough to alert us only when disparate pieces of information can be connected in a plausable scenario. Thus Sleuth must be more powerful than a keyword type of system. Sleuth must be able to make inferences so that the number of false alarms given is low.

General Issues in Intelligent Noticing

"Commonsense domains" are the real world domains which everyone understands; which don't require great problem solving skills. Common sense domains share certain features. They are

A) Incomplete
B) Unstable

They are incomplete because not all the information we might need in order to make an inference is available at a particular time. For instance, if you must decide if the price of wheat is going to rise, you may know only the supply, and not the demand for wheat. They are unstable since the particular subset of information available can change fairly rapidly in the real world. This gives representations of these domains a feature of instability.

Each week the farmer will receive different reports concerning his wheat crop. Each of these may affect his calculations concerning the price of wheat. However, since the various reports arrive at different times, at any one time some pieces of information necessary for making a decision may be missing.

Knowledge based approaches to understanding in common sense domains have focused on the use of frames or scripts (Cullingford, 1977; DeJong, 1977; Reiger, 1977). This has been a successful approach in domains with well structured semantics in which new information can be viewed as instantiating static frames.

However, for complex events it is impossible to give static descriptions which encompass all alternatives. Intelligent noticing has a problem solving flavor in that the goal is to try and create a plausable scenario out of whatever pieces of information happen to be available. The design of Sleuth extends traditional problem solving approaches to common sense domains. It trys to provide a more unified view of problem solving and common sense understanding by considering the creation of event descriptions from a given set of assertions to involve a series of inferences which link these assertions in support of some central hypothesis. The justification for this dynamic view of event instantiation derives from an examination of the properties of complex events.

A) Events have fuzzy boundaries.

Consider a complex event, such as the recent invasion of Lebanon. What are the components of this event? We could define it as consisting of all armed clashes between Israeli forces and Palestinians after Israeli troops crossed the border. However, some people might also include the previous raid into Israel which provoked the invasion. We could go even further, and expand our description to include the United Nations meetings on the invasion, the attitude of Syria, or even the events that led to the creation of the Palestinian refugees. There is no fixed definition or single frame for this event. Like many others it has fuzzy boundaries. Using inferences to link the available facts yields a dynamic description for an event. The "boundaries" can be extended at will by making further inferences to include more facts.

B) Events are not "things".

For example, consider Israeli versus Palestinian descriptions of the invasion. Inference processes are flexible enough to capture this distinction by applying different rules of inference to some initial hypothesis to generate different chains of inference linking different sets of assertions in support of the same central hypothesis.

C) Events have variable instantiations.

For any particular kind of event, a different subset of features may be missing for each instance. By using chains of inference to connect assertions we can evaluate the plausibility of any particular event description without necessarily having to specify beforehand all acceptable partial instantiations.

D) Anything could be relevant.

If we ignore plausibility, we can create an event description with almost any set of assertions. For example, if we follow sports we might postulate a scenario in which the Texas Rangers win the world series. We do not want to exclude such far fetched scenarios a priori from the range of possibilities. Indeed, we wish to do just the opposite. We would like to retain the ability to create event descriptions which involve a given set of facts, and then judge whether the scenario is plausible or not. This allows us to adjust the false alarm rate to correspond to the expected utility of the result.

There are special problems in recognizing an event which arise from its essentially dynamic nature. For instance the assertions comprising a particular event may be added to the database over a period of time. This corresponds to fact that events occur sequentially over time. Sequential instantiation of an event can lead to problems.

A) Current assertions may become obsolete, change or be deleted before the final components of the description are added to the database. An event description is built on shifting sands, so to speak, and in generating these descriptions we must be able to respond to these changes.

B) Conditionality in the recognition process. Where we go depends on what has gone before. For example, in a condition called Frost Heaving, a sequence of thaws and subsequent freezes can tear the roots of the winter wheat crop. If however, a thaw is followed by another thaw, the next freeze will not damage the roots. Consequently, the order in which the freezes and thaws occur is just as important as the number. Somehow we must keep a history of the calculation. This will allow us to determine which features to attend to based on the features we have already seen.

Part II

In this section I will discuss how a traditional reasoning program, such as Planner, can be extended for use in a domain with incomplete information. Consider, for example, how a farmer, (let's call him Farmer MacDonald), might go about making weekly decisions concerning

his crop of wheat. He formulates the hypothesis: "will the price of wheat rise?". If he can generate a scenario from available information which supports this hypothesis, he will adopt one set of farming strategies; if not, another. If Farmer Macdonald has a consultant program which resembles Planner (Sussman, Winograd, and Charniak, 1971), he might formulate the problem as follows:

```
(Thgoal (price-increase wheat) $?T)
```

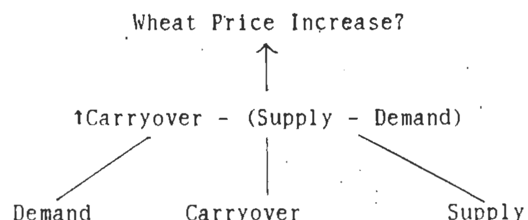
This goal is a hypothesis about the state of the world. The consultant makes inferences on the current set of assertions to see if an event description supporting it can be generated. We can define an event as being a set of assertions, justified by rule instances, which support some central hypothesis. If farmer MacDonald's assistant knows the following theorems and assertions, it will be able to construct a plausible chain of inferences:

```
(Thconse Thm1 (x) (price-increase $?X)
     (Thor (Thgoal (Supply-&-demand $?X)
           (Thgoal (Speculation $?X)))

(Thconse Thm2 (x) (Supply-&-Demand $?X)
     (Thcond
          ((Thand (Thgoal supply $?X)
                  (Thgoal demand $?X)
                  (Thgoal carryover $?X)
             (greaterp (carryover x)
                        ((supply x) - (demand x)))))

(Thassert Supply Wheat 180,000,000 bushels)
(Thassert Demand Wheat 170,000,000 bushels)
(Thassert Carryover Wheat 15,000,000 bushels)
(Thassert old-supply wheat 182,000,000 bushels)
```

Thm1 specifies that to show a price increase for wheat, either of two subgoals can be attempted. The first subgoal specifies a pattern which matches that of Thm2. Thm2 will succeed only if the difference between supply and demand is greater this year than last. If so, Thm1 will succeed. A price increase for wheat is infered, and Farmer Macdonald can increase his wheat plantings. (Note that we are simplifying the decision processes of a farmer; he has not looked at demand for alternate crops; whether his production costs on wheat have gone up; nor what sort of growing season is predicted. However, our purpose is not to show how Planner can be used in farming, but to illustrate some limitations of Planner.) We can illustrate this graphicly as follows:



Current demand for wheat is reported weekly by the U.S.D.A., based on domestic reports, and satellite observations of foreign lands. Each week, as farmer Macdonald reads his newsletter, he marks existing assertions as old, and adds new ones.

A) Current information is marked as old, or erased:

```
(Thgoal (demand wheat $?X))
(Thassert (old demand wheat $?X))
(Therase (demand wheat $?X))
```

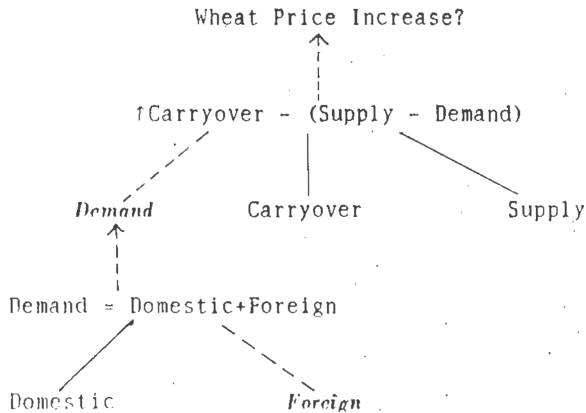
B) New information is asserted:

```
(Thassert (demand wheat 180,000,000 bushels)
```

Suppose that the next week, due to very foggy weather in central Asia, no satellite photos are taken. As a result, the U.S.D.A. issues no new demand statistics. However, the old demand values have already been erased. This time, when the consultant thinks about the price of wheat, the previous inferences would fail, since no assertion matching the pattern for (Demand Wheat $?X) would be found. For such cases, Planner provides a strategy. If an assertion is not in the database, Planner will try and prove it:

```
(Thconse Thm3 (x) (demand $?X)
  (Thcond ((Thand (thgoal domestic-demand x)
                  (thgoal foreign-demand x))
          (Plus (domestic-demand x)
                (foreign-demand x))))))
```

Theorem 3 states that to deduce demand for a commodity, find the foreign and domestic demand for this commodity, and add these together. If there are assertions for foreign and domestic demand, this theorem would succeed. However, since the total demand reported by the U.S.D.A. is based on the missing estimates of foreign demand, Thm3 will also fail. In this case, if there are no other methods for proving the missing assertions, the consultant must give up.



This presents a fairly brittle mechanism for dealing with domains which share the properties of incompleteness and instability. If the needed assertions are not in the database, and if they cannot be infered, the consultant will fail. It will fail, however, not necessarily because it is wrong, but because not enough information exists to make inferences. People are not quite so brittle reasoners, since they often cannot postpone decisions until more knowledge is available. For example, Farmer MacDonald might reason that as long as the supply of wheat is decreasing, it will be worthwhile to plant more regardless of demand. (Note: I am not suggesting that this is the "best" decision; merely that it is a plausible decision, and represents the sort of flexible reasoning which people are capable of.) By being willing to make assumptions, Farmer MacDonald is able to use a form of a rule which can operate on the information available. We might express this by modifying Thm2 to:

```
(Thconse Thm2A (x) (Supply-&-demand $?X)
  (Thcond ((Thand (Thgoal supply $?X)
                  (Thgoal demand $?X)
                  (Thgoal carryover $?x))
           (greaterp (carryover x)
                     ((supply x) - (demand x))))
          (Thelse
             (Thor
                (Thgoal supply-decrease $?X)
                (Thgoal demand-increase $?X)))))
```

If the full set of assertions concerning supply, demand and carryover are unavailable, Tm2A now suggests either trying to prove that supply has decreased, or demand has increased. By creating goals that require only a subset of the assertions that our original theorem required, Thm2A starts to capture our notion of <u>rule simplification</u>.

Thm2A does not quite capture our intuitions about simplifications. A rule should give advice about which other theorems can function in its stead as simplifications. We can then choose to use this advice or not, depending on our strategy. By treating simplifications like other goals in Thm2A, we loose this intuition. More importantly, while we can express simplifications of rules as theorems of the same sort as other theorems, they are not equivalent to other consequent theorems. A simplification is not used as a sub-goal in instantiating a theorem. Instead it replaces that theorem and is an alternative method for proving that theorem's goals *given the conditional circumstances of missing antecedents.* The simplifications, for the purposes of making inferences, are considered equivalent to the original theorem under the given circumstances.

A rule can have more than one procedural counterpart. Part of Planner's contribution to the notion of pattern directed invocation of rules was the insight that a rule has both a consequent and antecedent meaning. These can be expressed as two classes of theorems, antecedent and consequent theorems, which can be invoked by different patterns. We can extend this a further step by postulating that a rule has another bundle of procedural counterparts corresponding to its simplifications.

Actually, these simplifications will simply be other theorems. However the knowedge of when these other theorems can be used as simplifications, and which theorems can be used must be represented. Sleuth can be viewed as an intelligent interpreter which can make use of this metaknowledge to substitute simpler theorems for a rule which fails. The appropriate place to specify this metaknowledge is in a separate class of theorems. e.g.

```
(Thconse Thm2 (x) (supply-&-demand $?X)
     (Thcond
        ((Thand (Thgoal (supply $?X))
                (Thgoal (demand $?X))
                (Thgoal (carryover $?X)))
          (greaterp (carryover x)
                    ((supply x) - (demand x))))))

(Thassume Thm4A (x) (Supply-&-demand $?X)
     (Thgoal (supply-decrease $?X))
     (Thcaveat (Default $?X)))

(Thassume Thm4B (X) (Supply-&-Demand $?X)
     (Thgoal (demand-increase $?X))
     (Thcaveat
        (Thnot (Thgoal (supply-increase $?X)))))
```

We now introduce a new class of theorems, such as Thm4A and B, indicated by the label Thassume.

These theorems contain information concerning simplifications and assumptions. A Thassumption will specify A) a goal; theorems satisfying this goal can function as a simplification; B) the assumptions involved in using that simplification (expressed as a caveat).

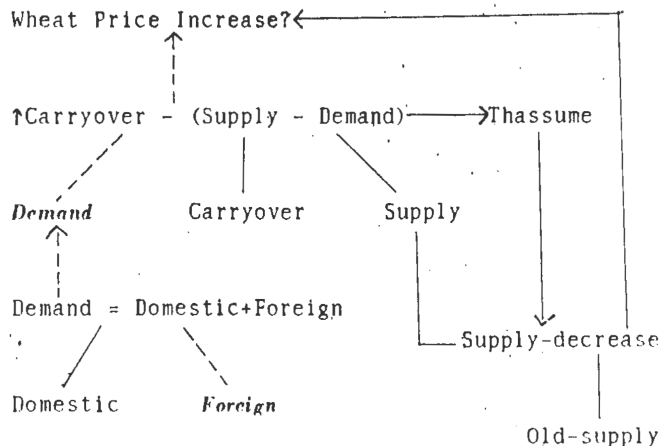
For instance, in Thm4A, proving a decrease in supply can function as a simplification of Thm2. Since no assumptions are specified in the Caveat, these can be ignored. This is explicitly expressed in the caveat as a default. If instead we use the goal of an increase in demand as a simpification, as in Thm4B, we must take account of the caveat that supply must not have increased for this simplification to be valid.

When Thm2 fails, the consultant can choose to make assumptions which will allow a simplification to succeed on the assertions which are available, by using Thm4A or B. A rule and its associated simplifications are related through the set of assumptions they embody. When Farmer MacDonald decides to ignore the demand for wheat, he is doing so because he is willing to assume that if demand changes, it will not change in a direction which would invalidate his proof. By making explicit this notion of assumptions, we can extend the list of options available in using a theorem to achieve a goal.

After Thm2 fails, simplifications will be considered, and Thm4A Found. Thm4A first tries to satisfy the goal (Thgoal Supply-decrease $?X). No assertion matching this pattern exists. However, a theorem, Thm5, can be used to prove this assertion.

```
(Thconse Thm5 (x) (Supply-decrease X)
   (Thcond ((Thand (Thgoal Supply X)
                   (Thgoal old-supply X))
           (Thcond ((greaterp (old-supply x)
                              (supply x)) T)))))
```

The value for old-supply was one of the initial set of four assertions given to the consultant. Since current supply and old-supply are known, Thm5 will succeed, and support the hypothesis of higher wheat prices.



This chain of inferences results in a less plausible scenario than one requiring no simplifications.

We may want to examine and save our scenarios. There are several reasons for this. We may want to know what assertions and theorems have been used, if we evaluate the evidence at a later time. If our hypothesis is stable, we may want to maintain a representation as the information it is based on changes. In a relational database there are many ways to represent events. We could, at each step, simply assert the necessary information. However, for each inference made there are potentially many kinds of information we may wish to save. For example, we may wish to indicate the theorems used in infering an assertion, the assertions these theorems use, as well as knowledge about which theorems failed. To represent this, we can annotate each inference. Annotation consists of an organized set of assertions of attributes of a labeled instance of a step in an inference. e.g.

```
(Thassert annotation1 goal (price-increase
wheat))

(Thassert annotation1 theorems
          (↑(Carryover - (Supply - Demand))))

(Thassert annotation1 assertions
          (Carryover Supply Demand))
```

Part III

The outcome of an intelligent noticing attempt in support of a hypothesis is a set of annotated assertions and instances of theorems. The annotation available at any given time represent the consultant's "understanding" of the domain. As the database changes, Sleuth will try and maintain its hypotheses. This will be reflected

in the changing set of annotation associated with each hypothesis. Sleuth assumes support for an hypothesis is conditional on the assertions available at the time it was first considered. Hence this support must be monitored and changed as the database changes.

Sleuth extends the concept of rule interpretation by making the maintenence of goals a function of the interpretation of rules. Sleuth does this by giving each active rule the autonomy to respond to changes in its environment. As each rule is interpreted, Sleuth creates an associated Sentinel for that instance of the rule. The sentinel gives the rule instance the knowledge of how to respond to changes in its antecedents or consequents. The result is the maintenence of hypotheses through a method of local autonomy.

By using sentinels, Sleuth extends the basic idea of a rule which is evaluated successfully if its antecedents are satisfied at the initial time of evaluation. A rule instance must be continuously enabled while it is used in support of some hypothesis. Before describing sentinels we must first consider under which conditions we wish the rule instance to be active.

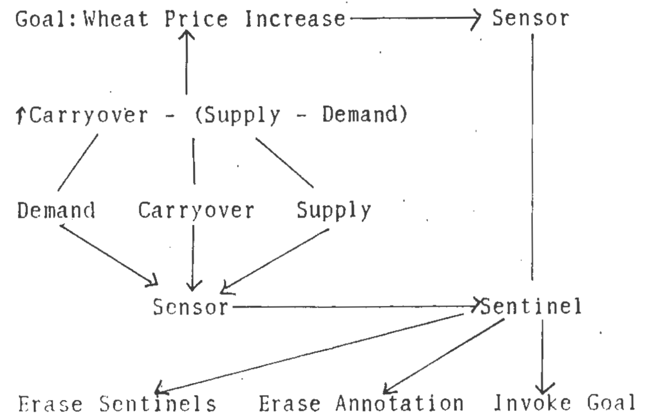
1) An instance of a rule used in support of some hypothesis must continue to have its conditions met while that hypothesis is successful.

2) We can extend this to theorems which fail. These can also be monitored, as long as the hypothesis is successful. Failed rules may succeed at subsequent times, if the necessary antecedents are asserted or proved.

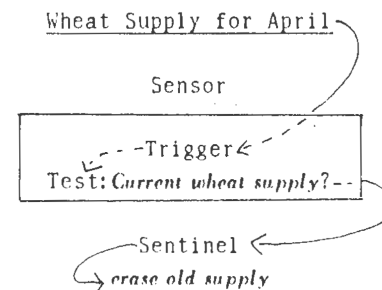
3) This notion can be extended one step further. The hypothesis may have initially failed. However, the goal of evaluating that hypothesis is maintained. In that case, the theorems attempted are still active, although no event description or scenario exists. If, at a later time, they can succeed, they will reactivate the attempt.

Therefore we can define an active instance of a rule as being any instance for which annotation exists.

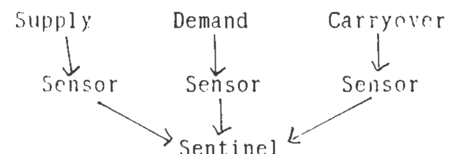
Sentinels were first developed by myself and Jim Stansfield as an aid in instantiating frames and automating the recognition of simple sequences of events within a changing database, using FRL (Roberts and Goldstein, 1977). Sleuth extends this by associating Sentinels with the interpretation of rules, and by making this use of sentinels a property of the rule interpreter, rather than a task for the user. A sentinel associated with a rule instance will look like:



A sentinel has sensors which report to it. (Sensor => Sentinel). A sensor has a two part condition. The first part, a trigger, is a demon which resonds to changes in the pattern that triggers it. For example, in the following case, the trigger responds to any addition or deletion of patterns involving the wheat supply. The sensor then tests the pattern against some criterion. For instance, this sensor is only interested in assertions concerning current wheat supply.



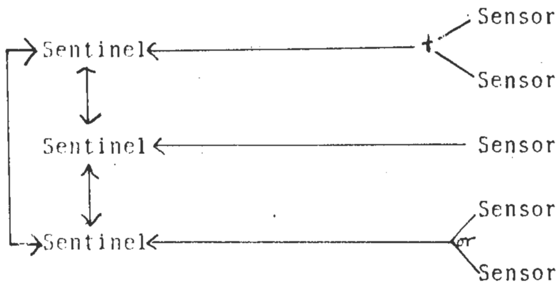
A sentinel can have several sensors which report to it. The sentinel is satisfied when some arbitrary logical conjunction of its sensors succeed. Although for the task of maintaining hypotheses more complex relations are not needed, a sentinel has the capacity to evaluate conditional relations among its sensors, and even to remove current sensors and place new ones as a resonse to these conditional constraints.



The sensors function as triggers for the sentinel, which is "data driven". A sentinel and its sensors are theorems which are created for a specific purpose. Unlike other theorems in the database, they have a limited lifespan. A sentinel can choose to erase itself and its

sensors upon completing its goal. For the current task, sentinels are not required when their associated rule instance is no longer active. In this case, the sentinel will erase itself.

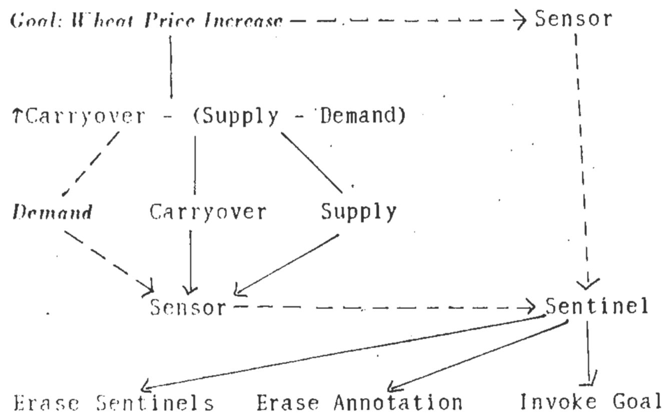
Since there will be several aspects of a rule's enviroment we wish monitored, we can create a cluster of sentinels, each of which is responsible for one aspect, such as the antecedents. Each sentinel in the cluster will know about the other sentinels. If any one succeeds, the other sentinels in the cluster will be able to erase themselves.



Sleuth creates, for each instance of a rule, a cluster of sentinels which monitor the rule's antecedents, goals, and annotation while the rule is active.

Lets examine what happens with a successful rule when its sentinel is triggered.

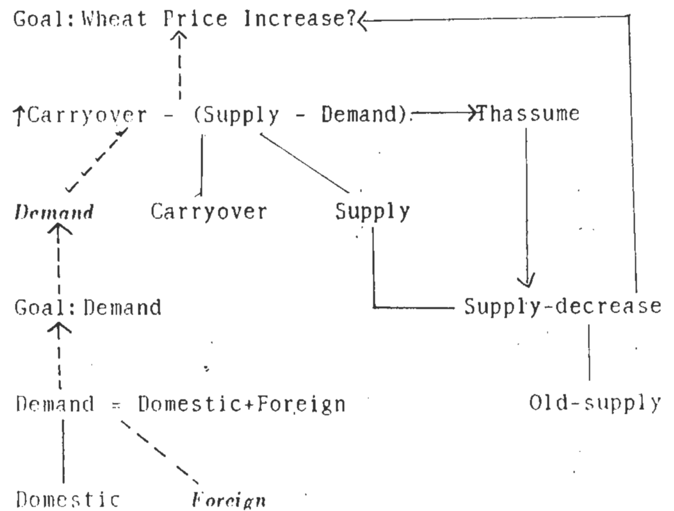
<u>*SUCCESSFUL RULE AND SENTINEL*</u>



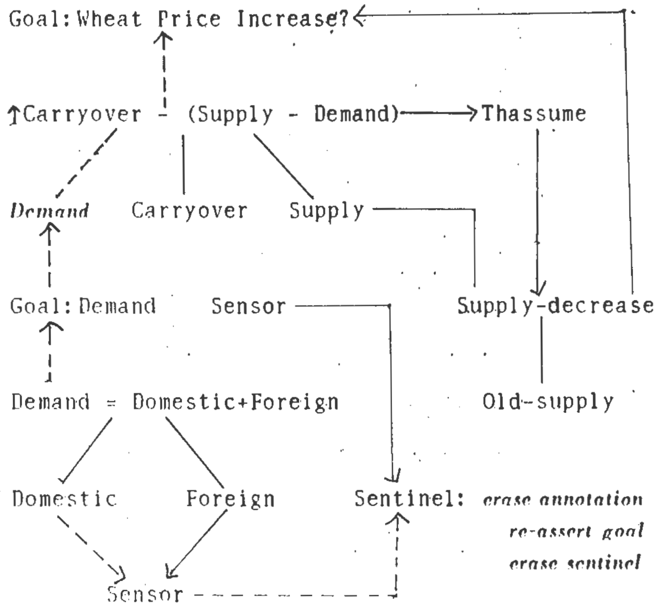
Individual rules will succeed or fail as a function of their antecedents. When a successful rule's antecedents change, its sentinel will be triggered. When this happens the sentinel causes the goal the rule was supporting to be re-evaluated. It removes the old annotation, as new annotation is created for the new evaluation of the goal. At this point the sentinel can erase itself. (Note: A sentinel causes a goal to be re-evaluated. There is no constraint that Sleuth must use the same rule again. However, in this and the following examples, it is assumed that there have been no other changes in the state of the system that would cause another rule to be selected first.)

If the rule's goal changes, perhaps because we are no longer interested in the hypothesis it supports, the sentinel will also be triggered. In this case we do not wish to re-evaluate the goal. The sentinel will remove itself and erase the associated annotation. Thus the rule instance will no longer be active, since no trace of it will remain.

Now let's consider how this local association of rule instances with sentinels can give rise to the right global behavior. The following represents the state of our deduction so far:

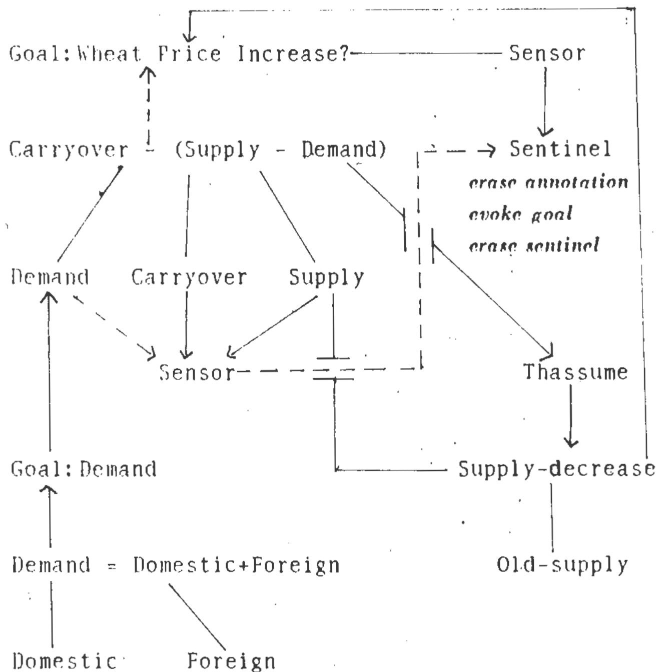


Suppose that a failed rule now is capable of succeeding, through its missing antecedent being asserted. For instance, the missing foreign demand for wheat can be asserted. This would trigger the sentinel associated with that rule instance:



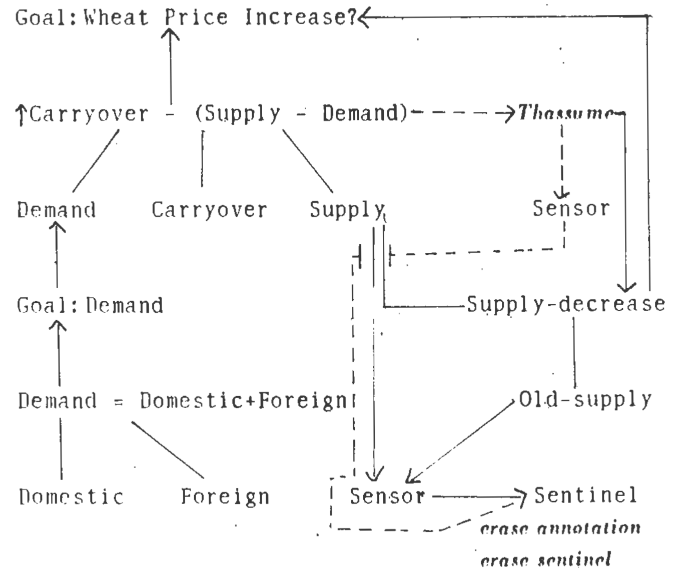
This will trigger the associated sentinel to erase the annotation for this rule instance, reassert the goal as something to be proved, and then to erase itself. This time the rule succeeds, resulting in a proof of the missing demand for wheat. This will in turn trigger the sentinel associated with the rule instance of which the missing demand is an antecedent:

FAILED RULE SUCCEEDS



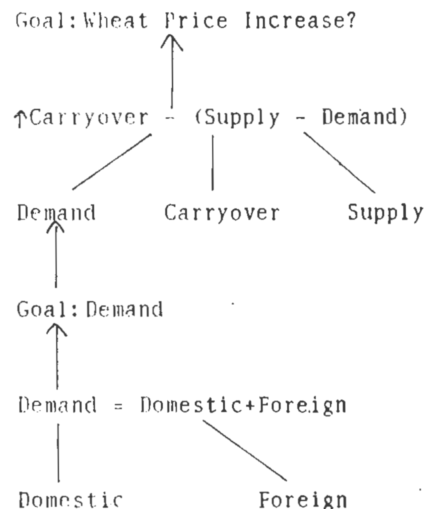
This sentinel repeats the actions of the prior sentinel. However, in erasing the annotation, it erases the record of the assumption made. This will trigger the sentinel on the rule which is a simplification. Since the use of a simplification is conditional on another rule failing, the sentinels associated with theorems used as simplifications monitor the annotation recording that failure, so that they will know when the simplification is no longer required. They will then respond to the erasure of this annotation by erasing the annotation for the simplification.

FAILED RULE SUCCEEDS



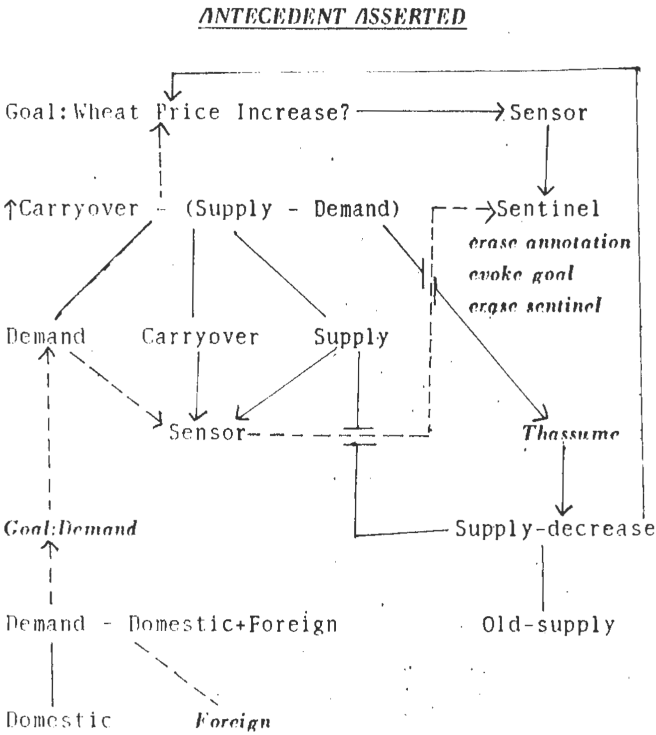
If Thm2 again failed due to a missing antecedent, Sleuth would once more try a simplification. Since the formerly missing antecedent for demand is known, Thm2 succeeds, and results in the following final proof tree:

FAILED RULE SUCCEEDS

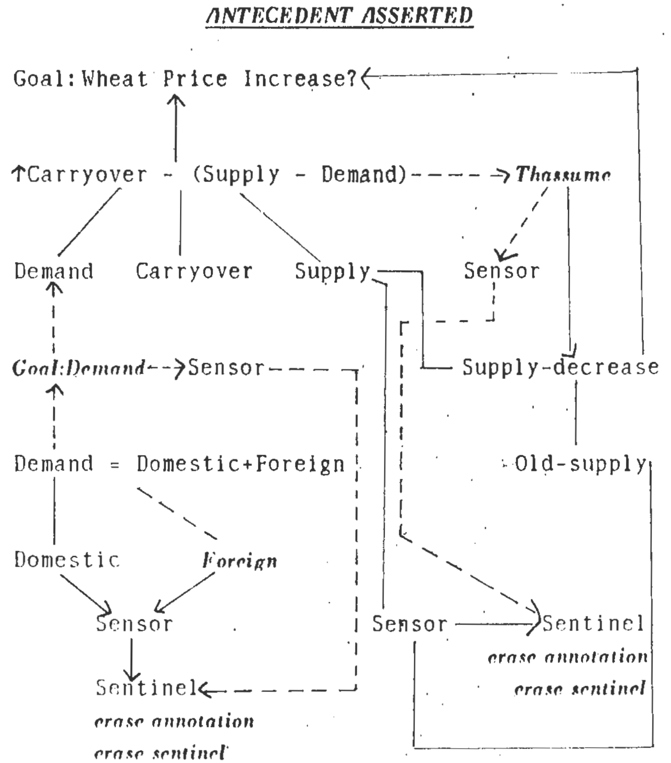


In this next example, the missing antecedent

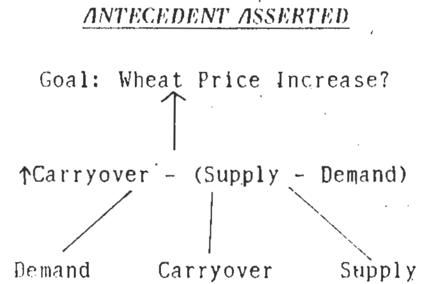
for wheat demand is asserted, although the rule involved (Thm2) has already "succeeded" by using a simplification. This will trigger the sentinel associated with the rule instance of Thm2:



When Thm2 is re-evaluated it succeeds without recourse to either using a simplification or trying to prove the now not missing antecedent. There is no explicit mechanism responsible for removing the now unneeded rule instances. Instead, by erasing its annotation, Thm2 triggers the sentinels associated with the subgoal of proving demand, and the simplification:



Both these sentinels, since their reason for existing is gone, erase the annotation, and then erase themselves. This results in the following final state:



Thus, unneeded rule instances will know when to remove themselves. Through local propagation, the representation responds to changes in the available database.

Consequently, intelligent noticing is a dynamic process. Once attempted, it is locally data-driven. These changes will reinvoke the goal of inferencing, which can then proceed in a goal driven fashion. Obsolete parts of the representation are able to remove themselves by noticing local changes in the environment.

Sleuth, once given a goal, will attempt to recognize this event whenever the database contains the right set of assertions. Sentinels set in the interpretation of rules will individually call Sleuth to re-evaluate particular goals. (This can be contrasted to Doyle's Truth Maintenence system for maintaining

contexts of assertions. (Doyle, 1977.)) Sleuth will develop new ways of supporting its hypotheses in response to these local calls for re-evaluation. Once applied, each sentinel has the autonomy to respond to changes in the database. These changes propagate, through a method of local autonomy.

The current version of Sleuth is programmed in FRL, a frame representation language (Roberts and Goldstein, 1977). FRL does not contain the pattern directed invocation of rules as a feature of the language. The current design iteration involves transfering Sleuth from FRL to a Planner-like language as described in this paper.

REFERENCES

Cullingford, Richard E. 1977, "Controlling Inference in Story Understanding", Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, Mass.

DeJong, Gerold. 1977, "Skimming Newspaper Stories By Computer", Proceedings of the 5th International Joint Conference on Artificial Intelligence, Cambridge, Mass.

Doyle, John. 1977, "Truth Maintenence Systems for Problem Solving", M.I.T. Artificial Intelligence Laboratory Technical Report 419.

Reiger, Charles. "The Magic Grinder For Story Comprehension". To appear in Discourse Processes, A Multidisciplinary Journal.

Roberts, Bruce R. & Goldstein, Ira P. 1977, "The FRL Manual", MIT-AI Memo 409.

Sussman, Gerald J., Winograd, Terry, & Charniak, Eugene. "Micro-Planner Reference Manual", MIT-AI Memo 203A.

A Test-bed for Developing Support Systems for Information Analysis.

Jim Stansfield.
M.I.T. AI Lab.

Fast effective decision making is necessary in many important activities. Complex dynamic situations that require control make huge demands on the decision maker to consider large amounts of information. This is the case in many areas of decision making including politics, economics, defense, business and medicine. Although data base systems allow easy access to masses of data, considerable processing must be done before an analyst can use the data effectively. I am designing *Support Systems* to relieve an Information Analyst of the burden of these tasks and leave him free to concentrate his expertise where it is most needed. The point of view is that analysts organize the data around "*mental models*" of their problem area. It follows that a support system that can manipulate structures corresponding to these models will be better able to assist an analyst. The domain being used is the analysis of supply, demand and price of commodities. In this paper I describe a program written in FRL (Frame Representation Language) [Roberts and Goldstein, 1977] which models a commodities situation and I discuss representation and reasoning in frames systems. Commodities markets are an information-rich real-world system in which practical decisions are continually being made. Moreover, there is a wealth of data readily available about the current commodity situation and a large background of literature concerning the theory and mechanism. A model of commodities markets is an excellent test-bed for developing support systems.

There are many reasons why an intelligent interface between an analyst and his data is desirable. First, the information an analyst receives may be incomplete, inaccurate or out of date. He may wish to check his sources, extend his information search, or bring old information up to date by estimation. A support system can make this practical. Second, the information relevant to any particular goal of the analyst must be sifted out. To do this, a support system needs to know about the problem solving methods used by the analyst even if it is unable to solve the problem itself. Third, implications of the data are needed. A support system could make inferences and provide an analyst with a complete picture of his situation allowing him to make quicker decisions. This requires the system to maintain theories about what can happen in its area of expertise. Finally, an analyst would like to check the repercussions of his decisions. These are often subtle and counter-intuitive as illustrated by the behavior of economic systems. By describing a situation as a dynamic model, the expected consequences of a decision can be forecast.

The commodities domain includes primary producers, storage managers, exporters, speculators and other agents. Each has a range of possible actions. For example, a farmer is able to produce, store and sell a variety of crops. He can choose how much to plant and when to harvest and sell. These actions are specific to him. A farmer is also in a position to tell the state of the crop before other participants and he has other private information such as the progress of the harvest or when farmers will sell. On the other hand, speculators are expert in estimating the impact of supply and demand factors upon price. Their decisions will affect the price of futures and indirectly control the behavior of storage managers and farmers. Some speculators may put more weight on certain information than others and so have different plans. The way the commodities market reacts to news depends on its state as well as the news. Thus, an analyst

should consider information about the situations and plans of the participants in order to determine the effect of news about a new grain sale or a dock strike.

Developing an analyst's support system with the capabilities mentioned is an ambitious goal. This paper describes the framework being used and illustrates it with a prototype program which achieves a small part of the desired expertise. I first describe the framework showing how the expertise fits into it. Then I describe the prototype program and explain how it will be extended. Finally, I sketch some of the developments that were needed to represent rules and constraints in FRL. The framework for support systems is shown in figure 1. It is organized using the acronym ART which stands for Analyst, Reporter and Test-bed.

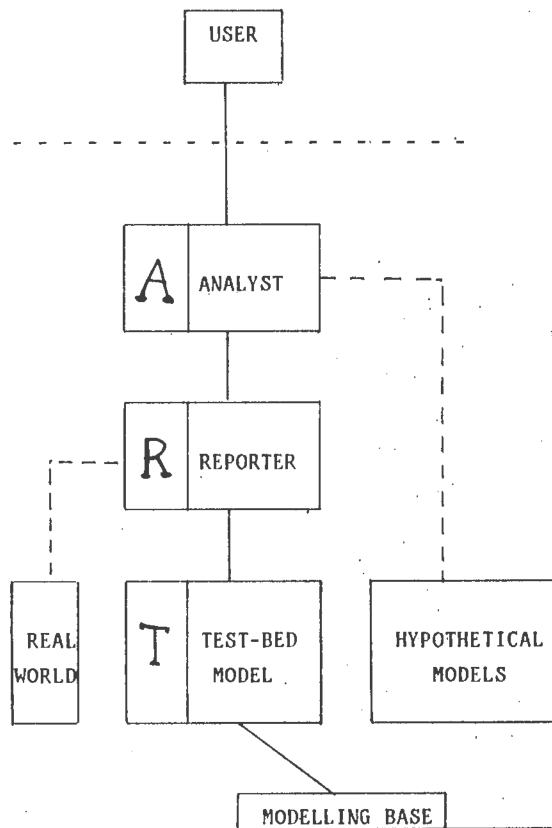


Figure 1. The ART architecture.

The first component, the *Test-bed*, is a simplified world model which can be used to generate behavior and reports for an analyst and is easier to work with than the real world. A test-bed takes the place of the real world and the user is assumed to have incomplete information about it. The user's information comes from the *Reporter* which generates reports resembling those a real analyst receives including incompletenesses, estimates and conflicts. The *Analyst* works as an assistant to the user and gathers information relevant to the user's goals, make inferences for him and test out his theories and decisions. Its role is to assist the user in making sense of data about the model world.

Two flows of information connect the components of an ART system. Commands flow downward from the user toward the test-bed and descriptions of the test-bed's behavior flow back up to the user. The command chain allows the user to request information from the analyst. Requests may be for a simple report, an analysis of a situation, or to ask the analyst to examine the consequences of a hypothetical event. The analyst remembers these user goals until the time that it can complete them. Some goals may be ongoing requests to monitor particular information. To carry out all or part of a request, the analyst needs information about the world which it obtains by asking the reporter. Requests for reports are generated from rules specifying the type of data needed for an analysis and from the analytical inference mechanism through subgoals which are immediately reportable. This illustrates that, to select relevant data for the user, the analyst has to know something about the expert's problem-solving methods. The reporter has a repertoire of methods for finding reports and keeps a check-list of the particular reports the analyst component is waiting for. In summary, commands pass down the command chain giving rise to goals which are stored in the analyst and the reporter. They determine what data flows upwards. The analyst acts as a filter to give the user only relevant information and as an organizer to provide possible models describing the state of the world.

The development of test-bed models is based on the viewpoint that decision-making situations can be considered as dynamic interactions between sets of agents each with its own views, expectations, goals and resources. In a political situation, the agents may be nations, economically they may be businesses and in the commodities world they include producers, storers, users and speculators. Each agent has a state and a strategy, and his

behavior depends on the interaction between his strategy and his current situation. For example, a farmer may decide to sell if he needs to make room in his store for the upcoming harvest. The strategies are executed by a modelling system to determine a set of actions for the player which produce changes in the model. If the execution cycle is repeated the agents interact since each one now behaves according to the state of the world which has arisen from previous actions of the others. From this viewpoint, test-bed models have a psychological aspect which makes them different from many simulation situations. They model the rule-based behavior of a set of interacting agents. Some concepts however have been adapted from dynamic simuation (Forrester, 1962) and an FRL-based version of the basics of Forrester's simulation system has been written.

Test-beds have important advantages. First they clearly define the limits of the domain being studied. Lack of such a clear definition makes it very difficult to keep the domain contained. Second, the analyst part of the system can use the same modelling system that the test-bed is based on. This clearly defines the space of possible models and the analysis problem becomes well-defined. Third, the test-bed can be used to examine the performance of the system. As the test-bed world is exactly known the accuracy of the analyst's hypotheses based on reports can be measured exactly. Support systems can be developed for an increasingly complex set of test-beds allowing expertise to be extended gracefully.

The analyst component is not just a modelling system since its job is to assist the user in determining the state of the world from reports. However, this can be seen as constructing a model to fit some data so the analyst must be an expert in building and discussing models. It uses a set of hypothetical models of the agents in the world and chooses appropriate ones based on reported information. These are assembled into a model of the entire situation and the alternative combinations are presented to the user with supporting explanations. At the time of writing, the model-building aspect of the analyst is still being planned.

The Prototype System.

The prototype system consists of a simple test-bed model and a basic reporter and analyst. The reporter and analyst are ad hoc LISP programs embodying a few simple rules. The reporter produces reports about a fixed set of situations and these reports trigger analytical rules which print out simple analyses. There is no provision for the user control described above and the analyst does not yet consider hypothetical models.

The prototype test-bed consists of a system which supports models, and a two player game running in that system. The game is between a producer of wheat and a user. Each harvest time the producer's inventory is increased by the production for that season. His goal is to sell his inventory throughout the year and end with clear storage ready for the next harvest. He sells to a market by offering quantities of wheat at the current market price. The user has some storage, and wheat is removed from this at a constant rate throughout the season. He makes bids to the market to replenish his store. The producer's strategy is to sell more when the price is high and less when it is low. The user does the opposite, buying more when the price is low. At the beginning of each season both the producer and user are given expected prices. These correspond to their opinions about the market and determine how they know when the price is good.

Here is an example of the game. Computer output begins with a "*", user replies begin with ">", and comments are in italics.

The game begins with a phase in which frame structures are set up to represent the players and the market. They are initialized in a dialogue with the user.

```
* This is a two player game between a producer of
  wheat and a user. It is just before harvest.
  How long is the season?
> 6
* What is the inventory of the producer at time 0?
> 20
* What is the inventory of the user at time 0?
> 60
* What is the market price at time 0?
> 25
* Its a new season. The carryover is 80.0. What
  is the production?
```

> 300
* What price does the producer expect?
> 26
* What is the demand?
> 200
* What price does the user expect?
> 24

When the game is prepared, rules are invoked to produce reports on the state of the system. The reports are given to the analyst which uses rules describing the players to make a forecast of the future behavior. I have rewritten the output in better English.

* REPORTER: There is a surplus of 180.
ANALYST: This will tend to make prices fall. The effect is large.
REPORTER: Farmers expect the price to rise.
ANALYST: They will hold onto their crop. This will tend to make prices rise. The effect is small.
REPORTER: Users expect the price to fall.
ANALYST: They will wait to buy. This will tend to make prices fall. The effect is small.

The model is now run for one cycle and the results are printed out.

* PRODUCER
INVENTORY: 320
DESIRED-SELL-RATE: 53.3
EXPECTED-PRICE: 26

USER
INVENTORY: 60
USAGE-RATE: 33.3
EXPECTED-PRICE: 24

PRICE AT TIME 0 = 25
PRODUCER OFFERED 51.2
USER BID 32
AMOUNT TRADED 32
PRICE FALLS TO 23.8

Next the reporter gives a bulletin describing a change in the situation, and the analyst forecasts the consequences.

* REPORTER: The price has fallen below the users' expected price.
ANALYST: Users' will increase their buying pressure.

After the model has been running for several time cycles, statistics for this period are printed out. We can see that the price fell due to the large supply and at the end of the season, in period 5, there was still a sizable carryover. User's initially refrained from buying but began to increase their inventory as the price fell below their expectations.

TIME	PRODUCER2 INVENTORY	USER3 INVENTORY	MARKET4 PRICE
0	320.0	60.0	25.0
1	288.0	58.7	23.8
2	254.4	58.9	23.0
3	219.6	60.4	22.3
4	183.7	63.0	21.7
5	146.9	66.5	21.3
6	409.3	70.7	21.0
7	347.1	66.3	21.5
8	283.3	63.4	21.9

Table 1.

The first part of the protocol is a dialogue with the user in which a game is set up. A game is represented as a frame structure which has a GAME frame at top level and includes slots for the producer, the user and the market. The producer is an instance of a generic PRODUCER frame and similar situations hold for the user and the market. The top-level frame for the game is shown in figure 2 in a simplified form.

```
GAME1
  SEASON    $VALUE  6
  PRODUCER  $VALUE  PRODUCER2
  USER      $VALUE  USER3
  MARKET    $VALUE  MARKET4
  TIME      $VALUE  TIME-0
```

Figure 2. The top-level game frame.

The generic GAME frame knows how to ask the user how to set up a new game. During this process it needs to set up a new PRODUCER frame. Since the generic PRODUCER frame knows how to set up a new producer, not all of the setup work is done by GAME. Knowledge about setting up any

frame is attached to the corresponding generic frame either in a special slot or in each slot of the frame under an $ASK facet. The system also provides standard setup procedures and is able to check that constraints hold true for the answers to its questions to the user. The setup mechanisms and the constraint handling mechanisms are discussed elsewhere [Stansfield 1977]. In the example game, the producer and user were set up with just an initial inventory and a strategy. The program chose their strategies by default.

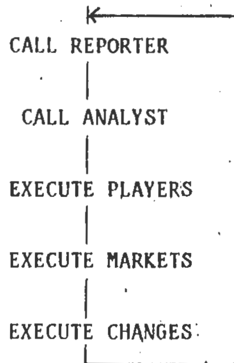


Figure 3. The Execution Cycle.

The system repeatedly executes the cycle of operations shown in figure 3. At the beginning of the cycle it checks if it is a new season which triggers questions to update the system. The reporter is called next and applies a set of rules which examine the state of the test-bed. These may instantiate actions to produce short reports which are passed on to the analyst. The analyst has a set of rules describing market behavior and player behavior. Using the reports, it applies its rules and produces statements about the behavior that can be expected from the system and the reasons for this behavior. Next, the players strategies are executed giving rise to offers and bids which are placed in the market. The market is executed and the offers and bids become trades and price changes. Offers, bids, and trades are all represented as frames giving buyer, seller, price, and date. The market executes trades to produce change descriptions that correspond to changes in the inventories of the players.

The producer, the market, and the user are arranged in the chain shown in figure 4. The behavior of any element in the chain affects the adjacent elements giving rise to feedback loops. There are two negative feedback loops. The producer increases his sales as the price rises which in turn makes the price fall. The user increases his demand if the price falls and this makes the price rise. Feedback loops are important determinants of system behavior inasmuch as they make simple cause and effect descriptions of behavior inadequate.

Figure 4. The example game arrangement.

Figure 5 shows the arrangement of influences that make up the producer's strategy. Total supply for the season determines his desired selling rate each cycle since his goal is to clear his storage. His desired selling rate is one factor in determining how much he offers at any time. If he desires to sell more each month he clearly must offer more. The other determinant is the difference between his expected price and the actual price. A plus sign on an arrow in the figure means that if the factor at the tail of the arrow increases, it will cause an increase in the value at the head. A minus sign means that a positive change will cause a decrease. The arrow from the amount offered to the market price is part of the market structure.

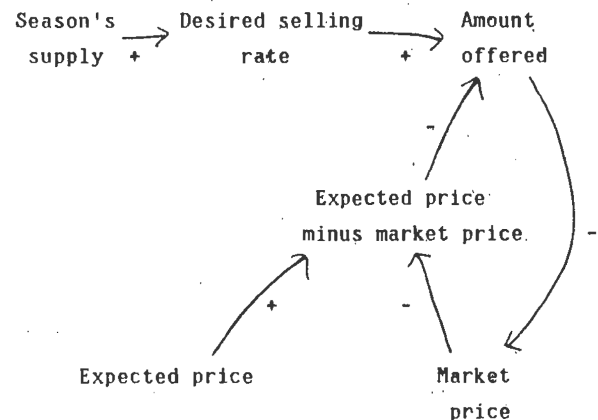


Figure 5. The producer model.

To see how this works, examine the first entry in table 1. From the setup data the season length is 6 and the supply is 320. This implies a desired selling rate of 53 per cycle. Since the expected price is 26 but the market price is only

25 the producer decided to hold back and offered less than 53. The user only bid for 32 units however, so only 32 were traded. Demand was insufficient.

A similar diagram would illustrate the user's strategy. His desired buying rate is determined by the demand rate per cycle for the season. In this case, a rise in the expected price causes a rise in the amount bid for and this causes a rise in the price due to the increased demand.

The market is an interface between the producer and the user. Since different amounts are offered than are bid for, it provides a mechanism for equalizing these. The price of the market is adjusted to reflect the difference in buying and selling pressures. When buying and selling pressures are equal, the price will not change. It must be remembered that this is a feedback situation and a change in price will affect both of these pressures. Figure 6 illustrates the two feedback loops involved in a market situation.

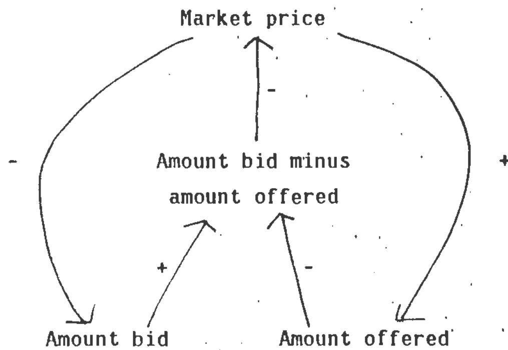


Figure 6. The market feedback structure.

Each model component represents a large number of players as a group. They have the same strategy but their expectations and decisions are distributed. A component of the model with an expected price of 25 represents a group of players with similar strategy whose mean expected price is 25. This is necessary if the system is to model the collective behavior of participants in the commodities world. Whenever a group of participants can be divided into two classes with different strategies, such as optimists and pessimists, or risk takers and risk avoiders, two components can be used to model them.

Extending the model.

The prototype system modelled a simple commodities situation and its analyst component embodied only a few analytic rules. This section describes how the scenario can be extended and discusses the kind of analysis that will be needed.

The next version of the test-bed will include four types of agent which are, producers, storers, consumers and speculators. They will interact by means of two markets, a cash market and a futures market. The cash market is used to buy and sell the actual commodity and the futures market to trade futures contracts. This situation, although only twice the number of component types than before, allows many more strategies and covers many situations in the real commodities world.

There will also be a four or five-fold increase in the number of possible strategies for an agent. A farmer will have to make decisions about planting, storing, harvesting and selling and will also make an estimate of the expected price. The basis for such decisions are his current state including acreage and available storage, and some information about the rest of the system and about the weather. For example, harvesting time depends on whether the crop is ripe, how much needs to be harvest, whether the crop is susceptible to wind damage and whether it is dry enough for the machinery to work. Harvest time is also a crucial time since much of the crop flows to market and speculators must watch developments to estimate prices and quality.

In order to analyze situations involving several agents the analyst needs rules to describe them. Such a rule-based model of an agent is different from the test-bed's model since it is used to reason about the agent rather than to execute a model of him. In the prototype these rules were embedded in a simple LISP program. Future analysts will be written using rules implemented in FRL. Rules will be in the form of productions whose condition is some state of affairs that the agent reacts to and whose action is the consequence of that state. Simplified versions of the rules embodied in the prototype are shown below.

Producer

Producer expects price rise -> Producer holds
Producer holds stock -> Selling pressure less
Harvest is near and insufficient storage
-> Producer clears storage
Producer clears storage -> Selling pressure more

Market

Selling pressure higher -> Price tends to fall
Buying pressure higher -> Price tends to rise

Rules can be used in several ways. If the reporter gives the analyst information which matches the condition of a rule in one of its farmer models, then that model can be hypothesized as applying. The consequence of the rule can be expected and if it occurs it will support the hypothesis. Alternatively, rules can be used from right to left. If the reporter gives information about a farmer's actions which corresponds to the right hand side of a rule then the analyst can hypothesize that the rule may be attributed to the farmer and that he is reacting to a state of affairs corresponding to the left hand side of the rule. Again, several rules may fit and multiple hypotheses may be carried forward.

The modeling rules embody the common-sense knowledge an analyst has about the actions of agents in his world. From a person's actions we can sometimes infer the plans he is executing, the situation he is responding to and his intentions. Indeed, this ability in peolpe is crucial to their effective communication. The analyst uses rules to mimic this and to fill out a picture of the world from partial information.

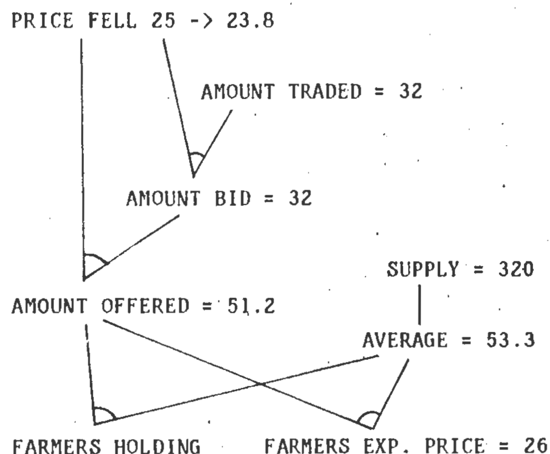


Figure 7. Analysis trace.

Figure 8 shows an example of chaining of rules that falls within the context of the session with the prototype presented earlier. Suppose the reporter said that the price fell, the amount traded was 32 and the years supply was 320. It is possible to deduce from this that farmers expect the price to rise. Since the price fell, the producers were offering more than the users bid for. The users therefore filled their orders and must have bid for 32. The price fall of 1.2 and the amount bid of 32 corresponds to an offering of about 51. But the average sale need to clear the total supply is 53.3 since there are 6 cycles per season in the model. So farmers were holding back. This means they expect the price to rise.

Clearly this is a simple analysis based on a few rules. However, analysis such as this will come into play in the extended system. For this reason, I am developing a representation for rules in FRL and discuss this next.

Frames.

The system is implemented using FRL, a frames based language written in LISP. I have extended this for setting up frame structures by dialogue with a user, for representing events as clusters of frames and for reasoning. A simple FRL frame represents an item and has of a set of properties of that item. A PERSON frame may have the properties HEIGHT and NAME and each property may have a value. Frames form a hierarchy, so the frame for a particular person, PERSON1, will be lower in the hierarchy than the *generic* PERSON frame which will be lower than the ANIMAL frame. Inheritance is an important property of the frame hierarchy. Lower frames inherit information from the more general frames higher up. This is a powerful organizing principle for information. Procedures can be attached to a slot of a frame to accomplish various actions. An *if-added* procedure triggers when a new value is added to the slot so that the new information can be processed. *Requirements* check values before they are accepted and can call complaint mechanisms if they are violated. *If-needed* procedures are used to calculate values of slots from other information. All these procedures are inherited in the frame hierarchy just as values are. This means new concepts can be defined as generic frames

containing sets of properties and procedures to handle them.

Representation.

When the analyst component is written in terms of rules, it will use rules similar to production rules. The left hand side will be an event or combination of events and the right hand side will be an action or statement which is a consequence of that event occurring. It was necessary to define event-like concepts in frames. A representation scheme based on the facilities of FRL was applied to business events from the commodities world such as "selling", "exporting" and "exchanging". Concepts are represented as *clusters* of frames and there are primitive events, such as "changing", which describe modifications to a frames data-base. A "changing", for example, might describe a change in the value of a particular slot of some frame at a particular time. Further event types are constructed from the primitives by *specialization* and *aggregation*. To specialize an event type, another is created below it in the frame hierarchy with extra knowledge attached in the form of procedures and values. One specialization of "changing" is "growing" where the HEIGHT slot of a specified frame is changed. Aggregation is a way to build new event types by putting together several others. A new aggregation includes a set of constraints that specify how the component events are arranged in any particular instance. For example, "exchanging" is a simple aggregation composed of two "transferring" events between the same two parties but in opposite directions. Aggregation and specialization can be freely applied to aggregates and specialization and an infinite variety of new concepts can be defined.

Although events are constructed from primitives as in conceptual dependency [Schank, 1975], our representation scheme is different in many respects. An event need not be reduced to its primitives but may be treated at a higher level. Also, specialization highlights the differences and similarities between events in a clear way.

In both representation and reasoning I have taken a different approach from KRL [Bobrow and Winograd, 1977]. KRL is a large complex system and perhaps unwieldy to apply since it addresses very many representation issues. In contrast, I began with a simpler core system and have built on it as the application required.

Constraints.

Reasoning about frame structures is needed for the analyst component of the support system. Rules can act as constraints between the values of a set of slots in a frame structure. A constraint watches over a *domain* of slots. When any domain slot is altered, the rule is invoked and applied to the domain. Rules are made general by placing their *triggers* as if-added procedures in slots of generic frames. Because of inheritance, a rule then applies to any instance of the generic frame.

There are two cases, *simple constraints* and *complex constraints*. Simple constraints have all their domain slots within a single generic frame. Such a constraint handles the SUPPLY, CONSUMPTION and CARRYOVER of a crop forecast frame, making sure that SUPPLY minus CONSUMPTION equals the CARRYOVER. The triggers are added to slots in the generic CROP-FORECAST frame. Each instance of a crop forecast is then subject to the rule. Any new information about supply, consumption ar carryover will invoke the rule. Simple constraints are discussed in earlier papers [Stansfield, 1977; Rosenberg, 1977].

The complex case constrains slots which are spread over a group of frames. One frame is considered *central* and domain slots outside it are called *outliers*. An example is the HARVEST frame. The AMOUNT of a HARVEST event is its ACREAGE multiplied by the YIELD of the FIELD harvested. The YIELD slot is an outlier since it belongs to the FIELD frame. This frame is not even known in a particular instance until the FIELD slot of the HARVEST is given a value.

Complex constraints require two new mechanisms, a *reference mechanism* and an *identification mechanism*. Each outlying slot is associated with a corresponding *auxilliary slot* in the central frame. The auxilliary slot of HARVEST is a YIELD slot to hold the yield of its field. Reference triggers are set up so that when a value is added to the FIELD slot the auxilliary slot will be identified with the YIELD slot of the FIELD. By this method, a complex sentinel is reduced to a simple sentinel and any rule can be defined within a single generic frame. The

calculations are done within that frame using auxilliary slots as temporary variables.

Directions.

Frames-based semantics and constraints will both be used in developing the rules of the Analyst component of ART. The current test-bed will be extended to provide a qualitatively rich repertoire of behavior for the Analyst to deal with. Several experiments will be possible with the proposed version of the system. Since it will be rule based it will be possible to interact with an expert analyst to determine a set of rules which will help him in his analysis. It will also be possible to try interfacing the system to actual commodities news rather than a test-bed model.

References.

Bobrow, D. G. & Winograd, T. An Overview of KRL, a Knowledge Representation Language. Cognitive Science vol 1, 1. 1977.

Forrester, J. W. 1962, Industrial Dynamics. MIT Press.

Roberts, R. B. & Goldstein, I. P. 1977, The FRL Manual. MIT AI Lab memo 409.

Rosenberg, S. 1977, Reporter: an Intelligent Noticer. MIT AI Lab working paper 156.

Schank, R. C. 1975, Conceptual Information Processing. North-Holland.

Stansfield, J. L. 1977, COMEX: A Support System for a Commodities Expert. MIT AI Lab memo 423.

ACKNOWLEDGEMENTS.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. It was supported by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

LOCATING THE SOURCE OF UNIFICATION FAILURE

Philip T. Cox

Department of Computer Science
University of Waterloo
Waterloo, Ontario

Abstract: The problem of determining whether or not a set of expressions is unifiable is of key importance in many applications, mechanical theorem proving in particular. Consequently, the unification problem has been studied intensively. A related problem of equal importance in theorem proving and other applications which use unification, has received little attention; namely, the problem of what to do when unification fails. In a recently proposed deduction system, the first step in solving this problem is to find out why unification has failed. We describe a method for accomplishing this step.

1: Introduction

There are many applications in which the problem of unification plays an important role: mechanical deduction is probably the best known of these applications, and has caused intensive study of unification since Robinson presented his Resolution Principle and unification algorithm in 1965 [10]. A problem which has received little attention, however, is the problem of what to do when unification fails.

When a mechanical theorem-prover is searching for a proof, there is usually a variety of actions to be performed: the system must choose the subproblem to work on next, then choose one of several solutions to it. At some point in every search for a proof, however, it usually happens that the system fails to solve a subproblem because two literals cannot be unified. The system must then "backtrack" to some earlier point in the search, and attempt an alternative solution to a previously solved subproblem. The usual backtracking strategy employed is to return to the last point in the search at which there exists an untried alternative solution. This may not be the correct place to try an alternative, however, and although the correct point will eventually be reached, much effort will meanwhile be expended in exploration of irrelevant areas of the search space.

When a nonunifiability arises, the object of backtracking is to remove this nonunifiability. The exhaustive technique described above investigates only one of several ways of removing it. This limitation results from the fact that most theorem provers apply the unifying substitutions to the clauses in the proof; this makes it difficult to locate any of the sources of a unification failure except the obvious one used in exhaustive backtracking. In a new deduction system proposed in [6,7], substitutions are not performed: instead, each application of a deduction rule produces a set of constraints (a "constraint" is an unordered pair of expressions); and the set of all constraints produced in the construction of the proof is checked for unifiability. If it is not unifiable, all its maximal unifiable subsets are determined, one of these subsets is selected, and exact backtracking is performed by removing those deductions which produce constraints not in the selected subset. In what follows, we describe the process of determining the maximal unifiable subsets of a set of constraints.

2: Preliminaries

In this section, we give a few familiar definitions and define notation.

2.1: Graph Theory

Our notation and definitions for the concepts of graph theory follows [4] with a few minor exceptions.

2.1.1: Definition: A directed graph G is an ordered pair $< V(G),E(G) >$ where $V(G)$ is a nonempty set of vertices and $E(G) \subseteq V(G) \times V(G)$ is the set of arcs. If $e = (u,v)$ is an arc then e is said to join u to v, to leave u and enter v, and u and v are called the tail and head of e respectively. We will abbreviate "directed graph" to "digraph".

2.1.2: Definition: A digraph D is a subdigraph of a digraph G if $V(D) \subseteq V(G)$ and $E(D) \subseteq E(G)$.

2.1.3: Definition: If G is a digraph, a directed walk in G of length n $(n \geq 1)$ is a sequence $v_1,e_1,v_2,e_2,\ldots,e_n,v_{n+1}$ whose elements are alternately vertices and arcs, such that $e_i = (v_i,v_{i+1})$ for $1 \leq i \leq n$. v_1 and v_{n+1} are called respectively the origin and terminus of the walk. A directed walk is said to be closed if its origin and terminus are identical. A directed walk is called a directed path if for all i and j $(1 \leq i \leq j \leq n+1)$, $v_i = v_j$ implies either $i = j$ or $i = 1$ and $j = n+1$. A directed cycle is a closed directed path. We will denote the set of arcs on a directed walk W by $E(W)$.

2.1.4: Definition: A labelled digraph G is an ordered triple $< V(G),E(G),I(G) >$ where $V(G)$ is a nonempty set of vertices, $E(G) \subseteq V(G) \times I(G) \times V(G)$ is the set of arcs and $I(G)$

is the set of labels. If $e = (u,\ell,v) \in E(G)$, ℓ is called the label of e. All the concepts defined in 2.1.1 to 2.1.3 are defined in exactly the same way for labelled digraphs.

Since we consider only directed graphs we will usually omit the word "directed" and the prefix "di-", using "graph", "walk", "subgraph", etc. instead of "directed graph", "directed walk", "subdigraph" etc.

2.2: Language

2.2.1: Definition: An alphabet is a triple (V,F,degree) where V,F are mutually disjoint nonempty sets of variables and function symbols respectively, and degree is a function from F to the nonnegative integers. If degree(f) = m for some $f \in F$ we say that f is of degree m.

2.2.2: Definition: An expression over an alphabet is:

either (i) a variable

or (ii) a string of the form f( ) where f is a function symbol of degree 0. Such expressions are called constants. The constant f( ) is usually abbreviated to f.

or (iii) a string of the form $f(p_1,\ldots,p_n)$ where f is a function symbol of degree $n > 0$, and $p_1,\ldots,p_n$ are expressions.

An expression which is not a variable is called a term.

2.2.3: Definition: If p and q are expressions, then q is a subexpression of p if:

either (i) $q = p$

or (ii) q is a subexpression of p_i for some i $(1 \le i \le n)$ where $p = f(p_1,\ldots,p_n)$.

If q is a subexpression of p and q is a term, then q is called a subterm of p.

We use the notions "substitution", "application of substitution" and "unifiability" with their standard meanings. We do, however, extend the latter two of these as follows:

2.2.4: Definition: If E is a set of expressions, $\mathcal{E}$ is a set of sets of expressions and θ is a substitution, then we define the application of θ to E and $\mathcal{E}$ denoted $E\theta$ and $\mathcal{E}\theta$ respectively by:

$$E\theta = \{ e\theta \mid e \in E\}$$
$$\mathcal{E}\theta = \{ F\theta \mid F \in \mathcal{E}\}$$

2.2.5: Definition: If $\mathcal{E}$ is a set of sets of expressions, then $\mathcal{E}$ is said to be unifiable iff there is a substitution θ which unifies every member of $\mathcal{E}$. θ is called a unifier of $\mathcal{E}$.

2.2.6: Definition: A constraint is an unordered pair of expressions. An expression p is a subexpression (or subterm) of a set of constraints C iff there is a constraint $\{q,r\} \in C$ such that p is a subexpression (or subterm) of r.

2.2.7: Definition: If C is a set of constraints and $C_1 \subseteq C$, then C_1 is said to be a maximal unifiable subset of C iff C_1 is unifiable and C_1 is not properly contained in any other unifiable subset of C.

3. The Baxter Unification Algorithm

Our method for finding the maximal unifiable subsets of a set of constraints is based on a unification algorithm due to Baxter [2,3], which operates in two stages. The first (transformational) stage detects unification failure resulting from an attempt to unify subterms beginning with different function symbols, as in P(x,x) and P(a,f(y)) for example. The second (sorting) stage detects failure resulting from an attempt to unify a variable with a term in which it occurs, as in P(x) and P(f(x)).

The transformational stage of the algorithm manipulates two sets: a set S of constraints which is initially C, the input set of constraints, and a set F which is a partition of the set of subexpressions of C. F is initially F_{IN}, the partition in which each class contains one and only one subexpression. This algorithm either halts with unification failure, or halts returning the partition F_{OUT} which is unifiable iff C is unifiable. In what follows, we use $[p]_F$ to denote the class in F which contains the subexpression p of C, and write $p \equiv q \bmod F$ iff $[p]_F = [q]_F$. When F is understood from context, we will write [p] for $[p]_F$.

```
algorithm TRANSFORM(C);
S <-- C;
F <-- F_IN;
while S ≠ ∅
  do  Delete a constraint {p1,p2} from S;
      if [p1] ≠ [p2]
      then  if [p1] contains a term
               f1(q11,...,q1m)
               and [p2] contains a term
               f2(q21,...,q2n)
            then  if f1 ≠ f2
                  then  unification fails;
                        stop
                  else  add to S the pairs:
                        {q11,q21},...,{q1n,q2n}
                  Replace [p1] and [p2] by
                  [p1] ∪ [p2] in F
stop
```

If TRANSFORM(C) succeeds, it outputs a partition F_{OUT} which has the property that any two terms in the same class begin with the same function symbol.

In the sorting stage, a digraph D is constructed such that $V(D) = F_{OUT}$ and E(D) is defined as follows. Suppose F_{OUT} has m classes containing terms, and let $t_1,\ldots,t_m$ be terms representing these classes. Suppose also that $t_i = f_i(p_{i1},\ldots,p_{in_i})$ for $1 \le i \le m$, then:

$$E(D) = \{ ([t_i],[p_{ij}]) \mid 1 \le i \le m,\ 1 \le j \le n_i\}$$

Given F_{OUT}, D is unique (i.e. independent of which terms we choose to represent the classes in F_{OUT}).

3.1: Theorem: A set of constraints C is unifiable iff TRANSFORM(C) succeeds producing partition F_{OUT}, and the digraph D constructed from F_{OUT} has no cycles.

For examples, and proofs of the above results, see [2,3].

4: Detecting and Locating Nonunifiabilities

The unification algorithm described above halts at the first sign of nonunifiability. A set of constraints, however, can be nonunifiable for more than one reason, so since we wish to determine all sources of nonunifiability, we modify the transformational stage so that it continues to merge classes of the partition even though they may contain terms beginning with different function symbols. Similarly, in the sorting stage, we must enumerate cycles rather than simply detect them. See [6] for omitted proofs.

4.1: Transformational Stage

The modified transformational algorithm manipulates three sets: the sets S and F which are the same as in TRANSFORM(C), and a set P which is a partition of the set of subterms of C, and is initially P_{IN}, the partition in which each class has one member. The output partitions are denoted F_{OUT} and P_{OUT}, and as before, C is unifiable iff F_{OUT} is unifiable. In what follows we denote by $\langle t \rangle_p$ the class in P containing the subterm t of C, and write $t_1 \equiv t_2 \bmod P$ iff $\langle t_1 \rangle_p = \langle t_2 \rangle_p$. When no ambiguity is likely, we write $\langle t \rangle$ for $\langle t \rangle_p$.

```
algorithm CLASSIFY(C);
S <-- C;
F <-- F_IN;
P <-- P_IN;
while S ≠ ∅
  do  Delete a constraint {p1,p2} from S;
      if [p1] ≠ [p2]
      then  T <-- [p1];
            while T contains a term
                  t1 = f(q11,...,q1n)
              do  Delete from T all terms in
                  < t1 >;
                  if [p2] contains a term
                      t2 = f(q21,...,q2n)
                  then  Add to S the pairs:
                        {q11,q21},...,{q1n,q2n};
                        Replace < t1 > and
                          < t2 > by
                        < t1 > ∪ < t2 > in P
            Replace [p1] and [p2] by
              [p1] ∪ [p2] in F
stop
```

The following two lemmas state important properties of CLASSIFY, and show a relationship between CLASSIFY and TRANSFORM which will later be strengthened.

4.1.2: Lemma: (a) For any two terms s_1 and s_2, s_1 and s_2 are in the same class of P_{OUT} iff s_1 and s_2 are in the same class of F_{OUT} and s_1 and s_2 begin with the same function symbol.
(b) If two terms $f(p_{11},\ldots,p_{1m})$ and $f(p_{21},\ldots,p_{2m})$ are in the same class of P_{OUT}, then for each i $(1 \le i \le m)$ p_{1i} and p_{2i} are in the same class of F_{OUT}.

4.1.3: Lemma:

(a) If TRANSFORM(C) succeeds returning partition F_{OUT}, then there is an execution of CLASSIFY(C) returning partition F_{OUT}, and in each class of F_{OUT}, all terms begin with the same function symbol.
(b) If TRANSFORM(C) fails, then there is an execution of CLASSIFY(C) returning partition F_{OUT}, where some class of F_{OUT} contains terms beginning with different function symbols.

4.1.4: Example: Consider the set of constraints C as follows:

$¢_1$: { G(s,z), G(v,F(y,y))}
$¢_2$: { u, F(y,G(s,z))}
$¢_3$: { u, F(H(w),G(x,r))}
$¢_4$: { F(H(H(u)),H(u)), F(H(H(v)),v)}
$¢_5$: { v, F(y,y)}

The output partitions of CLASSIFY(C) are:

F_{OUT} = {{ u,v,s,z,x,r,
F(H(w),G(x,r)), F(y,G(s,z)),F(y,y),
H(u), H(v)},
{ H(H(u)), H(H(v))},
{ y
G(s,z),G(v,F(y,y)), G(x,r),
H(w)},
{ F(H(H(u)),H(u))), F(H(H(v)),v)},
{ w} }

P_{OUT} = {{ F(H(w),G(x,r)), F(y,G(s,z)), F(y,y)},
{ H(u), H(v)},
{ H(H(u), H(H(v))},
{ G(s,z), G(v,F(y,y)), G(x,r)},
{ H(w)},
{ F(H(H(u)),H(u)), F(H(H(v)),v)} }

4.2: The Automaton for a Constraint Set

CLASSIFY(C) divides the set of subexpressions of C into classes of expressions which must be unifiable for C to be unifiable: therefore, if two terms t_1 and t_2 begin with different function symbols and occur in the same class, then C is not unifiable. By inspecting each class, we can discover every such pair of incompatible terms. We now introduce a mechanism for determining why two incompatible terms are in the same class: that is, for finding all the constraints responsible for

this situation.

4.2.1: Definition: If C is a set of constraints, denote by M(C) the set of all function symbols occurring in C. We then define:

$$\text{degree}(C) = \max_{f \in M(C)} \text{degree}(f)$$

and $N(C) = \{ i | 1 \le i \le \text{degree}(C)\}$

4.2.2: Definition: If C is a set of constraints, then A(C) is a labelled, directed graph, where:

$V(A(C)) = \{ p | p$ is a subexpression of $C\}$
$I(A(C)) = C \cup (M(C) \times N(C))$
$E(A(C)) = \text{TRANS}(A(C)) \cup \text{PUSH}(A(C)) \cup \text{POP}(A(C))$

where TRANS(A(C)), PUSH(A(C)) and POP(A(C)) are mutually disjoint sets of arcs defined by:

$\text{TRANS}(A(C)) = \{ (p_1, ¢, p_2) | \{p_1, p_2\} = ¢ \in C\}$

$\text{PUSH}(A(C)) = \{ (p,(f,i),t) |$ p and t are subexpressions of C, $t = f(p_1,\ldots,p_n)$, and $p = p_i$ for some $i \in \{1,\ldots,n\}\}$

$\text{POP}(A(C)) = \{ (t,(f,i),p) |$ p and t are subexpressions of C, $t = f(p_1,\ldots,p_n)$, and $p = p_i$ for some $i \in \{1,\ldots,n\}\}$

If $e = (p_1, \text{label}, p_2) \in E(A(C))$, we denote by e^{-1} the ordered triple (p_2, label, p_1). Then $(e^{-1})^{-1} = e$; $e \in \text{TRANS}(A(C))$ if and only if $e^{-1} \in \text{TRANS}(A(C))$; and $e \in \text{PUSH}(A(C))$ if and only if $e^{-1} \in \text{POP}(A(C))$.

Note that A(C) can be regarded as a finite nondeterministic, pushdown automaton [1], where V(A(C)) is the set of states, (which we will also refer to as vertices of A(C), and subexpressions of C), and the transition function is defined in the obvious way by the arcs. The npda A(C) has unspecified initial and final states; input alphabet C; and pushdown alphabet M(C) x N(C), which we will henceforth refer to as Z. Accordingly, we call A(C) the automaton for C, and make the following definitions.

4.2.3: Definition: If X is any finite set, a word of length n over X, is any sequence of elements of X of length n. The word of length 0 is denoted by $\emptyset$; the set of all words of positive length over X by X^+; and the set of all words over X by X^*. We will denote the length of a word x by $|x|$, and denote concatenation of words by juxtaposition.

4.2.4: Definition: If $a \in C^*$, $\gamma \in Z^*$, and $p \in V(A(C))$, then (p,a,γ) is called a configuration of A(C), and p,a and γ are called the state, input and stack of the configuration respectively.

4.2.5: Definition: If $e \in E(A(C))$, we define a relation $\vdash e \dashv$ on the set of configurations of A(C) as follows: let $e = (p_1, \text{label}, p_2)$, then $(q_1,a_1,\gamma_1) \vdash e \dashv (q_2,a_2,\gamma_2)$ if and only if:

(i) $q_1 = p_1$, $q_2 = p_2$

and (ii) (a) if $e \in \text{TRANS}(A(C))$, and label $= ¢ \in C$ then $a_1 = ¢a_2$

$\gamma_1 = \gamma_2$

(b) if $e \in \text{PUSH}(A(C))$, and label $= (f,i) \in Z$ then $a_1 = a_2$

$\gamma_2 = (f,i)\gamma_1$

(c) if $e \in \text{POP}(A(C))$, and label $= (f,i) \in Z$ then $a_1 = a_2$

$\gamma_1 = (f,i)\gamma_2$

4.2.6: Definition: An alternating sequence of n+1 configurations and n arcs of A(C):

$(p_1,a_1,\gamma_1),e_1,(p_2,a_2,\gamma_2),\ldots,(p_n,a_n,\gamma_n),e_n,$
$(p_{n+1},a_{n+1},\gamma_{n+1})$

is called a chain of length n in A(C) from (p_1,a_1,γ_1) to $(p_{n+1},a_{n+1},\gamma_{n+1})$, if and only if:

$(p_i,a_i,\gamma_i) \vdash e \dashv (p_{i+1},a_{i+1},\gamma_{i+1})$ for all $i \in \{1,\ldots,n\}$.

4.2.7: Definition: For each integer $n \ge 0$, we define a relation $\vdash^n$ on the set of configurations of A(C) as follows:

(i) $(p_1,a_1,\gamma_1) \vdash^0 (p_2,a_2,\gamma_2)$ if and only if

$p_1 = p_2$, $a_1 = a_2$, and $\gamma_1 = \gamma_2$

(ii) If $n > 0$, $(p_1,a_1,\gamma_1) \vdash^n (p_2,a_2,\gamma_2)$ if and only if there is a chain of length n in A(C) from (p_1,a_1,γ_1) to (p_2,a_2,γ_2)

We abbreviate $\vdash^1$ as $\vdash$, and also define the relation $\vdash^*$ on the set of configurations of A(C) as follows:

$(p_1,a_1,\gamma_1) \vdash^* (p_2,a_2,\gamma_2)$

iff $(p_1,a_1,\gamma_1) \vdash^n (p_2,a_2,\gamma_2)$ for some $n \ge 0$

When two terms are unified, their corresponding subexpressions are also unified as stated in Lemma 4.1.2 (b). Hence the process of unification of a set of constraints defines an equivalence relation on the subexpression of C characterized by the partition F_{OUT}. As it happens, we can also characterize this equivalence relation by considering certain chains in A(C), and through this characterization, we can show the uniqueness of F_{OUT} and P_{OUT}, and prove the exact relationship between TRANSFORM and CLASSIFY.

4.2.8: Definition: If p and q are two subexpressions of C, then p is said to be attached to q in A(C) if and only if for some $a \in C^*$, $(p,a,\emptyset) \vdash^* (q,\emptyset,\emptyset)$. We denote this $p \sim q$ mod C. We also say that p is attached to q by the word a. It is easy to show that $\sim$ mod C is an equivalence relation.

4.2.9: Lemma: For any set of constraints C, if

F_{OUT} is an output partition of CLASSIFY(C), then $p \equiv q \bmod F_{OUT}$ iff $p \simeq q \bmod C$.

4.2.10: Corollary: The output partitions F_{OUT} and P_{OUT} of CLASSIFY(C) are unique: that is, independent of the choices made during execution. Because of this result, we henceforth refer to F_{OUT} and P_{OUT}, output by CLASSIFY(C), as $F_{OUT}(C)$ and $P_{OUT}(C)$.

Proof: The uniqueness of F_{OUT} is obvious from lemma 4.2.9, and implies the uniqueness of P_{OUT} by lemma 4.1.2 (a). □

4.2.11: Corollary: TRANSFORM(C) succeeds, returning partition F_{OUT} if and only if CLASSIFY(C) returns partition F_{OUT} where each class of F_{OUT} contains at most one class of P_{OUT}.

Proof: By applying corollary 4.2.10 to lemma 4.1.3. □

Note that corollary 4.2.11 establishes the uniqueness of the output of TRANSFORM(C) and of the digraph D, henceforth denoted D(C).

4.2.12: Example: The automaton for the set of constraints of example 4.1.4 is illustrated in figure 1. By inspecting the classes of $F_{OUT}(C)$ given in example 4.1.4, we see that $x \equiv u \bmod F_{OUT}$, so by lemma 4.2.9, $x \simeq u \bmod C$. Figure 2 shows a chain demonstrating this attachment.

4.3: The Unification Graph for C

Recall that if the transformational stage of the Baxter algorithm succeeds, the resulting partition is used to construct a digraph which must be topologically sorted. Similarly, from the output partition $F_{OUT}(C)$ of CLASSIFY(C), we construct a labelled digraph U(C).

4.3.1: Definition: If C is a set of constraints, the unification graph U(C) for C, is a labelled, directed graph, where:

$V(U(C)) = F_{OUT}(C)$

$I(U(C)) = M(C)$ (definition 4.2.1)

and the arc set is defined as follows. Suppose there are m classes in $P_{OUT}(C)$, and let $t_1,\ldots,t_m$ be m terms such that $\langle t_i \rangle = \langle t_j \rangle$ if and only if $i = j$. Suppose $t_i = f_i(p_{i1},\ldots,p_{in_i})$ for $1 \le i \le m$, then:

$$E(U(C)) = \{ ([t_i],f_i,[p_{ij}]) \mid 1\le i\le m,\ 1\le j\le n_i\}$$

Note that U(C) is unique in view of lemma 4.1.2 (b).

4.3.2: Lemma: If TRANSFORM(C) succeeds, then D(C) has a cycle if and only if U(C) has a cycle.

We may now prove the correctness of our modified version of Baxter's algorithm.

4.3.3: Theorem: A set of constraints C is unifiable if and only if every class of $F_{OUT}(C)$ contains at most one class of $P_{OUT}(C)$, and U(C) has no cycles.

Proof: By theorem 3.1, C is unifiable if and only if TRANSFORM(C) succeeds and D(C) has no cycles.

By corollary 4.2.11 and lemma 4.3.2, TRANSFORM(C) succeeds and D(C) has no cycles if and only if every class of $F_{OUT}(C)$ contains at most one class of $P_{OUT}(C)$, and U(C) has no cycles. □

4.3.4: Example: Figure 3 illustrates the unification graph for the set of constraints C of example 4.1.4. Note that by Theorem 4.3.3, C fails to be unifiable for two reasons: some classes of F_{OUT} contain more than one class of P_{OUT}, and U(C) has cycles.

4.4: The Failure Location Process

Theorem 4.3.4 allows us to determine whether or not a set of constraints is unifiable: in the case when they are not, it also shows us all the sources of nonunifiability due to clashes between incompatible terms, and due to cycles. We now show how this information can be used to guide the investigation of the automaton in order to determine which constraints cause these nonunifiabilities.

We have already established that attachment in the automaton corresponds to equivalence under the output partition F_{OUT} of CLASSIFY. By finding words which attach incompatible terms, therefore, we can find those subsets of the constraints which cause these terms to be attached.

In this section, we also find that there is a similar correspondence between cycles in the unification graph, and "loops" in the automaton; where loops are chains roughly equivalent to closed walks in a directed graph.

There is in general an infinite number of chains demonstrating the attachment of a pair of terms, and an infinite number of loops. However, it turns out that we can limit our attention to chains and loops with certain properties, and that the class of such chains is finite.

4.4.1: Definition: A chain from a configuration $(p,a,\emptyset)$ to a configuration $(p,\emptyset,\gamma)$ where $\gamma\neq\emptyset$, is called a loop on p with value a in A(C).

The next three results establish the relationship between walks in U(C) and chains and loops in A(C).

4.4.2: Lemma: If C is a set of constraints and U(C) contains a closed walk from [p] to [p], then there is a loop on p in A(C).

As one travels along a chain, the automaton's stack increases and decreases a number of times before it reaches its final value. For each element on the final stack, however, there is some configuration on the loop which is reached from the previous configuration by adding this stack element, and is such that the stack is no shorter in any subsequent configuration. The states of these "plateau" configurations are the subject of the next lemma.

4.4.3: Lemma: If $(p_1,a_1,\gamma_1),e_1,\ldots,e_n,$ $(p_{n+1},a_{n+1},\gamma_{n+1})$ is a chain in $A(C)$ such that $\gamma_1 = \phi$, and $\gamma_{n+1} = (f_m,j_m)\ldots(f_1,j_1)$, then there are m unique states $q_1,\ldots,q_m$, m stacks $\alpha_1,\ldots,\alpha_m$, and m integers $i_1,\ldots,i_m$ such that:

(1) $1 < i_1 < i_2 < \ldots < i_m \leq n+1$

(2) $q_j = p_{i_j}$, $\alpha_j = \gamma_{i_j}$ for $1 \leq j \leq m$

(3) q_j is a term beginning with f_j, for $1 \leq j \leq m$

(4) $|\gamma_k| \geq |\alpha_j|$ for $k \geq i_j$, $1 \leq j \leq m$

(5) if $m > 1$, $([q_j],f_j,[q_{j-1}] \in E(U(C))$ for all $2 \leq j \leq m$

(6) $[q_m] = [p_{n+1}]$

(7) $([q_1],f_1,[p_1]) \in E(U(C))$

An obvious consequence of lemmas 4.4.2 and 4.4.3 is:

4.4.4: Corollary: $U(C)$ has a closed walk iff $A(C)$ has a loop.

4.4.5: Definition: If $A(C)$ has a loop $(p_1,a_1,\gamma_1),e_1,\ldots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$, where $|\gamma_{n+1}| = m$, then the m states $q_1,\ldots,q_m$ defined in lemma 4.4.3 are called the characteristic states of the loop. For each characteristic state q_k, we define a loop on q_k called the q_k-canonical form of the original loop, as follows. Suppose $q_k = p_j$, then $|\gamma_i| \geq |\gamma_j|$ for $i \geq j$. Therefore, for each $i \geq j$, we can write γ_i in the form $\beta_i\gamma_j$ for some $\beta_i \in Z^*$. Also, $a_1 = ca_j$ for some $c \in C^*$. Then the q_k-canonical form of the original loop is:

$(p_j,a_jc,\beta_j),e_j,\ldots,e_n,(p_{n+1},c,\beta_{n+1}),e_1,$
$(p_2,c,\gamma_2\beta_{n+1}),e_2,\ldots,e_{j-1},(p_j,\emptyset,\gamma_j\beta_{n+1}),$

which is a loop since $\beta_j = \emptyset$, and $\gamma_j\beta_{n+1} \neq \emptyset$ (since $\gamma_{n+1} = \beta_{n+1}\gamma_j$ and $\gamma_{n+1} \neq \emptyset$).

Note that:

$\{ \text{ç} | \text{ç} \text{ occurs in } a_1\} = \{ \text{ç} | \text{ç} \text{ occurs in } a_jc\}$

We now turn our attention to certain restricted classes of chains and loops.

4.4.6: Definition: A chain in $A(C)$: $(p_1,a_1,\gamma_1),e_1,\ldots,e_n,(p_{n+1},a_{n+1},\gamma_{n+1})$ is said to be semi-simple if and only if for all i and j such that $1 \leq i < j \leq n+1$, either $p_i \neq p_j$ or $\gamma_i \neq \gamma_j$. The above chain is said to be simple if and only if it is semi-simple, and for all i and j such that either $1 \leq i < j < n+1$ or $1 < i < j \leq n+1$:

if $p_i = p_j$

then $\exists k \in \{i+1,\ldots,j-1\}$ such that:

$|\gamma_k| < |\gamma_i|$

and $|\gamma_k| < |\gamma_j|$

4.4.7: Definition: A loop in $A(C)$ is said to be fundamental iff all its canonical forms are simple.

4.4.8: Definition: If p and q are subexpressions of C and there is a simple chain from $(p,a,\emptyset)$ to $(q,\emptyset,\emptyset)$ for some $a \in C^*$, then p is said to be simply attached to q by a in $A(C)$. It is easy to verify that simple attachment is an equivalence relation. The reader should note that simple attachment implies attachment but not vice versa.

It happens that the number of chains demonstrating that two expressions are simply attached is finite, and that the number of simple loops in $A(C)$ is also finite. These facts are consequences of the following lemma:

4.4.9: Lemma: Let Δ be the set of all simple chains in $A(C)$ for which the stack of the initial configuration is $\emptyset$, and the input of the final configuration is $\emptyset$. Then Δ is finite.

The following result shows that we can indeed restrict our attention to simple chains and fundamental loops when we wish to separate incompatible terms and break cycles.

4.4.10: Lemma:

(a) If there is a chain of length n in $A(C)$ from (p,a,γ) to (q,b,α), where either $p \neq q$ or $\gamma \neq \alpha$, then there is a semi-simple chain of length $\leq n$ in $A(C)$ from (p,c,γ) to (q,b,α), for some $c \in C^*$.

(b) If $A(C)$ has a semi-simple chain of length n that is not simple, then $A(C)$ has a simple loop of length $< n$.

(c) If $A(C)$ has a loop, then $A(C)$ has a fundamental loop.

We now have all the machinery necessary to describe the process of finding the maximal unifiable subsets of a set C of constraints. Informally, the process is as follows.

We find all pairs of incompatible terms which are in the same class of $F_{OUT}(C)$, and find all cycles of $U(C)$ [8,9,11]. We then find all words which simply attach incompatible terms, and for each cycle find the value of a corresponding fundamental loop.

4.4.11: Definition: If L is any finite set, denote by $\mathcal{B}(L)$, the set of all Boolean expressions over L constructed without complementation. If $B \in \mathcal{B}(L)$, denote by $[B]$ the function from $2^L \dashrightarrow \{0,1\}$ defined by:

$[0](L_1) = 0$ for all $L_1 \subseteq L$

$[1](L_1) = 1$ for all $L_1 \subseteq L$

$[\ell](L_1) = 0$ iff $\ell \in L_1$

$[B_1+B_2](L_1) = [B_1](L_1) + [B_2](L_1)$

$[B_1 \cdot B_2](L_1) = [B_1](L_1) \cdot [B_2](L_1)$

4.4.12: Definition: If C is a set of constraints we define several sets as follows:

(i) If p and q are subexpressions of C:
ATTACH(p,q) = {a|p is simply attached to q by a}

(ii) CONFLICT = {{p,q}|[p] = [q] and <p> ≠ <q>}

(iii) For any subexpression p of C:
LOOP(p) = {a|∃ a simple loop on p with value a}

(iv) For any arc $e \in E(U(C))$:
TAIL(e) = {t|t begins with f, where e = ([t],f,[p])}

(v) CIR = set of all cycles of $U(C)$

(vi) Let H be the assertion defined by:

$H(\mathcal{E})$ iff for all $k \in$ CIR, $\mathcal{E} \cap E(k) \neq \emptyset$

Then let COVER be any subset of $E(U(C))$ satisfying the condition:

$$|\text{COVER}| = \min_{\substack{\mathcal{E} \subseteq E(U(C)) \\ \text{and } H(\mathcal{E})}} |\mathcal{E}|$$

Clearly if CIR $= \emptyset$, COVER $= \emptyset$.

4.4.13: Definition: We now define several Boolean expressions over C as follows:

(i) If $a \in C^+$, $B_w(a) = \sum_{¢ \text{ occurs in } a} ¢$

(ii) If p and q are distinct subexpressions of C:

$$B_A(p,q) = \begin{cases} 1 & \text{if ATTACH}(p,q)=\emptyset \\ \prod_{a \in \text{ATTACH}(p,q)} B_w(a) & \text{otherwise} \end{cases}$$

It is easy to show that if $a \in$ ATTACH(p,q) then $a \neq \emptyset$, so that $B_w(a)$ is defined. Also, by lemma 4.4.9, ATTACH(p,q) is finite.

(iii) $$B_{CON} = \begin{cases} 1 & \text{if CONFLICT}=\emptyset \\ \prod_{\{p,q\} \in \text{CONFLICT}} B_A(p,q) & \text{otherwise} \end{cases}$$

(iv) For any subexpression p of C:

$$B_L(p) = \begin{cases} 1 & \text{if LOOP}(p) = \emptyset \\ \prod_{a \in \text{LOOP}(p)} B_w(a) & \text{otherwise} \end{cases}$$

Again, it is easy to show that if $a \in$ LOOP(p), then $a \neq \emptyset$, so that $E(a)$ is defined; and by lemma 4.4.9, LOOP(p) is finite.

(v) For any $e \in E(U(C))$:

$$B_T(e) = \prod_{t \in \text{TAIL}(e)} B_L(t)$$

Note that TAIL$(e) \neq \emptyset$.

(vi) $$B_{CYC} = \begin{cases} 1 & \text{if COVER} = \emptyset \\ \prod_{e \in \text{COVER}} B_T(t) & \text{otherwise} \end{cases}$$

(vii) $B_{UNIF} = B_{CON} \cdot B_{CYC}$

4.4.14: Lemma: If $C_1 \subseteq C$, and $a \in C^+$, then: $[B_w(a)](C_1) = 0$ if and only if $a \in C_1^+$.

4.4.15: Lemma: If $C_1 \subseteq C$, and $A(C_1)$ either has a loop, or has a chain that is semi-simple but not simple, then $[B_{UNIF}](C_1) = 0$.

Proof: If $A(C_1)$ has a semi-simple chain that is not simple, by lemma 4.4.10(b), $A(C_1)$ has a loop.

If $A(C_1)$ has a loop, then by lemma 4.4.10(c), $A(C_1)$ has a fundamental loop. Suppose this loop has value $a \in C_1^+$. Since $A(C_1)$ is a subgraph of $A(C)$, this loop is also in $A(C)$. Let $q_1,\ldots,q_m$ be the characteristic states of this loop, then by lemma 4.4.3, there is a closed walk $[q_1],e_1,[q_2],\ldots,[q_m],e_m,[q_1]$ in $U(C)$. Either this walk is a cycle, or some subset of its arcs form a cycle. In either case, for some $j \in \{1,\ldots,m\}$, $e_j \in$ COVER and $q_j \in$ TAIL(e_j) by lemma 4.4.3 condition (3). Since the loop in $A(C)$ is fundamental, its q_j-canonical form is simple, so that $B_w(b)$ is a factor of the product B_{UNIF} where b is the value of this canonical form. Also:

$\{¢ | ¢ \text{ occurs in } b\} = \{¢ | ¢ \text{ occurs in } a\} \subseteq C_1$

so that $[B_w(b)](C_1) = 0$, by lemma 4.4.14

$\therefore$ $[B_{UNIF}](C_1) = 0$ □

4.4.16: Lemma: If $C_1 \subseteq C$, then:

$[B_{UNIF}](C_1) = 1$ iff C_1 is unifiable.

Proof:

(A) Suppose C_1 is not unifiable, then by theorem 4.3.3 we have two cases:

case(a): There exist subexpressions p and q such that $p \equiv q \bmod F_{OUT}(C_1)$ and $p \not\equiv q \bmod P_{OUT}(C_1)$. By lemma 4.2.9, $p \simeq q \bmod C_1$, so there exists a chain in $A(C_1)$ from $(p,a,\emptyset)$ to $(q,\emptyset,\emptyset)$ for some $a \in C_1^*$, so by lemma 4.4.10(a), there is a semi-simple chain in $A(C_1)$ from $(p,b,\emptyset)$ to $(q,\emptyset,\emptyset)$. Note that $b \in C_1^*$. We have two cases:

(i) Suppose this chain is simple. Since $A(C_1)$ is a subgraph of $A(C)$ the chain is in $A(C)$; and by lemma 4.1.2(a) since p and q begin with different function symbols, $p \not\equiv q \bmod P_{OUT}(C)$

$\therefore$ $\{p,q\} \in$ CONFLICT
and $b \in$ ATTACH(p,q)

Therefore $B_w(b)$ is a factor in the product B_{UNIF}. But $b \in C_1^*$, so by lemma 4.4.14:

$[B_w(b)](C_1) = 0$

$\therefore$ $[B_{UNIF}](C_1) = 0$

(ii) If this chain is not simple, then by lemma 4.4.15:

$[B_{UNIF}](C_1) = 0$

case(b): $U(C_1)$ has a cycle. In this case, by corollary 4.4.4, $A(C_1)$ has a loop, so by lemma 4.4.15:

$[B_{UNIF}](C_1) = 0$

(B) Now suppose that $[B_{UNIF}](C_1) = 0$. Then we have two cases:

case(a): $[B_{CYC}](C_1) = 0$.

In this case, for some subexpression p of C, there is a simple loop in $A(C)$ on p with value $a \in C^*$, such that $[B_w(a)](C_1) = 0$. It is easy to show that $a \neq \emptyset$, therefore $a \in C^+$, so by

lemma 4.4.14 $a \in C_1^*$, so that the simple loop on p is in $A(C_1)$. Consequently, $U(C_1)$ has a cycle (corollary 4.4.4), so that by theorem 4.3.3 C_1 is nonunifiable.

case(b): $[B_{CON}](C_1) = 0$.
In this case, for some subexpressions p and q of C, $p \equiv q \bmod F_{OUT}(C)$, $p \not\equiv q \bmod P_{OUT}(C)$, and p is simply attached to q by $a \in C^*$ in $A(C)$, where $[B_w(a)] = 0$. It is easy to show that $a \neq \emptyset$, therefore $a \in C^+$, so by lemma 4.4.14, $a \in C_1^*$ so that p is simply attached to q by a in $A(C_1)$. Hence $p \simeq q \bmod C_1$ so by lemma 4.2.9, $p \equiv q \bmod F_{OUT}(C_1)$. But p and q begin with different function symbols by lemma 4.1.2(a), so by the same lemma $p \not\equiv q \bmod P_{OUT}(C_1)$. Therefore by theorem 4.3.3, C_1 is nonunifiable. □

If B is a Boolean expression constructed without complementation, then there exists [5] a unique (modulo commutativity of Boolean sum and product), sum of products expression B' with the properties:

(a) No product in B' subsumes any other product in B'.
(b) No product in B' contains repeated variables.
(c) B' defines the same Boolean function as B.

Also, for any two Boolean expressions B_1 and B_2, $[B_1] = [B_2]$ if and only if B_1 and B_2 define the same Boolean function.
We may now prove the main result.

4.4.17: Theorem: C_1 is a maximal unifiable subset of a set of constraints iff $C_1 = C - \{ ¢_1,\ldots,¢_n\}$ where $¢_1 \ldots ¢_n$ is a product in B'_{UNIF}.

Proof:

(A) Suppose $C_1 = C - \{¢_1,\ldots,¢_n\}$ where $¢_1\ldots¢_n$ is a product in B'_{UNIF}.
Then $[¢_1\ldots¢_n](C_1) = \prod_{i=1}^{n}[¢_i](C_1)$

$= 1$ since $¢_i \notin C_1$ for $1 \le i \le n$

$\therefore [B'_{UNIF}](C_1) = 1$

$\therefore C_1$ is unifiable (lemma 4.4.16).

Now suppose $C_1 \subseteq C_2$, and C_2 is unifiable, then there exists a product $m_1\ldots m_k$ in B'_{UNIF} such that:

$[m_1\ldots m_k](C_2) = 1$

$\therefore m_i \notin C_2$ for $1 \le i \le k$

$\therefore m_i \notin C_1$ for $1 \le i \le k$

$\therefore \{m_1,\ldots,m_k\} \subseteq \{¢_1,\ldots,¢_n\}$

If these sets are not equal, then the product $m_1\ldots m_k$ subsumes the product $¢_1\ldots¢_n$, which is impossible. Therefore $\{m_1,\ldots,m_k\} = \{¢_1,\ldots,¢_n\}$, so that $C_1 = C_2$.
$\therefore C_1$ is a maximal unifiable subset.

(B) Suppose C_1 is a maximal unifiable subset of C. Let $C_1 = C - \{¢_1,\ldots,¢_n\}$. Now $[B'_{UNIF}](C_1) = 1$, since C_1 is unifiable. Therefore there exists a product $m_1\ldots m_k$ in B'_{UNIF} such that:

$\prod_{i=1}^{k}[m_i](C_1) = 1$

$\therefore m_i \notin C_1$ for $1 \le i \le k$

$\therefore \{m_1,\ldots,m_k\} \subseteq \{¢_1,\ldots,¢_n\}$.

Let $C_2 = C - \{m_1,\ldots,m_k\}$. Then C_2 is unifiable by (A), and $C_1 \subseteq C_2$. But C_1 is a maximal unifiable subset, so that $C_2 = C_1$

$\therefore ¢_1\ldots¢_n$ is a product in B'_{UNIF}. □

4.4.18: Example: For the set of constraints C of example 4.1.4, by investigating $F_{OUT}(C)$ and $P_{OUT}(C)$, we find that the set of incompatible terms is:

CONFLICT = {{ F(H(w),G(x,r)), H(u)},
{ F(H(w),G(x,r)), H(v)},
{ F(y,G(s,z)), H(u)},
{ F(y,G(s,z)), H(v)},
{ F(y,y), H(u)},
{ F(y,y), H(v)},
{ G(s,z), H(w)},
{ G(v,F(y,y)),H(w)},
{ G(x,r), H(w)} }

The set of cycles of U(C) is:

CIR = { ([u],e_1,[y],e_2,[u]), ([u],e_y,[u])}

Of the two possible "coverings" for CIR, we choose:

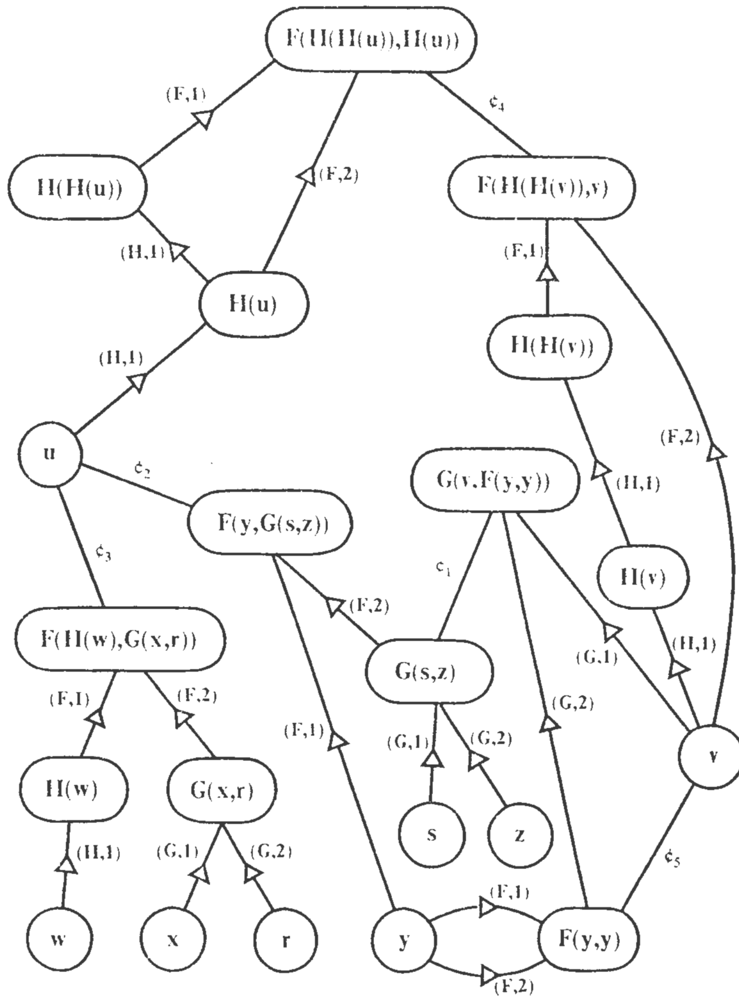
COVER = { e_1,e_7}

So the sets of states of A(C) which must be investigated for loops is:

TAIL(e_1) = { F(y,y), F(H(w),G(x,r)), F(y,G(s,z))}
TAIL(e_7) = { H(u), H(v)}

By investigating the automaton A(C) (figure 1) we obtain the following:

ATTACH(F(H(w),G(x,r)),H(u)) = { $¢_3¢_4¢_4$}
ATTACH(F(H(w),G(x,r)),H(v)) = { $¢_3¢_4¢_4¢_4$}
ATTACH(F(y,G(s,z)),H(u)) = { $¢_2¢_4¢_4$}
ATTACH(F(y,G(s,z)),H(v)) = { $¢_2¢_4¢_4¢_4$}
ATTACH(F(y,y),H(u)) = { $¢_4$}
ATTACH(F(y,y),H(v)) = { $¢_4¢_4$}
ATTACH(G(s,z),H(w)) = { $¢_2¢_4¢_5¢_5¢_4¢_3$, $¢_2¢_4¢_5¢_2¢_3$}
ATTACH(G(v,F(y,y)),H(w)) = { $¢_3¢_2¢_5¢_4¢_2¢_1$, $¢_3¢_4¢_5¢_5¢_4¢_2¢_1$}
ATTACH(G(x,r),H(w)) = { $¢_3¢_4¢_5¢_5¢_4¢_3$, $¢_3¢_2¢_5¢_4¢_3$}



The automaton A(C) for the set of constraints C of our example. Note that to simplify the diagram we have used a single arc to represent a pair of arcs e, e^{-1} of the automaton.

Figure 1.

State	Stack	Input
x	∅	$¢_3¢_2¢_1¢_4$
G(x,r)	(G,1)	$¢_3¢_2¢_1¢_4$
F(H(w),G(x,r))	(F,2)(G,1)	$¢_3¢_2¢_1¢_4$
u	(F,2)(G,1)	$¢_2¢_1¢_4$
F(y,G(s,z))	(F,2)(G,1)	$¢_1¢_4$
G(s,z)	(G,1)	$¢_1¢_4$
G(v,F(y,y))	(G,1)	$¢_4$
v	∅	$¢_4$
H(v)	(H,1)	$¢_4$
H(H(v))	(H,1)(H,1)	$¢_4$
F(H(H(v)),v)	(F,1)(H,1)(H,1)	$¢_4$
F(H(H(u)),H(u))	(F,1)(H,1)(H,1)	∅
H(H(u))	(H,1)(H,1)	∅
H(u)	(H,1)	∅
u	∅	∅

Figure 2.

$$LOOP(F(y,y)) = \{ ¢_1¢_2¢_4¢_5 \}$$
$$LOOP(F(H(w),G(x,r))) = \emptyset$$
$$LOOP(F(y,G(s,z)) = \{ ¢_2¢_4¢_5¢_1 \}$$
$$LOOP(H(u)) = \{ ¢_5¢_1¢_2,\ ¢_1¢_2 \}$$
$$LOOP(H(v)) = \{ ¢_4 \}$$

We obtain the Boolean sum of products over C:

$$B'_{UNIF} = ¢_4¢_1 + ¢_4¢_2$$

Therefore C has two maximal unifiable subsets, namely:

$$\{ ¢_2,\ ¢_3,\ ¢_5 \}$$
and
$$\{ ¢_1,\ ¢_3,\ ¢_5 \}$$

References

[1] Aho, A.V. and Ullman J.D.
The Theory of Parsing, Translation, and Compiling. Volume I: Parsing
Prentice-Hall (1972).

[2] Baxter, L.D.
A Practically Linear Unification Algorithm
Research Report CS-76-13, Department of Computer Science, University of Waterloo (1976).

[3] Baxter, L.D.
The Complexity of Unification
Ph.D. Thesis, Department of Computer Science, University of Waterloo (1976).

[4] Bondy, J.A. and Murty, U.S.R.
Graph Theory with Applications
MacMillan (1976).

[5] Brzozowski, J.A. and Yoeli, M.
Digital Networks
Prentice-Hall (1976).

[6] Cox, P.T.
Deduction Plans: a graphical proof procedure for the first-order predicate calculus
Ph.D. Thesis, Department of Computer Science, Research Report CS-77-28, University of Waterloo (1977).

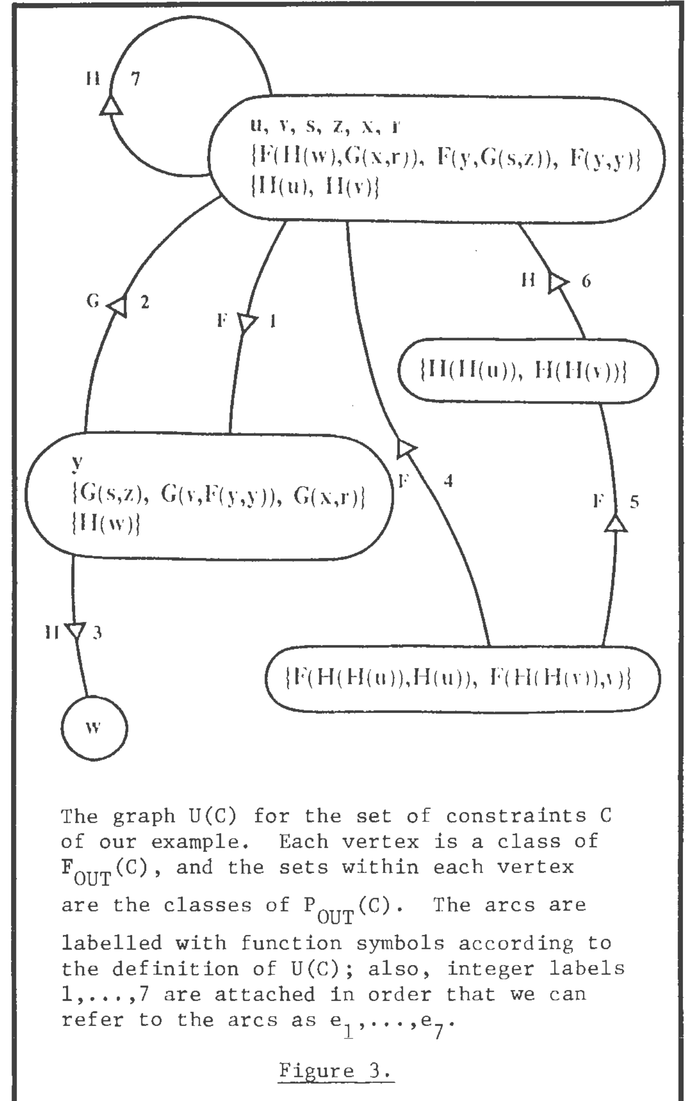
[7] Cox, P.T.
A graphical proof procedure for first-order logic
Proceedings of A Conference on Theoretical Computer Science, University of Waterloo, Waterloo, Ontario, (August 1977).

[8] Johnson, D.E.
Finding all the Elementary Circuits of a Directed Graph
Technical Report 145, Computer Science Department, Pennsylvania State University, (1973).

[9] Read, R.C. and Tarjan, R.E.
Bounds on Backtrack Algorithms for listing cycles, paths and spanning trees
Memo ERL-M433, Electronics Research Laboratory, College of Engineering, University of California, Berkeley (1973).

[10] Robinson, J.A.
A machine oriented logic based on the resolution principle
J.ACM 12, no. 1, 23-41 (1965).

[11] Szwarcfiter, J.L. and Lauer, P.E.
A New Backtracking Strategy for the Enumeration of the Elementary Cycles of a Directed Graph
Technical Report 69, Computing Laboratory, University of Newcastle upon Tyne (1975).



The graph U(C) for the set of constraints C of our example. Each vertex is a class of $F_{OUT}(C)$, and the sets within each vertex are the classes of $P_{OUT}(C)$. The arcs are labelled with function symbols according to the definition of U(C); also, integer labels 1,...,7 are attached in order that we can refer to the arcs as $e_1,\ldots,e_7$.

Figure 3.

An Analysis of Theorem Proving by Covering Expressions

L. J. Henschen
W. M. Evangelist
Northwestern University

Abstract We present a technique for dealing with constrained-variable instances of clauses similar to Gilmore's method. Rather than perform saturation over H, however, we show how the basic calculation can help guide the exploration of H. The theorem proving process then consists of 1) forming some constrained copies of input clauses, 2) calculating a cover expression and 3) if a proof has not been found, using the cover expression to pick new instances of clauses. The last part is the main point of departure from prior constrained variable methods. We also present a complexity analysis of covering and relate an incomplete ground case strategy to a form of unit resolution.

I. Introduction

We present a new method for automated theorem proving that is currently being studied with an interactive program. The method is equivalent to Gilmore's method [6] in the ground case and is related to Prawitz' method [12] in the general case. Some of the interesting features are:

A. for the ground case

i. each clause is used once in the calculation of a covering expression and then discarded, unlike resolution and matrix methods where a clause may have to be used for several resolutions, extensions or matrix reductions;

ii. for a satisfiable set of ground clauses the result is a concise description of the kinds of clauses that would have to be added to obtain unsatisfiability; (this point is crucial for the general case);

iii. the ground clauses can be used in any order; there is never any backing up or undoing of a calculation that has already been performed, and thus one can say there is no "search" involved.

B. for the general case

i. replicas of the input clauses are used (i.e., constrained variable copies [11]); this gives a momentarily fixed set of literals so that the ground cover calculation can be performed;

ii. after a replica has participated in the basic cover calculation, only the literals need be saved for possible future matings; no record of which literals belong to which replicas need be kept; only one copy of a literal need be saved regardless of how many replicas it occurs in;

iii. in relation to remark A. ii above, when not enough replicas have been taken to form an unsatisfiable set of instances, the cover expression is used to help choose new replicas and substitutions. We point out that the search process in the general case is concerned solely with finding the unsatisfiable instances. Once these have actually been found, remark A.iii from above applies. Resolution, on the other hand, combines the search for instances (unification) with a search for a demonstration of unsatisfiability (forming a resolvent).

In [6], Gilmore proposed calculating the DNF of $C_1 \& C_2 \ldots \& C_n$ for a set $C_1, \ldots, C_n$ of clauses. He then proposed to form successive saturations of this DNF matrix over the Herbrand Universe, H. Among the more serious disadvantages of this approach are that 1) the normal conversion from CNF to DNF leads to redundancy among the disjuncts and 2) there is generally little guidance in searching through H. To help overcome these disadvantages, we propose starting with an initial set of instances, performing a different but similar transformation and then using the result to help choose the next set of replicas. Note that the new approach leaves the input clauses as clauses, the often more natural form. Second, the modified conversion will produce DNF's which are minimal in the sense that no individual disjunct is implied by remaining ones. Finally, we make heavy use of the fact that the disjuncts of the modified DNF represent a minimal set of clauses which, if added to the current set of replicas, will produce unsatisfiability. Moreover, the representation is in a form that can be compared directly to the input clauses to determine appropriate new instances.

We assume the reader is familiar with first-order logic and resolution. We begin with the ground case and describe the cover calculation. We then discuss how we currently use the covering idea in a first-order, interactive theorem prover. Finally we make a brief comment about the computational complexity of the ground case. In what follows, we use a slightly different notation than DNF, namely we reverse the signs and don't write the literals themselves. The sign reversal allows a more direct comparison of the cover expressions with input clauses.

II. The Ground Case

Let a set S of ground clauses be given and let the atoms in S be $p_1,\ldots,p_n$. A term is a string of n symbols, each symbol either +, -, or 0. An expression is a set of terms. The covering expression for a clause $D = d_{i_1} \vee d_{i_2} \vee \ldots \vee d_{i_m}$, $1 \le i_j \le n$, is the set of terms

$000s'_{i_1}$ 0000.... 00

$000s_{i_1}$ $0\ldots0s'_{i_2}$ $0\ldots0$

$000s_{i_1}$ $0\ldots0s_{i_2}$ $0\ldots0s'_{i_3}$ $0\ldots0$

.

.

.

$000s_{i_1}$ $0\ldots s_{i_2}$ $0\ldots \ldots \ldots s_{i_{m-1}}$ $0\ldots0\ s'_{i_m}$ $0\ldots0$

where s_j is the sign of d_j and s'_j is the opposite sign. Note that $-(d_{i_1} \vee \ldots \vee d_{i_n})$ is equivalent to:

$$-d_{i_1} \vee d_{i_1}\&-d_{i_2} \vee \ldots \vee d_{i_1}\&\ldots d_{i_{m-1}}\&-d_{i_m}.$$

The covering expression, then, contains one term to represent each of the above disjuncts. The cover expression of a clause D is denoted by C(D).

Examples. Let n = 6

$C(p_2)$ = 0-0000

$C(-p_1 \vee p_6)$ = +00000 -0000-

$C(p_1 \vee -p_3 \vee p_4)$ = -00000 +0+000 +0--00

Definitions Two terms conflict if one contains a + and the other a - in the same position. If two terms s and t do not conflict, then their intersection s*t is the term calculated position by position according to the following: +*+ = +, -*- = -, +*0 = + and -*0 = -. The product s*t of two covering expressions S and T is the set of all terms u such that u = s*t, s in S and t in T and s and t do not conflict.

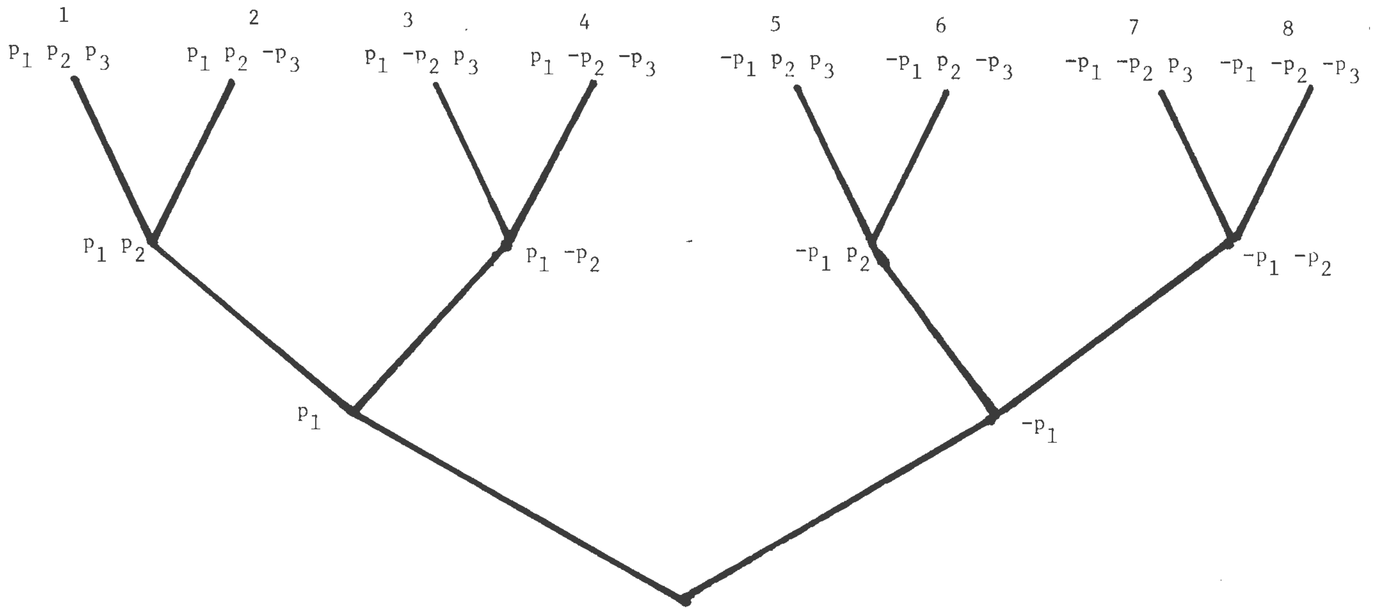
Example. Let C be the clause $p_1 \vee p_3$ and let D be the clause $-p_3 \vee p_4$. Then C(C) = -000 +0-0 and C(D) = 00+0 00-- and C(C)*C(D) = -0+0 -0-- +0-- (+0-0 and 00+0 conflict). It often happens that two terms in an expression are exactly alike except in one position, for example ++-0++ and +++0++. In such cases, the pair of terms are replaced by the single term with a 0 in the disagreeing position, e.g., ++00++. Such an operation is called a merge and is equivalent to applying the rule p&A v-p&A ≡ A. We note that * is commutative and associative. We therefore, extend the notion of a cover expression so that the cover of a set $S = \{C_1, \ldots C_n\}$ is given by $C(S) = C(C_1)*C(C_2)*\ldots*C(C_n)$.

We now relate covering terms and expressions to a tree structure which we call the standard tree (of order n). We are still considering sets of clauses over the Boolean variables $p_1, p_2, \ldots p_n$.

Definition A standard tree of order n is the binary branching tree of depth n in which

1. the clauses attached to the two level 1 nodes are p_1 and $-p_1$ and
2. if C_i is the clause attached to a level i node n then the clauses attached to the two level i+1 nodes descended from n are $C_i \vee p_{i+1}$ and $C_i \vee -p_{i+1}$.

For example, the standard tree of order 3 is shown below. The leaf nodes have been numbered for reference.



Definition The set of leaves covered by a term t is the set of all leaves of the standard tree whose attached clauses contain p_i if the symbol in position i of t is + and $-p_i$ if the symbol in position i is -. The set of leaves covered by an expression E is the union of the set of leaves covered by the terms of E, and is denoted by K(E).

Example. Suppose n=3. Then relative to the above figure

K(+0+) consists of leaves 1 and 3
K(00+) consists of leaves 1, 3, 5, and 7
K(+-- -00) consists of leaves 4 thru 8.

Note that the covering terms are like the "pebbles" used by Paterson and Hewitt [11] to close off a tree.

We have the following theorems:

Theorem 1. Let S be a set of clauses whose atoms are among $p_1,\ldots,p_n$. S is unsatisfiable if and only if S subsumes all the leaves of the standard tree of order n.

Theorem 2. The set of leaves covered by the intersection of two terms s and t is the intersection of the set covered by s and the set covered by t.

Theorem 3. The set of leaves covered by C(D) where D is a clause is the set of leaves NOT subsumed by D.

Theorem 4. The set of leaves covered by C(D)*C(E) is the set of leaves not subsumed by either D or E.

Theorem 5. S is unsatisfiable if and only if C(S) is empty.

We now reiterate one of the main features of ground covering. From Theorems 1 and 4 we have that if S is satisfiable, C(S) tells which leaves of the tree have been left unsubsumed--that is, in a sense, what kinds of clauses are missing from S. This will play an important role in the next section.

We close this section with an example of a set of clauses and the calculation of its cover. PRODUCT COVER is the product of the covers of all the clauses up to that point. We use p, q, and r to avoid writing subscripts, and drop v.

Clause	Clause Cover	Product Cover
p q r	-00 +-0 ++-	-00 +-0 ++-
-p -q -r	+00 -+0 --+	+-0 ++- -+0 --+
q -r	0-0 0++	+-0 -0+
p -q	-00 ++0	-0+
-p r	+00 -0-	empty

Note that a merge occurred in the third line. There is no resolution, model elimination, linear, GC or linked conjunct refutation of the above set of clauses that does not use at least one of the clauses more than once for a resolution, extension, or conjunct node. If we consider the set of just the first 4 clauses, we have a satisfiable set. The product cover after the first four clauses have been used is not empty, and indicates that we need a clause subsumming -p r in order to get unsatisfiability. Finally, note that the first two clauses processed have only tautologous resolvents. While such are normally discarded in resolution, they pose no particular problems for covering.

III. The General Case

We propose to use the above scheme as the truth-functional unsatisfiability test in the basic replica-method for theorem proving. Such a method has the format:

1. Form a set of replicas of the input clauses.
2. Unify some of the literals or otherwise determine some substitutions for the constrained variables.
3. Test for truth-functional unsatisfiability.
4. If the test succeeds, stop; otherwise, add new replicas and/or substitutions and go to 3.

The last step is the crucial one. Covering expressions provide a convenient form for analyzing what new replicas to form, unlike other replica methods or even resolution in general.

We begin with a trivial example to illustrate the basic approach. Suppose S has 3 clauses, Pa, -Px Pf(x), and -Pf(f(a)), and suppose we start with one replica of each. Taking the literals in the order they appear below, the replicas and the covering expressions are:

Clause	Clause Cover	Product Cover
1. Pa	-000	-000
2. $-Pv_1Pf(v_1)$	0+00 0--0	-+00 ---0
3. -Pf(f(a))	000+	-+0+ ---+

Now the object is to produce the empty cover, i.e., to annihilate all the terms. The first term in the above cover can be annihilated by mating literals 1 and 2, i.e., Pa and Pv_1, by substituting a for the constrained variable v_1, making literals 1 and 2 identical. Then a term like -+0+ is self-conflicting because if the substitution had been made before the cover were calculated and only, say, literal 1 had been kept, such a term would never have been formed in calculating the product. After making the substitution we have a product cover -x-+ where the x indicates that literal 2 has now become identical to some other literal. No other substitutions are possible, so we now look for another replica. The leaf indicated by the remaining term is -Pa -Pf(a) Pf(f(a)). If we can find a replica that subsumes this, we will be done. Of course the appropriate replica is -Pf(a) Pf(f(a)).

Generally, it will not be the case that some subset of literals in an unsubsumed leaf will have a common instance with all of an input clause. For example, if the third input clause above had been -Pf(f(f(a))), the unsubsumed leaf would have been - Pa -Pf(a) Pf(f(f(a))). The best we could have done would have been to take replicas that had 1 literal in common with the unsubsumed leaf --either -Pf(a) Pf(f(a)) or -Pf(f(a)) Pf(f(f(a))). In either case, adding such a new replica (with a new literal) would again lead to a cover expression whose indicated leaf was fully subsumed.

The basic method, then, is to start with some replicas, calculate a covering expression, and then use that expression to guide the choice of new replicas or substitutions. We make some comments.

1. One can make wrong choices of replicas and substitutions; so there is definitely "search" in the general case; however, the search is confined solely to discovering the correct instances of the input clauses, and not to examining Boolean relations among these instances; in resolution-based methods, the forming of instances (by unification) is intimately bound to the examination of the Boolean relations.

2. When two replica literals are made equal, only one copy is saved; moreover, only one copy of a literal is needed regardless of how many clauses contain that literal; except for proof recovery, no record need be kept of which clauses contain a literal; literals are kept only to help choose new replicas.

3. Most importantly, it appears that one can apply more analyses to the choice from among a number of possible new replicas because the entire instance of the new clause is present, unlike binary resolution; this remark applies equally well to hyper- and UR-resolution, and represents, in the authors' view, a distinct potential advantage of these methods over binary resolution.

We are currently experimenting with an interactive FORTRAN program at Argonne National Lab based on the above method. We prefer interactive at this point in order to experiment and determine possible effective heuristics to use with this technique. Because the program is interactive it will not be possible to compare our results with those of more well-developed, fully automated programs. The basic command in the program is SGST, L, n where L is a literal number and n is an integer. This command forms common instances between subsets of the clauses indicated by the covering expression and input clauses in which n literals in the input clause are unmated. It then displays these unmated literals and, if the user desires, the instances of the input clauses themselves. (The name SGST stands for "suggest"; one way of interpreting the data that is displayed is as suggestions of the form "if you want to prove L, try to prove one of the following literals ..."). The user then chooses from among the instances. The program performs some tests automatically -- 1. it rejects tautologies, 2. if the semantic flag is on, it rejects instances whose unmated literals do not have false instances, 3. it performs demodulation. The user can choose from among the remaining suggestions; we currently use such information as 1. the number of supported literals used, 2. the structure of the terms, 3. the number of instances yielding the same suggestions, etc.

At the moment we are confining our experiments to Horn sets; this allows a heuristic which is analogous to the emphasis on units in resolution, namely, we add instances of only unit clauses. In this case, the covering expression serves basically as an efficient recording mechanism for constrained UR resolution [10].

The problem used in the example is Theorem H5 in [10], that "less-than-or equal" is transitive in Henkin models i.e., Q(A,B) and Q(B,C) imply, Q(A,C). The axioms and computer dialogue are listed in the Appendix.

IV On the Computational Complexity of Ground Covering

The time complexity of any new algorithm for refuting unsatisfiable ground sets is of interest because of the close relationship to the NP-complete problems. (See Cook [2] and Karp [9]). That is, P=NP if, and only if, there exists a sound and complete polynomial time algorithm for refuting unsatisfiable sets.

Cook and Reckhow [3] have investigated the question of the existence of a super proof system for the tautologies of the propositional calculus. A proof system π is super if there exists a polynomial p such that for every tautology x there is a proof of x in π no longer than $p(|x|)$, where $|x|$ is the number of symbols in x. Reckhow showed that NP is closed under complementation if, and only if, there exists a super proof system. The existence of a super proof system does not imply P=NP, since the mere existence of a short certificate of tautologyhood does not imply that the certificate can be found in a short (polynomial) amount of time. Of course, a proof that no super proof system exists would imply P≠NP, since P is closed under complementation.

We study the complexity of covering by trying to show that a given covering strategy is not a super proof system. If the strategy is sound and complete, then such a result implies that the strategy has non-polynomial time complexity, but the latter result would not imply the former. The length of a proof using a covering strategy will be the number of terms generated during the covering refutation of an unsatisfiable set.

1. Two non-polynomial strategies

It is not difficult to find a covering strategy that requires more than a polynomial number of steps. The following algorithm is that used for the ground case in the current implementation of covering (see Henschen and Evangelist [7]).

Algorithm 1. Simple scan.

Method. Read the input as given without attempting to sort the clauses.

Theorem 1. Simple scan covering is not a super proof system.

Proof. (Outline) The complexity of any covering strategy increases sharply when large numbers of pairwise-disjoint clauses are processed in sequence. This phenomenon occurs because the covering expressions associated with pairwise-disjoint clauses are always compatible. Each member T of the infinite family of unsatisfiable sets first given by Tseitin [13] contains large numbers of pairwise-disjoint clauses. Thus, we give these clauses in sequence to Algorithm 1 followed by the rest of the clauses in T, which is sufficient to force the simple scan method into non-polynomial behavior.

Corollary 1. Simple scan has non-polynomial time complexity.

Algorithm 1 is inefficient, because it is not well enough informed to avoid intersecting consecutive pairwise-disjoint clauses. The next algorithm seeks efficiency by avoiding pairwise-disjoint clauses whenever possible.

Algorithm 2. Model-directed covering.

Method. Pick an initial clause arbitrarily. Choose as the next clause one that eliminates as possible models the largest number of leaves in the associated semantic tree.

Theorem 2. Model-directed covering is not a super proof system.

Proof. (Outline). Again, we use Tseitin's formulas. The existence of exponentially long snakes (chordless cycles) in the n-cube, from which the formulas are derived, allows us to force Algorithm 2 into exponentially long proofs. (See Danzer and Klee [4] on the length of snakes in the n-cube).

Corollary 2. Model-directed covering has non-polynomial time complexity.

Model-directed covering was designed to discover quickly the unsatisfiable kernel of a set of clauses and will be most efficient on sets that are not minimally unsatisfiable. The sets of formulas described by Tseitin are minimally unsatisfiable and because of the long snakes in the graphs from which they are derived, contain long chains of formulas that are not pairwise-disjoint clauses but, at the same time, do not interact strongly enough to bound the lengths of proofs by a polynomial.

2. An incomplete strategy

In an attempt to understand the nature of covering better, we consider the following highly constrained version:

Algorithm 3. K-bounded covering.

Method. Choose the next clause using any desired strategy. The only constraint is that no more than k terms may be generated at each step, for some constant k.

Note that k is a bound on the number of terms generated by the intersection process and not on the number of terms required to represent the clauses. For example, the pyramids $P_n = x_n$ & $(x_{n-1} \vee -x_n)$ & $(x_{n-2} \vee -x_{n-1} \vee -x_n)$ & ...& $(x_1 \vee -x_2 \vee \ldots \vee -x_n)$ & $(-x_1 \vee -x_2 \vee \ldots \vee -x_n)$ are refutable by 1-bounded covering but require $O(n^2)$ terms to represent the clauses.

It is clear that if k-bounded covering were complete for some constant k, then Algorithm 3 would be a super proof system. Unfortunately, this is not the case. Let L_n denote all the 2^n clauses over n variables.

Lemma 1. The sets L_n require at least n-1 covering terms to be generated at some point in their refutation.

Proof. (Outline) Use the proof technique of Paterson and Hewitt [11] that demonstrated that log N pebbles are required to close off a tree of N nodes.

Cook [2] has given a technique for replacing any CNF formula F with a 3CNF (no more than 3 literals per clause) formula F' such that F is unsatisfiable if, and only if, F' is unsatisfiable. The idea is to use distinct dummy variables to represent excess literals in each clause. Further, F' can be constructed in a number of steps no greater than some polynomial in the length of F. Let L'_n be the translation of L_n into 3CNF using Cook's algorithm.

Lemma 2. The sets L'_n require at least n-1 covering terms, also.

Proof (Outline) Covering the leaf clauses in the semantic tree associated with L_n is equivalent to covering the corresponding subtrees in the much larger tree associated with L'_n. Thus, at least as many terms will be required.

The preceding lemmas give us the desired result.

Theorem 3. For each constant k, there exists an infinite family of unsatisfiable sets of 3CNF clauses not refutable by k-bounded covering.

Corollary 3. K-bounded covering is not a super proof system.

Let AUGUNIT be the family of sets of clauses refutable by unit resolution (see Chang and Lee [1]) augmented to allow all resolutions that produce unit clauses as well as those that use unit clauses. Let 1COV be those sets refutable by 1-bounded covering.

Theorem 4. AUGUNIT=1COV.

Proof (Outline) The case AUGUNIT $\subset$ 1COV is obtained by a straightforward simulation of augmented unit resolution by 1-bounded covering. For the case 1COV $\subset$ AUGUNIT, we use induction on the number of clauses. The central idea is to show that the DNF clause implied by the single covering term after processing j clauses is provable by augmented unit resolution operating on the same j clauses.

Theorem 5. 1COV is polynomially decidable.

Proof (Outline) By the fact that AUGUNIT is polynomially decidable. (See Jones and Laaser [8] for the complexity of UNIT).

The time complexity of k-bounded covering is still under investigation. It is possible that the set KCOV is NP-complete (clearly, KCOV is in NP). We conjecture, however, that it is not. In fact, we believe that it is related to the bounded resolution of Galil [5] as follows:

Conjecture p(k)-bounded resolution can simulate k-bounded covering for some polynomial p.

One goal for future research is to show that unrestricted covering is not a super proof system. We are currently attempting to reach this goal in the following way:

Conjecture Unrestricted covering is not a super proof system.

Proposed method of proof. A more careful analysis of the number of covering terms required for refuting the set L'_n for sufficiently large n may show that exponentially long proofs are required.

We also hope to study such topics as 1) representation of covering as a language recognition problem for finite-state automata (this representation appears to avoid the complexity inherent in processing consecutive pairwise-disjoint clauses using the covering expression representation), and 2) using dummy variables to represent subexpressions, like those of extended resolution (see Tseitin [13]).

References

1. Chang, C. L., and Lee, R.C.T., Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York, 1973.
2. Cook, S.A., "The Complexity of Theorem Proving Procedures", Proc. Third ACM Symposium on Theory of Computing, 1971, 151-8.
3. Cook, S.A. and R. Reckhow, "On the Lengths of Proofs in the Propositional Calculus", Proc. Sixth ACM Symposium on Theory of Computing, 1974, 135-48.
4. Danzer, L., and V. Klee, "Lengths of Snakes in Boxes", J. Comb. Theory, 2, 258-65, 1967.
5. Galil, Z., "On Resolution with Clauses of Bounded Size", SIAM J. Comp., Vol. 6, 1977.
6. Gilmore, P.C., "A Proof Method for Quantification Theory: Its Justification and Realization," IBM Journal of Research and Development 4, 1960.
7. Henschen, L., and W. M. Evangelist, "Theorem Proving by Covering Expressions", Proc. International Joint Conf. on Art. Intell., 1977.
8. Jones, N.D., and W.T. Laaser, "Complete Problems for Deterministic Polynomial Time", Theory Comp. Sci., Vol. 3, No. 1, 105-17.
9. Karp, R. M., "Reducibility Among Combinatorial Problems," Complexity of Computer Computations, R. E. Miller and J. W. Thatcher (Eds.), Plenum Press, New York, 1973, 85-103.
10. McCharen, J.D., Overbeek, R.O., and Wos, L., "Problems and Experiments for and with Automated Theorem-Proving Programs," IEEE Transactions on Computers C-25, 1976, 773-782.
11. Paterson, M.S., and C.E. Hewitt, "Comparative Schematology" Rec. of Proj. MAC Conf. on Concurrent Systems and Parallel Computation Dec., 1970, 119-28.
12. Prawitz, D., "Advances and Problems in Mechanical Proof Procedures", Machine Intelligence 4, Meltzer and Michie, eds., Edinburgh University Press, 1968, 59-71.
13. Tseitin, G.S. "On the Complexity of Derivations in the Propositional Calculus", Structures in Constructive Mathematics and Mathematical Logic, Part II, A. O. Silenko, (Ed.) 1968, 115-25.

Appendix

The problem in this example is the theorem that "less-than-or-equal" is transitive in Henkin Models. We give below commentary about the axioms as well as the input for the program and the interactive dialogue for the actual computer run. Qx,y means $x \leq y$, Px,y,z means $x/y = z$, and R stands for equality. E is the smallest element of the structure, D the largest. In this run we used a model of the axioms and hypotheses containing 5 elements, $E<A<B<C<D$. With this option, SGST rejects suggestions that do not have false instances and reports the number of these rejections and the number of tautologous suggestions. The proof of this theorem requires the use of one equality axiom. When equality axioms are present and participate in a normal resolution search, they tend to produce large numbers of resolvents. The model in this run eliminated practically all of these except the one needed for the proof. We also used demodulation of terms in suggestions. We used an option of SGST in our program that lists only the unmatched literals in the new suggestions without listing the entire clause(s) that produced the suggestion. For space considerations we abbreviate some of the program's clerical responses.

appendix continued on following page

The input to the program consists of the following:

```
INCL
-Q(V1V2) P(V1V2E)/                                  x ≤ y iff x/y = E
-P(V1V2E) Q(V1V2)/
-P(V1V2V3) Q(V3V1)/                                 x/y ≤ x
-P(V1V2V4) -P(V2V3V5) -P(V1V3V6) -P(V6V5V7) -P(V4V3V8) Q(V7V8)/     (x/z)/(y/z) ≤ (x/y)/z
Q(EV1)/                                             E is the smallest
-Q(V1V2) -Q(V2V1) R(V1V2)/                          x ≤ y and y ≤ x imply x = y
Q(V1D)/                                             D is the largest
P(V1V2F(V1V2))/                                     closure
-P(V1V2V3) -P(V1V2V4) R(V3V4)/                      well definedness
P(V1DE)/                                            Theorem 1. x/D = E
P(EV1E)/                                            Theorem 2. E/x = E
P(V1V1E)/                                           Theorem 3. x/x = E
P(V1EV1)/                                           Theorem 4. x/E = x
```

all the equality axioms for P, Q, R, and F

```
Q(AB)/                                              deny transitivity
Q(BC)/
-Q(AC)/
/
COPY, 5                                             COPY, N   causes a new constrained instance of clause N to
COPY, 7                                             be formed.
COPY, 8                                             In this example we start with instances of the
COPY, 10                                            units only. This causes no restriction because
COPY, 11                                            SGST intersects all the terms of the covering
COPY, 12                                            expression anyway.
COPY, 13
COPY, 14                                            Clause 14 is R(V1V1)
COPY, 24
COPY, 25                                            Clauses 24-26 are the three units of the denial.
COPY, 26
END
R(F(V1D)E)/                                         These are the demodulators corresponding
R(F(EV1)V1)/                                        to Theorems 1-4 above.
R(F(V1V1)E)/
R(F(V1E)V1)/
/
```

there follows here tables defining a model of the above axioms plus Q(AB) and Q(BC) which contains 5 elements -- $E < A < B < C < D$.

We now present the actual dialogue with commentary. User input is in lower case, computer response in upper case. Literals are numbered in the order in which they are entered into the search space via either the COPY command (see above input) or the TRY command (see below).

```
read input                                  read the input clauses, demodulators, and model.
sgst, 11, 1                                 11 is QAC from the denial -QAC

2 TAUTOLOGIES  14 SEMANTIC REJECTS

SUGGESTIONS                                 Program found 2 tautologies and 14 suggestions
1  -PACE                                    that failed the semantic test. These were mostly
OK                                          uses of equality axioms that are not relevant to
                                            the proof. Only one suggestion remains.
try 1                                       TRY adds the necessary new literals and updates the
NEW LITERAL IS 12                           covering expression.
sgst, 12, 1
3 TAUTOLOGIES    4 SEMANTIC REJECTS
SUGGESTIONS                                 Again the semantics rejected a lot of potential
1  -RF(AC)E                                 uses of equality axioms with only one suggestion,
                                            the right one.
OK
try 1
NEW LITERAL IS 13
sgst, 13, 1
0 TAUTOLOGIES    0 SEMANTIC REJECTS         No help from semantics here.
SUGGESTIONS                                 The reason is that now all of the suggestions
1  -QF(AC)E                                 are saying the same thing in different forms.
2  -PF(AC)EE
3  -PxDF(AC)
```

Transcript	Commentary
4 -PExF(AC) 5 -PxxF(AC) 6 -REF(AC) OK	
try all	Try all of the suggestions at once.
NEW LITERAL IS 14 NEW LITERAL IS 15 NEW LITERAL IS 16 NEW LITERAL IS 17 NEW LITERAL IS 18 NEW LITERAL IS 19 OK sgst, 14, 1	
TOO MANY SUGGESTIONS	The program stops looking after a certain amount of space is used up.
try, 15, 1	
NONE	15 yields nothing new
. . .	
try, 19, 1	None of the others yields anything new.
NONE	
sgst, 10, 1	At this point we go back and try another hypothesis, QBC.
2 TAUTOLOGIES 1 SEMANTIC REJECT	
SUGGESTIONS 1 PBCE OK	Note, semantic rejection applies only if unpaired literal is negative. Otherwise, as is the case here, a suggestion based on axioms and true hypotheses could never be found.
try 1 NEW LITERAL IS 20 sgst, 20, 1 3 TAUTOLOGIES 41 SEMANTIC REJECTS SUGGESTIONS 1 -PEF(CB)F(AC) 2 -PBCF(AC) 3 -PBDF(AC) 4 -PABE 5 -PEF(DC)F(AC) OK try all NEW LITERAL IS 21 NEW LITERAL IS 22 NEW LITERAL IS 23 NEW LITERAL IS 24 NEW LITERAL IS 25	
PROOF FOUND. LAST REQUIRED INSTANCE IS 1 -QAB PABE 9 24 end	At this point, all the ground literals required for a refutation are present, and only one more instance of an input clause is needed. The missing instance fully subsumes the covering expression and involves literals 9 and 24.

A Simultaniously Procedural and Declarative Data Structure and Its Use in Natural Language Generation

David D. McDonald
MIT Artificial Intelligence Laboratory
Cambridge, Massachusetts 02139, U.S.A.

Abstract

Goal-directed language generation is a decision process, drawing on knowledge about the phrases, words, etc. which could correctly express the speaker's intentions. Fluent speech must be planned. This entails being able to examine what one knows and to make inferences from it. This paper first sketches the generation process, showing where and how this knowledge is used. Then it presents the data structure that is used for encoding "how to say it" information. It is a schematic representation, which linguistic routines will interpret either as a procedure for constructing an English phrase or as descriptive data, depending on what is needed at the time.

Introduction

Language generation is not a simple process. It is not merely a matter of the speaker taking a representation of what he wants to say, looking up each of its tokens in a "mentalese to English" dictionary and then writing down the phrases in order. Natural language are a special purpose representation with detailed and complexly constrained grammars. They involve modes of representation that are (presumably) not used in ordinary thinking: serial presentation, the indication of relationships by morphological markings, etc. The "natural" translations of the tokens in a speaker's message are often mutually ungrammatical, making language generation a problem in planning.

The complexity goes beyond the problems of translation. What one actually tells an audience is different in quantity and quality from what one thinks to ones self. Not being telepaths, an audience has to be given adequate descriptions of whatever the speaker wants to refer to; they should hear only "relevant" information. Their reactions and inferences must be taken into account; arguments must be presented carefully. General maxims and discourse conventions exist and the speaker must follow them if he is to be understood.

Language generation - defined here as the process that takes place between the point when the speaker has decided what to say and the completion of a suitable English text - is best characterized as a decision making process. The speaker uses his special purpose knowledge to inform a series of decisions about which words to use, which constructions, what intonation, etc. To model the generation process in humans, or to build a computer program to be a "mouthpiece" for some computer program "speaker", requires characterizing the kinds of knowledge involved and specifying how it should be represented and where and how in the generation process it is used. What decisions must be made? What do they depend on? When are they made?

Over the last several years, I have developed a model of language generation. It is accompanied by an operational MacLISP program which provides its formalization and the beginnings of a practical utility. Operating alone, the program takes proofs or individual formulas in the predicate calculus as its input "messages" and renders them in English. It is presently being adapted as the mouthpiece of two advisor programs [Genesereth 1978][Goldstein [illegible]]. The model and program cover a good deal of territory, only a small part of which I will be able to discuss in this paper. The design of the generation grammar and the main process is given in [McDonald 1978b]. Implications of the model as a psycholinguistic theory are discussed in [McDonald 1978a]. A thorough specification of the model and a detailing of the implemented grammar will appear shortly in [McDonald in preparation].

A speaker knows how to describe the objects, relations, states, etc. that he will want to refer to. This knowledge is much more than just an association between tokens in his mental representation and English words. It is context sensitive - descriptions vary with who the audience is and

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intellience 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.

what has been said before. It interacts in a sophisticated way with the grammatical processing - choices are automatically added or omitted according to grammatical strictures, and output is readily customized to meet its syntactic context. And, as we will see, this knowledge is not only used but examined and reasoned about.

This paper is about the structure of this knowledge in my program. How does it appear? What does it do? How is it interleaved with the rest of the processing? The first half of the paper will sketch the model as a whole and motivate the position of this "lexicon" within it. The rest of the paper will present the representation for "entries" in the lexicon and illustrate how they are used.

Generation as Decision-making

There are an unimaginably large number of things we could talk about and a larger set of ways for expressing them in language. When we speak (or a computer program writes) we are, in effect, making a selection from that space of choices. More precisely, we are making a series of choices which, combined, pick out one text or utterance. This is a useful way to conceputalize the process because it suggests we look for specific "decision-makers", each possibly using different kinds of evidence and operating at different times according to how their decision effects the full text.

In studying generation, researchers have always divided the process into two parts: first deciding "what to say", then deciding "how to say that message" The first we can label "cognitive reasoning", the second "linguistic reasoning" The two processes are taken to rely on different kinds of knowledge, to use different predicates to interogate it, and generally to employ a different style of reasoning. For example, we might worry about telling our mother about our new "roommate" and try to anticipate how she would react. That is cognitive reasoning. Once we decide to do it, we worry if we can continually describe them in the passive and thereby avoid using that critical pronoun. That is linguistic reasoning.

My generation program only does linguistic reasoning. Its starting point is a "message" provided by some other program - the "speaker", for which it then finds an appropriate, translating text. The message is a representation of what the speaker wants to say, expressed in whatever formalism the speaker prefers. As the program will be doing all the linguistic reasoning and it does not operate until after the decision on what to say, this design implies that the construction of a message does not depend on any linguistic data (e.g. on how a word is pronounced or whether a pronoun will be used). To a first approximation, this appears to be true. Poets and diplomats may turn out to be exceptions, but at the present stage of research, a bicammeral process clarifies the problem and simplifies research.

The program puts no explicit requirements on the form of a message. There is no interlingua. Instead, for each new speaker program, a new "lexicon" is written. A lexicon is a translating dictionary. It records how the linguistics program is to interpret the tokens and relational structures that might appear in a message. It is where the overall system's "how to say it" knowledge is stored, since this information is exactly what the program needs in order to "read" an input message as a potential naturål language text.

In the systems I am developing, messages are given as set of statements. For example:

```
(is-true? A)
(assumed-true A)
A = (supports B6 B3)
focus = B3
```

The program has to know how these statements are related linguistically (they may well have been generated independantly by different parts of the speaker program). It then has to know how to decide what natural language form(s) to use to render them as one coherent utterance.

This information is provided entirely by the lexicon, in this case a lexicon for the Blocks World. The statements in this message, and generally so in all the systems under development, have the form:

(relation-i arg1 arg2 ...)

where any of the arguments may be relations themselves. Compounds statements - data base assertions - frames - etc. - are built up recursively from other statements, down to the level of the systems primitive objects.

The lexicon has an entry for each relation and primitive object that the speaker program might include in its "message language". Some of these "message elements" will denote goals - speech-acts to be performed like **is-true?** - which will be expressed as a polar question. Some will denote attitudes, e.g. **assumed-true**. These will effect how the main goal(s) is rendered. The use of "=" is a meta-syntactic device: used first to show that the argument to the two relations is the same object; and then to mark the binding of a "discourse-level" state variable. The **focus** fixs one of the degrees of freedom in the rendering of the "content" **support** assertion. **B6** and **B3** both denote blocks. The lexicon has one entry for "blocks" in general, which then refers to the "speaker internal" data structure for each particular block and uses its properties to pick a realization strategy. With the present lexicon, this message becomes: *"the green block is supported by the red*

one, isn't it?"

Constructing the Text

A generation grammar can be viewed as specifying a (tremendously large) space of possible utterances. One could encode the message to text generator as a grammar-based, augmented transition network (for example see [Wong 1975]), however that design is very inefficient, as it is a topdown search through the entire grammar. (For other difficulties with the use of ATN's in generation see [McDonald in preparation].)

The more direct technique is to let the message translate itself. This can be done with a syntax-directed design that reads through the statements and embedded relations of the message. Each element of the message, through its lexical entry, will contribute a phrase or modifier to the eventual final text. Just as the message is a composition of smaller message elements, the text that renders its intent into English will be a composition of smaller texts contributed by the elements' lexical entries.

Of course, while the interpreted message is the driving force of the procedure, the grammar must also be taken into account. Its role is to specify what constructions may be selected. It restricts the choices of the main, message-driven process to be grammatically consistant with the choices it has already made. It also performs the low-level, detailed actions which the language requires but which are automatic and not used to convey meaning, such as agreement and morphological adjustments to verbs and pronouns. In order to make this possible, a thorough syntactic, constituent structure description of every phrase is constructed as well as just the phrase's words, and words themselves are morphologically specialized only as they are being spoken.

* * *

Roughly speaking, each element of the message corresponds to one decision, i.e. what phrase is to render that element. The program is designed as a distributed process - therefore there is no *a priori* record in the lexicon or the grammar of when these decisions are to be made. The needed organization follows from the following two stipulations of the design.

(1) The process is to be incremental. No one decision-maker should have to specify more detail than necessary. If, for example, we have a relation with two arguments, like **(support B6 B3)**, then its entry will select, e.g., a clause with the verb *support*, but it will only insert the two arguments - untranslated - into slots in the constituent structure of that clause. It will not presume to refine them further. Left-right, sequence oriented linguistic phenomena, such as pronominalization, may well effect how the arguments to **support** are ultimately realized. Since the lexical entry for **support**, from its vantage point, can only appreciate hierarchical, conceptual relationships, it cannot make decisions that "hidden" grammatical phenomena may affect.

In the same vein, the clause which that entry selects is only minimally specified. The **support** relation may be being used for many different purposes in the message as a whole: modifier, topic, etc. Each of these would place it in a different grammatical context, where it would have to be morphologically specialized to fit its role. As the entry shouldn't need to anticipate these requirements - they are quite general - it selects an underspecified clause structure and the background grammatical process will later refine as required.

(2) All decisions are indelible. Once a word or phrase has been selected, or a word spoken (printed out) it can not be taken back. This has the immediate implication that the needed decisions must be made in a particular order, i.e. no decision can be made before all the other decisions which it may depend upon have been made. These dependancies fall into three categories: (a) those due to the role of the element in the message - what relation contains it; (b) dependancies due to where it will appear in the sequence of the final text; and (c) "global" references to it from other statements (e.g. **focus = B3** creates a dependancy on the **(support B6 B3)** relation, since it adds an additional linguistic role to those it must already play).

These classes of dependancies lead directly to the needed ordering of the decisions, or, to put in another way, they determine the order in which the entries of the elements of a message are to be consulted. To witt: first examine the statements of the message. Determine which are "content" elements, and which dictate another element's role (e.g. to define a question), or give "stage directions" (as in maintaining continuity of focus). These later statements will call for "global" realizations - side-effects on how the content elements are rendered. Cluster them with their affected content elements and then proceed to process the lexical entry of the conceptually dominant content element.

That entry makes its decision and returns a surface level linguistic structure which embeds the arguments of that element as constituents within it. The program now shifts from reading the message to reading this new linguistic structure. A simple interpreter will walk through its nodes and subconstituents, dispatching to grammatical routines indicated by the names of the nodes, names of their constituent slots

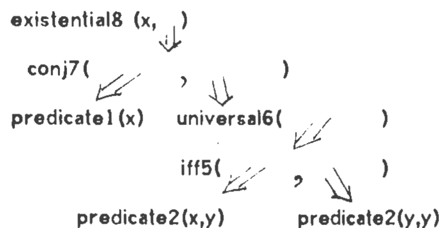
(see below) or grammatical features attached to them. This is the source of a "background" grammatical process.

Whenever a constituent is an embedded message element, its entry is processed and the linguistic structure that is returned replaces the element and is then recursively walked in turn. This surface structure representation, with the interpreter, is a totally adequate state description of the process - completed structure and spoken words behind the interpreter and planned linguistic structures and the remaining embedded message elements in front of it.

* * *

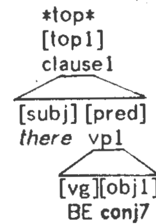
Let me move directly to an example. Consider this very simple message, a single formula in the predicate calculus. (This is the first line of a proof of Russel's "barber paradox" - the assumption which will be shown to lead to a contradiction.) **∃(x) barber(x) ∧ ∀(y)[shaves(x,y) ↔ ¬shaves(y,)]** This message is processed by a lexicon with an entry for each logical connective and special, schematic, entires for each predicate and default variable category (i.e. x and y denote people). They render that formula as: *"There is some barber who shaves everyone who doesn't shave himself."*

Now let me redisplay the formula as a tree of named subformulas. This is the way in which the program actually sees it, and it highlights its hierarchical nature.



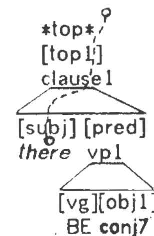
There is only one statement in this message, a content item, **existential8( )**. Therefore the process begins by consulting the entry for existentials (see below). I will go through this derivation quickly, emphasizing when choices are made and roughly what they are.

The entry for existentials must select a phrase - a linguistic context - in which its argument, **conj7**, can sucessfully be rendered. The "mathematical" phrasing, *"for some x, ..."* has this property, but it is a "less fluent" phrasing than say, *"There is an x which ..."*. The entry will look at **conj7**, or rather at the lexical entry for conjunctions, and "ask it" whether the more fluent phrasing is possible. The answer will be "yes" and this constituent structure is then returned and instaled as the process's state description. The interpreter for data-directed processing now starts at the top node - a formal node, *top*.



Names enclosed in brackets, [], are the names of cannonical locations in the consitituent structure. Names in italics are English words, embedded in the surface structure but not yet spoken - "spoken" text will appear in a line just below the tree. Syntactic nodes are given unique names based on their category. Names in bold-face are message elements. Such constituents are actually special data structures which represent those instances of those elements in the planned surface structure. This will be explained later.

The interpreter now moves down through [top1], the first constituent of the utterance. The name "top1" signals to the grammar (via a dispatch by the interpreter) that this is the start of a new sentence - the next word will now be capitalized. Moving into the clause causes the interpreter to rebind a series of grammatical status variables. (See [McDonald 1978b] for further discussion of how the grammatical processing operates.) Inside any node, the interpreter examines each of the nodes constituents in turn from left to right. Here, [subj] contains the word *there*. Whenever a word is reached, it is sent to a morphological subroutine for final contextual adjustments to its print name and then printed out on the console.



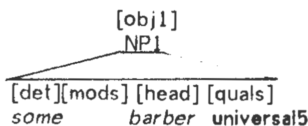
"There ..."

The next constituent, the clause's predicate, contains a node. The interpreter recurses and moves to that VP's constituents. The [vg] ("verb group" following Halliday) constituent initiates considerable amount of grammatical processing, as the information needed to fix the form of the verb's tense is located and centralized. This includes checking the number and person of whatever is the [subj]. The word *there* is actually an indirect pointer. When it was instaled as the subject, the instaling grammatical routine added a link to the [obj1] constituent. Therefore the question is "forwarded" to it. Since **conj7** is an embedded message element, the question is answered by consulting its lexical entry.

Now at [obj1], **conj7**'s entry is processed. Conjunctions, in issolation, could be rendered either as "real" conjunctions: "*X, Y, and Z*", or as specifying multiple properties describing one object - in this case the variable **x**. This specification could then be rendered either as a clause or a noun phrase. However, once a linguistic context has been established, the degrees of choice open to the conjunction entry (or to any entry) may be radically reduced. This instance of **conj7** is the [obj1] constituent of clause1. Here it must be rendered as a noun phrase. The entry for conjunctions will build a noun phrase by using its first conjunct to create the [determiner] and [head] and making subsequent conjuncts into modifiers. It does this schematically since it does not want to presume on how the entries for those conjuncts will actually render them. Since **Conj7** is the conjuction of **predicate1** and **universal6**. the [obj1] slot now looks like this, with the second conjunct attached as a property to this "instance" of **predicate1**.

```
[obj1]
 |---------| qualifier: universal5
predicate1
```

Predicate1 stands for **barber(x)**. The lexical entry for **barber** marks it as denoting a "category", something that may be rendered as a head noun or classifier. This causes the predication to be interpreted (by the entry for predicates) as focused on the variable as an object rather than on the predication as a relation. That entry, seeing that this instance is in a "nominal" context, initiates the construction of a noun phrase using the noun from the entry of **barber** as its head, and deriving the determiner from the quantifier used for the variable. (in a "clausal" context, it would have constructed a predicate-nominal) The noun phrase building process notices the affixed property and makes it a constituent.

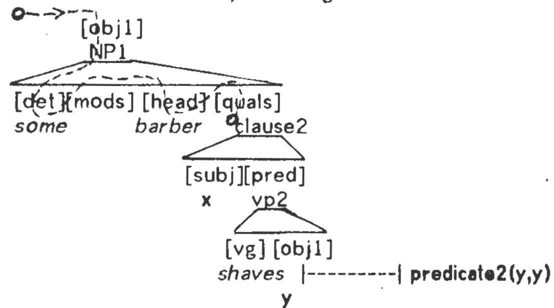


The interpreter moves through NP1's constitutents and has *some* and *barber* spoken. The null constituent, [mods] ("prenominal modifers") is ignored. The [quals] slot ("postnominal qualifiers") is associated with grammatical routines for forming relative clauses when required (a qualifier might alternatively be a preposition group or verb group).

Since relatives are (optionally) signaled by relative pronouns, these routines must know how, e.g., **universal6** is going to be realized before the interpreter actually reaches it. This can be read from its entry as we will see. Relative pronouns are grammatically "bound" to the object that the noun phrase derives from, in this case the variable **x**. The first instance of **x** in the phrase that realizes **universal6** will be supressed as part of the grammatical signature of a relative clause. The form of the relative pronoun depends on the grammatical relation of that first instance of **x** to the clause it is in - "subject" or "oblique" case. Again, we need to know something about **universal5** before we otherwise would get to it, and again, its lexical entry can be designed to supply the answer without actually building any linguistic structure.

Leaving aside for this paper how the relative clause forming routines operate, let us proceed. The relatives routine generates *who*, then the interpreter reaches **universal5**. Its entry has a choice between "literal" and "fluent" constructions, much like the entry for existentials. It must examine the potential linguistic renderings of its arguments to see it they will fit the requirements for the fluent form, else it takes the literal choice. As, in fact, the fluent choice can go through, (i.e. we can express the quantification through the determiner on the subject of a "subject-predicate" construction) I will move right on to **iff5** which directly replaces **universal6** in the [quals] slot.

The entry for biconditionals has a similar spread of choices, and its choice is dependant (1) on the potential of its arguments, and (2) on "rhetorical" context. It has a fluent construction based on interpreting the right-hand side of the biconditional as restriction on the variable it shares with the left-hand side. The right-hand side then becomes a restrictive relative. However, this reading omitts explicit mention of the "only if" aspect of the biconditional. An additional strategy, triggered by a rhetorical indicator can add that phrase. Another indicator will select a "reduced" reading, useful in making an argument: "*if X then Y, and if Y then X*". This instance is unmarked, and we get this result.



"*...some barber who*"

The grammar routines turn the instance of **x** in [subj] into a trace (in Chomsky's sense) when the interpreter reaches it. The rest of the surface structure is rendered in almost exactly the same way as the parts described so far.

One remark: the final instance of **y** will be pronominalized and then made into a reflexive by the morphology routine. Besides its surface structure plan, the

program maintains a time-organized record of what message elements have been mentioned so far, with an abbreviated discription of the grammatical circumstances. As part of its operation, the interpreter first checks each message element that it reaches to see if it has been spoken before. If it has not, the interpreter goes directly to its lexical entry; if it has, then it goes instead to a grammatical subroutine for pronominalization. This subroutine will construct a high-level description of the relations between this instance and its anaphor, plus a notation of other mentioned message elements with which it might be confused, and then makes a heuristic decision to use a pronoun or not. If it decides not to use a pronoun, the decription plus a notation of why a pronoun was inappropriate is made available to the element's entry.

The things you you need to do with a lexical entry

The primary "mission" of an entry is to record which linguistic forms could potentially be used to render elements of its type, plus the contextual conditions that select between them. These conditions can include: (1) internal details of the relation's arguments; (2) "global" restrictions introduced by other message statements; (3) the linguistic potential of some argument, e.g. whether it will be a noun phrase or a clause with a certain subject; (4) the discourse situation - has the element been mentioned before; (5) the grammatical context - what kind of constituent is this instance of the element.

The stipulation that the generation process be (1) incremental and (2) indelible has the effect of requiring the program to know things about the eventual linguistic form of a message element before it can legitimately be constructed. This information must be available from an element's lexical entry or else readily computable in the current context. This is a "secondary mission" which the entry representation must support. It includes:

(A) For organizational purposes, a short, highlevel description is needed of the element's eventual form. The syntactic category of the to be created node: NP, prepg, etc. will suffice. This is used in planning how to position elements with respect to each other and to the established surface structure.

(B) The need to choose among alternatives can call for applying linguistically oriented predicates to a relation's arguments, e.g. "possible-nominal", "predicate-about()", or more involved questions like "what will be eventual, default position of x in the unmarked rendering of **universal6**?".

(C) Ungrammatical possibilities should be automatically ruled out of an entry's list of possible realizations once the position of each instance of its element type within the planned surface structure is fixed.

(D) Part of an entry's realization strategy can include "forcing" the rendering of one or more of the arguments it embeds. It may preempt all or part of the decision process of the argument's entry so as to insure that an otherwise free choice goes the way that fits best with its intentions.

(E) Every relation type will have an "unmarked" rendering which is suited to expressing the content of that relation as it is usually intended. However, a speaker program can choose to use its data for other purposes, e.g. as a modifier, or in conjunction with a statement like **focus = B3**. This usage calls for applying a "transformation" to the entry's selection, but before actually constructing any linguistic structure since syntactic structure may not be changed one created. Rather, transformed instances are to be generated directly in their new form.

Realization strategies

A lexical entry, when used to construct a phrase, makes one or more linguistic decisions. These are decisions to create certain syntactic nodes, to fill certain of their consitituent slots with certain English words or argument elements, or to add grammatical properties to the existing plan. All such actions are a part of the program's grammar. They are formalized as "realization strategies" Operationally, a realization strategy is an object much like a function. It is applied to a set of arguments, whereupon it constructs a new constituent structure of the specified type embedding those arguments or modifies an existing structure supplied as one of its arguments.

An entry will record that it may select such and such a strategy if certain tests are met. That record may then be used to symbolically determine what the entry constructs by examining the stored schema that defines the strategy. This schema always has at least two fields, a "phrase" and a "map" (It may other fields annotating its grammatical features, what sort of thing it is useful for, etc..) For example:

```
CLAUSE-OBJ1
  phrase (basic-clause ()
                predicate (vp-obj1 () ))
  map ((first . mvb)
        (second . subj)
        (third . (pred obj1)) )
```

This indicates that the realization strategy named "clause-obj1", when applied to three arguments, will construct and return two syntactic nodes: the first a clause using the consitutent schema named "basic-clause",i.e. {[subj] [pred]}; and the second a verb phrase using "vp-obj1" - {[vg] [obj1]}. The verb phrase fills the [pred] constituent slot of the clause. To find out what would be the disposition of the strategy's arguments, its "map" field is interpreted. The map associates arguments (named by their order) with constituent slots in the phrase.

When reasoning about entries and their decisions, realization strategies are accessed in their schematic form by specially written procedures. When they are being used in their primary function - to actually build linguistic structure, either a general interpreter is used (particularly when debugging) or a "strategy compiler" is run as the program is loaded and efficient LISP routines are used instead.

the Design of an Entry

In the first implementation of this model, strategies and entries both were encoded as LISP functions. The flexibility of a procedure makes it a good implementation choice in just such an early point in the research when one is still unsure of what information will have to be encoded and of how it will be used. But, writing a procedure is a craft. When encoded procedurally, the information in the lexicon must be copied again for each new linguistic predicate. This work is redundant, errors are easily made, and the technique for creating new entries is harder to describe to potential users of the system.

In general, the answer to these problems is to develop a declarative, schematic representation. (1) The common procedural "glue" is extracted and consigned instead to a set of interpreters and compilers specialized for each different use of the material. (2) The different kinds of information within one of these objects can be named and distinguished within the schemata, making them easily accessible to routines that would want to reason about them.

The specification for a lexical entry that is given just below is the specification of what must be entered by the human designer. That structure is expanded when the program is actually loaded to be the more fleshed out structure that we will see shortly. It is important as a matter of human engineering to cut down the designer's "make work" load as much as possible by letting all the predictable construction be done automatically. Names in angle-brackets are non-terminals to be defined or are self-explanitory. Square-brackets enclose optional material. The star (*) is Kleene star.

```
<lexical-entry> ::=
      (def-entry <name> <list of argument names>
          <body> )
<body> ::= <decision>*
<decision> ::= (   [<name>]
                   [<skeleton>]
                   [local variable bindings]
                   <filter>* )
<filter> ::= ( <any contextual predicate>* <choice> )
<choice> ::= <name of a sub-entry>
             | <an editing action on the list of strategies>
             | <strategy-application>
<strategy-application> ::=
          ( <strategy name> <list of arguments> )
<skeleton> ::= ( skeleton <strategy-application> )
```

The overall form of an entry is a call to the read-time construction function, **def-entry**, which does all the cross-indexing needed to compile the entry's ultimate form. An entry's decisions are named according to the particular linguistic form involved. Noun phrases, for example, can involve independant decisions for their head noun, determiner, and modifier constituents. Clauses can involve one decision to fix verb phrase type and a set of other decisions to add adverbial or adjunctive modifiers. The named decision is the handle that the entries for "globally realized" message elements use to add to or preempt the decisions of a content element's entry.

Conceptually, the decision making process involves evaluating a decision tree of predicates against aspects of the current context, with the strategies as its leaves. The decision tree format however, while it is a good design for a procedure, needs to be unbundled if it is to be reasoned about. In this declarative design, making a decision is formalized as a "filtering" operation on list of possible strategies. To make a decision, an interpreter will evaluate all its filters and then elect the first strategy remaining in the list.

Each filter consists of a conjunction of predicates. The generation model makes no *a priori* restrictions on the kinds of predicates allowed. It does however make some kinds of information more accessible than others (cf. [McDonald 1978a] for more discussion). If all the predicates evaluate as true, then the filter's "choice" is taken. It may be the name of a "sub-entry", i.e. another block of filters which this one was the gate to. In the predicate calculus lexicon for example, the decision of how to interpret quantification as a determiner is shared between the entry for predicates and the one for issolated variables via a "sub-entry". Otherwise the choice could be either the application of a specific strategy by name, or else an editing instruction which reorders or deletes names from the list of possible realization strategies.

Below is the entry of the Blocks World lexicon for the relation **support**.

```
(def-entry support (under over)
 skeleton (clause-obj1 support under over))
```

This is a simple case, and it can be written simply through defaults implemented by **def-entry**. It makes no decision, that is, every instance of **support** rendered by this entry will go into the same linguistic form. The two arguments are named "under" and "over" for the convenience of the human designer - unfortunately the interpreter can not appreciate them except for their order.

Below, is the structure that "def-entry" builds (omitting a compiling operation). The default decision is named "clause-matrix" because the indicated skeleton realization strategy builds a clause. Unbeknownst to the **support** statement, other statements in a message might be intended as modifiers to it. They would be grouped as "clause-modifier" decisions, implicit in the construction of any clause

Notice that while the designer may not have had any more complex ideas of how the relation could be rendered, the grammar automatically adds possibilites just because the selected strategy, "clause-obj1", builds a clause which is a member of a particular transformational family. (This use of "transformation" is closest to Harris's usage.)

```
SUPPORT
   type lexical-entry
   realization class local     ;versus "global"
   system-status interpreted  ;vs. "compiled"
   arguments (under over)
   will-be clause
   skeleton (clause-obj1 support under over)
   decision structure descriptor  clause-matrix
   matrix
     (will-be clause
     strategy-set ( (clause-obj1 support under over)
                          (passive-by_phrase) )
     filter-set
          [details nil
               global  ( focus_on_object->passive )
               potential-realization nil
               discourse nil
               context nil ]
     ))
```

The filter set is organized by the domain of the predicates for the convience of later reasoning routines. The designer did not specify any filters, only a default. **Def-entry** added **focus_on_object->passive** and the corresponding **passive-by_phrase** strategy on its own by acting on a general grammatical principle that otherwise unmarked clause structure should be sensitive to discourse focus. Only the name of the filter has to be entered since it is common to many entries.

The filter is a LISP predicate. Its actions are sufficiently idiosyncratic that, given that it is otherwise annotated, it is expedient to leave it encoded as a procedure.

```
FOCUS_ON_OBJECT->PASSIVE
  type filter
  filter-type global
  predicate
    (cond (focus ;is there a current focus?
             (let ((obj (obj1-of skeleton)))
                       ;defines and initializes the variable obj
                       ;obj-of reads a strategy's map
                  (cond ((eq obj focus) (return t))
                        (t (return nil))) )))
  choice
      (select-strategy 'passive-by_phrase)
```

Evaluating linguistic predicates for planning purposes

The precise way in which a formula of a predicate logic is encoded often is the result of an unspoken convention for translating from English to logic. The conjunction of **barber(x)** and **universal6** above, for example, is just a conventional way to encode the type of the variable and does not play a real role in the formal manipulations of the proof. A generation lexicon in this domain often should "second-guess" the encoders of the logic and try to recover the fluid linguistic forms that they were very likely working from.

A case in point is the entry for universally quantified formulas:

```
(def-entry universal (variable formula)
   ( (variable->simplex-NP variable subformula)
     (embed-quantifier variable))
   (*default* (for-x-proposition subformula)) )
```

(Here again there is only one decision to be made and the outter parens are omitted.) The fluid choice, checked for by the linguistic predicate **variable->simplex-NP**, expresses the quantification via the choice of the determiner of the right toplevel NP within the clause. For example, **∀(x) man(x) → mortal(x)** meets this property and can be rendered: *"all men are mortal"*. Other formulas, such as **∀(x) lover(x) → loves(world, x)**, can not be put in that form without extensive transformations and changes in subordination - something that this entry would not want to be responsible for. These will be rendered with the "safe", mathematical form.

In English, what the predicate **variable->simplex-NP** does is to go through the set of possible strategies of the **clause-matrix** decision of the subformula. Its conditions are satisfied if the map of at least one of the strategies indicates that the variable will ultimately be a direct constituent of the toplevel clause, i.e. [subj], [obj1], or [obj2]. It also notes which is the relevant instance of the quantified variable inside the subformula, and, via a standard device in the entry interpreter, rebinds **variable** to that value.

Instances: recording and preempting decisions

Once the decision is made to "embed the quantifier", how is it to be implemented? Since its target, the first **y** in **iff5 - shaves(x,y) ↔ ¬shaves(y,y)**, is not the next element in the surface structure interpreter's "path", there must be some means of recording this decision until the proper time. This is done through the use of special data structures representing "instances" of message elements in the surface structure plan.

An instance is a scratch pad, a working data structure that exists only for the life of the plan. Instances are needed simply because whenever there is more than one reference in a plan to an element from the speaker's domain, say **x** or **shaves**, they may need to be treated differently. Properties, for example, are attached to distinct instances of a message element, not to the element itself. When the entry for a certain type of message element is processed, its predicates refer to the situation of particular instance of that element at a particular site in the constituent structure.

The instance structure for each element in the message is created at the beginning of the process. It is initially very sparse, and is augmented each time a decision is made that effects its message element, e.g. when it is positioned in the surface structure, or whenever a predicate in one of its filters is evaluated by a linguistic predicate "probing" its possible realizations. Strategy induced side-effects, like the decision to "embed the quantifier", effect instances by specially marking them.

This is the instance structure for the "y" of **shaves(x,y)** just after **embed-quantifier(variable)** is run. (Recall that **variable** was bound to just that instance when the predicate **variable->simplex-NP** "found" it.) The strategy acts by "preempting" the **NP-determiner** decision of this instance.

```
INSTANCE-37                ;a gensym
   type  instance-of-a-msg-elmt
   self  y
   global-entanglements  nil
      ;this is a collection site at which to position globally
      ;realized message elements until the content element they
      ;effect is reached.
   location-in-surface-structure  nil

   NP-head (status  untouched
            filters-evaluated  nil
            new-filters  nil
            current-phrase  basic-NP   ;the default
            current-map   ((person . head))  ;ditto
           )
   NP-determiner
      (status  preempted
       filters-evaluated  nil
       new-filters  nil
       current-phrase  nil    ;inherited from NP-head
       current-map  ((every . det))
       )
```

Embed-quantifier has (1) determinered which quantifier word to use, and (2) marked that instance of **y** so that the normal deliberation of the **"issolated-variables"** entry is preempted and its choice used instead.

Transformations and grammatical context

This same technique of writing "overriding" specifications into an instance structure is used to implement transformations and the "pruning" of realization strategies to fit the grammatical context.

A transformation is an operation on an instance's **current-map**. A globally realized statement like **focus = B3** triggers a filter which is not checked until after the basic shape of the clause has been decided upon, i.e. after the **current-phrase** and **current-map** have values. The action of, e.g., **passive-by_phrase** is to transform **current-phrase** to "clause-byobj" and switch the map: [subj] => [byobj], [obj1] => [subj]. Since this operation uses only a linguistic vocabulary, it can be used generally.

Grammatical procedures are run by the interpreter according to the details of the constituent structure plan as it walks through it. As part of their action, they will augment relevant instances with special purpose filters which have the effect of removing from consideration any strategies that would be ungrammatical in that context.

For example, the logic lexicon will render conjunctions of predicates over the same variables as either clauses or noun phrases. **Red(B6) ∧ supports(B6,B3)** can become either *"the red block supports the green one"*, or *"the red block supporting the green one"*. But in a "clausal" context the noun phrase rendering must be blocked (for example when its used to describe a state: "*when ...*"). The clause rendering can go through in nominal contexts because various nominalization transformations are available and would be applied by the background grammar process.

The block can be effected by the addition of this filter to the **context** field of the **new-filters** of the instance.

```
BLOCK-NP
   type  filter
   filter-type  context
   predicate  "true"
   choice   (remove  '(np))
      ;i.e. remove from the list of possible strategies any
      ;that lead to constituent structures dominated by a node
      ;with that feature. Any filters that only lead to such
      ;strategies are pruned with the same action.
```

* * *

This paper has tried to sketch the design of part of an extensive research project. The generation lexicon must coexist with a grammar, various process interpreters, and message-building conventions, none of which can be discussed in this short a space. The technical report in preparation should give all the facets of the design a through presentation.

The "meta-level" principle that motivates the design of this lexicon and of the model generally is a reaction to the extreme interaction that exists among the knowledge elements involved in natural language generation. I take it to be impossible, from an engineering standpoint, to design and implement unitary, fixed procedures to perform the generation process. Instead, the input and momentary state description must be used to determine what action to take dynamically. This requirement lends itself to the use of schematic representations and interpreters which react to the description languages developed to organize the grammar, the lexicon, and the intermediate states of the process.

References

Genesereth, Michael (1977) "An Automated Consultant for MACSYMA" MACSYMA memo 5, MIT Lab for Computer Science.

Goldstein, Ira, (1978 forthcoming) "The Genetic Epistimology of Rule Systems" memo 449, MIT Artificial Intelligence Lab.

McDonald, David D. (1978a) "A Model of Language Generation", memo, MIT Artificial Intelligence Lab.

_____ (1978b) "Language Generation: Automatic Control of Grammatical Detail", paper submitted to COLING-78, August 1978, Bergen Norway.

_____ , (1978 in preparation) "Linguistic Reasoning During Language Generation", Technical Report 404, MIT Artificial Intelligence Lab.

Wong, Harry (1975) "Generating English Sentences from Semantic Structures" Technical report 84, Dept. of Computer Science, University of Toronto.

KNOWLEDGE IDENTIFICATION AND METAPHOR

Roger Browse

Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5

Abstract

This paper deals with the organization of semantic properties of nouns and their use in the determination of case role acceptability. By only examining simple hierarchical inclusion properties, natural language systems have been unable to deal with metaphoric language use. The research outlined here is an investigation into the use of general methods for the determination of case fulfillment. These methods include a more flexible use of the hierarchical relationships represented in the system, and the methods also use the contents of other case-frames. These methods develop the degree of freedom necessary for the acceptance of metaphoric language. Other interesting properties of the system are described which also argue in favour of this approach to the organization of semantic knowledge.

Motivation

A major challenge in the development of a semantic component of natural language analysis is to represent knowledge in such a way that it will be useful in determining the meaning of input sentences. In addition to their use in describing sentence meaning, case grammar constructions have been used successfully to organize semantic knowledge. In particular, the use of cases provides insight into constraints that are important in establishing the roles played by noun groups. For example, the positioning of noun groups and the existence of certain prepositions can serve as constraints. However, the semantic properties of the noun group itself are of major importance in determining its acceptability as a case filler. The semantic requirements for case filling (often called selectional restriction rules) are usually specified as positions within a hierarchical breakdown of nouns. These requirements are often quite strictly applied. The research described in this paper is an investigation of more flexible methods of determining case role acceptability. The remaindure of this section demonstrates the restrictions which have been imposed upon natural language analysis through the strict use of hierarchical properties.

One step in developing a case system for a verb is to examine its range of use. As an example, consider a few of the different nouns which can occupy the object position for the verb "to play" (restricted to the musical sense).

Dick plays Beethoven.
Dick plays the piano.
Dick plays The Moonlight Sonata.

Each of the above sentences could actually be describing the same event, with slightly differing emphasis. There are essentially two ways of introducing simple hierarchical properties as the case role fulfillment requirements for this object position[1].

1) Develop several different sub-meanings. For the above example:

    PLAY1 (object must be COMPOSER)
    PLAY2 (object must be MUSICAL-INSTRUMENT)
    PLAY3 (object must be MUSICAL-PIECE)

   The advantage to this approach is that semantic properties and case requirements can be represented and processed in a uniform way. The disadvantage is that the verb's meaning has been fragmented: PLAY1 is no more closely related to PLAY2 than it is to JUGGLE1.

2) The other, more frequent, approach is to develop a specialized case system for the verb,

[1]A third possibility of introducing all-encompasing semantic properties such as "playable" has been ruled out for obvious reasons.

which indicates all the specific roles that are played by each possibily appearing noun type. Thus for the verb "to play", there might be, in effect, a case "composer". The advantage of this approach is that the meaning is centralized but the disadvantage is that the processes involved may be specific to each verb.

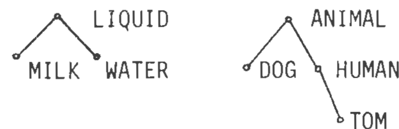
These two approaches differ in their shifting of emphasis from centralization of meaning to uniformity of processing, but they are also similar in an important way. The methods both involve a very direct pre-programming of permissible sentence meanings. It is important to develop a means of representing sets of acceptable meanings, particularly for clearly defined natural language interface programs. But sharp distinctions among semantic requirements do not exist in natural language use. Many researchers (Collins and Quillian 1972, Bobrow 1975, Wilks 1977) have pointed out that the breaking of semantic requirements (preferences) is the rule, not the exception. Any system which is to successfully understand natural language must take into account the freedom with which metaphoric language use enters into ordinary speech.

Prototypes

The research outlined here considers the possibility that the organization of knowledge, rather than its content, is primarily responsible for determining the breadth of meanings which the system can accept. Thus, the form of knowledge representation has been simplified to permit a more direct examination of organizational aspects.

The knowledge has two parts, prototypes and hierarchies. Prototypes are simple subject-verb-object triples which represent selectional restrictions upon the subject and object positions for the given verb. For example (HUMANS DRINK LIQUIDS) is a prototype which indicates that DRINK expects (or may accept) a HUMAN subject and a LIQUID object. They are called prototypes because they are intended to represent language-use experiences to which the comprehension process appeals in order to understand a new input.

Accompanying the prototypes is a hierarchy of semantic properties from which the selectional restrictions may be satisfied. For example:



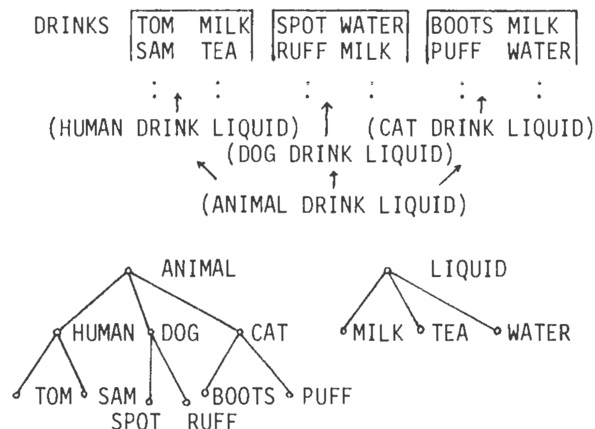
The goal of the processes which use this type of knowledge is to identify those pieces of knowledge which pertain to an input form which is also given as a subject-verb-object triple. This goal, and the form of the knowledge seems reasonable yet arbitrary. so, befofe continuing to show the ways in which prototypes are processed, the relation between prototypes and real-world knwoledge will be examined. As a result, the connection between the hierarchies and the prototypes will be more firmly established, and the goal of the processor will be justified. In addition, an organization of the prototypes will be introduced.

Consider the following simple representation of real-world knowledge about the relation "drinks":

DRINKS

TOM	MILK
SAM	TEA
SPOT	WATER
RUFF	MILK
BOOTS	MILK
PUFF	WATER
⋮	⋮

The use of such knowledge could be enhanced with a specification of the extent of the domains. A template such as (ANIMALS DRINK LIQUIDS) could be used as a preliminary test upon the use of the relation. Or, even better still, consider the following partitioning of the relation:



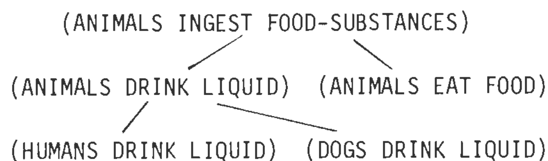
This shows a hierarchy of semantic categories to which the domain individuals have been assigned, and templates which represent the properties over which the partition has been made. These templates are, in fact, the prototypes described earlier. The hierarchical categories can be thought of as types, to which assignments are made, and then prototypes contain knowledge about the application of these types to the relation, the result of which is an indexing scheme

The generalizations about semantic categories that are contained in the prototypes are quite distinct from intensional knowledge (or generalizations about real-world knowledge). For example, the intension:

(X)(HUMAN(X) (EY)(LIQUID(Y) DRINKS(X,Y))

is still real-world knowledge, and not knowledge about the semantic categories involved in the partitioning of the relation.

It is useful to view prototypes in this way because it points up their organization into layers of generalization. For example:



This organization of prototypes serves as a means of allowing a match between an input and a prototype at any level of generalization. In addition, it provides a quick way to filter out possible prototypes.

For simple retrieval of extensional facts, the prototypes could be used as a straightforward indexing scheme. For more complex question-answering, the match of a prototype to an input could contribute towards the development of a form to be evaluated for an operation upon a data-base, as in Woods' procedural semantics (1968). It is a reasonable assumption that, regardless of the application, it would be useful to identify those prototypes which match an input. This idea is extended to the notion of <u>knowledge identification</u>: for a given input, a number of prototypes may have some relevance, one of which is distinguished as a <u>target prototype</u> of the matching process. For example:

```
input : DICK PLAYS BEETHOVEN
target: HUMANS PLAY MUSICAL-INSTRUMENTS
others: MUSICAL-INSTRUMENTS EMIT MUSIC
        COMPOSERS WRITE MUSIC
        BEETHOVEN IS COMPOSER
        DICK IS HUMAN[1]
```

<u>Knowledge Identification</u>

Knowledge identification works context-free over the prototypes and hierarchies to provide knowledge which pertains to the input. No attempt has been made to structure this knowledge into a meaning representation. Access to, and structuring of, real-world knowledge is an application dependent aspect of language analysis, which could hopefully operate more easily subsequent to a knowledge identification phase.

For an input triple, the processor examines the prototypes in the system, searching for one which can act as a target. Because the prototypes are hierarchically organized, the initial search concerns a relatively small number of highly generalized prototypes. These generalized prototypes are provided as the possibilities for the input verb. After such a prototype has been located, it may be instantiated with the aid of the semantic categories of the words used in the subject and object positions of the input. For example, for the input TOM DRINKS TEA, a prototype such as (ANIMATES CONSUME FOOD-SUBSTANCES) might be initially considered, and of course, TOM and TEA meet the requirements. Afterwards, a more specific prototype might be used, such as (ANIMALS INGEST FOOD-SUBSTANCES), or eventually (HUMANS DRINK LIQUIDS).

This example demonstrates another aspect of the process involved, that of determining the acceptability of a prototype as a target. Generally, a prototype can be a target if each of its subject, verb, and object components <u>traces</u> to the subject, verb and object of the input. In most cases the verbs must be identical, but some leeway is provided for the other two components. This section provides a sample of the tracing conditions which will permit some metaphoric

[1] The example shown here is a condensed form of the output from the implementation (see Browse 1977).

language use.

A. Tracing via hierarchical inclusion

As described in the first section, case role fulfillment is usually determined by simple hierarchical inclusion properties. This is a major means of tracing, and one which provides all the literal meanings that the system can accept. For the above example:

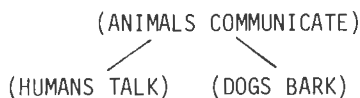
input : TOM DRINKS TEA
target: HUMANS DRINK LIQUIDS
others: TOM IS HUMAN
        TEA IS LIQUID

B. Jumping in hierarchies

Sentences such as:

John drank in the sunshine.
My car drinks gasoline.
The sargeant barked.

are all acceptable because of a close hierarchical connection between the verb which is used, and the verbs which could be substituted to develop a literal meaning. These connections are given in the connections between the prototypes.



In processing an input such as (SARGEANT BARK nil) an initial, generalized prototype (ANIMALS COMMUNICATE) will be acceptable on the basis of the hierarchical connections between the subjects of the input and target (tracing method A above). However, the system will fail to instantiate either of the more specific prototypes.

If the system were allowed to branch to a prototype whose verb differs from the input, but to one for which the subject traces easily, then one could obtain:

input : SARGEANT BARKS
target: HUMANS TALK
others: TALK and BARK are both COMMUNICATE
        SARGEANT IS HUMAN

This particular jump in the hierarchical organization suggests a similarity between the given act of barking and the known act of talking.

On the other hand, the system could be required to branch for instantiation strictly according to the input verb, and the rules for tracing the subject could be extended to include moving up and then down the hierarchical organization of noun concepts. For example:

input : SARGEANT BARKS
target: DOGS BARK
others: SARGEANT IS HUMAN
        DOG and HUMAN are both ANIMALS

This type of jump in the hierarchy suggests a similarity between SARGEANT and DOG rather than a similarity between the acts of talking and barking.

These operations involve an important part of metaphoric language use, particularly in personification.

C. Tracing through other prototypes

Sentences such as:

Sam read Shakespeare.
Jan drank the cup.

are acceptable not because of any hierarchical relations between the objects that were expected for the verbs and the ones that were given, but because of other relations. For example, for the input JAN DRINKS CUP, the prototype (HUMANS DRINK LIQUIDS) will be considered, and could be accepted if some other prototype links CUP to LIQUID. This is the case if the system has a prototype such as (CONTAINERS CONTAIN LIQUIDS), providing CUP is an instance of CONTAINER.

input : JAN DRINK CUP
target: HUMAN DRINK LIQUID
others: JAN IS HUMAN
        CONTAINERS CONTAIN LIQUIDS
        CUP IS CONTAINER

In the above example, it is important to note that not only was the technique of using another prototype involved in the tracing, but, as a sub-trace, hierarchical inclusion was used (CUP IS CONTAINER). This points up the recursive nature of the application of such rules in accepting prototypes.

consider again:

```
input : DICK PLAYS BEETHOVEN
target: HUMANS PLAY MUSICAL-INSTRUMENTS
others: DICK IS HUMAN
        MUSICAL-INSTRUMENTS EMIT MUSIC
        COMPOSERS WRITE MUSIC
        BEETHOVEN IS COMPOSER
```

The long trace between BEETHOVEN and MUSICAL-INSTRUMENT involves the use of other prototypes twice, and hierarchical properties once.

The processor is "cooperative", that is, it assumes that all inputs are meaningful, and attempts to find the appropriate target and traces. If an input is meaningless relative to the prototypes which the system contains, it might a long and obscure trace. Thus it is essential that the most direct methods, yielding literal meanings are tried first, and that some cutoff point be established for the length of traces.

Beyond This Format

Many of the interesting categories of metaphoric language involve accessing some fact which connects two concepts, but those connections cannot always be analyzed within the restricted format of the prototype system as described above.

Simile does not involve substitution of one concept for another, but is rather a direct statement of similarity. For example:

> He ran like lightning.
> The birches stood like frozen feathers.
> His hair was like snow.

It is interesting to notice that properties other than hierarchical categories are most often accessed in the use of simile. For the simile "hair like snow", it is only one property that hair and snow share that is of interest. It seems as if these types of properties are not appropriate to permit direct substitution, such as "he combed his snow". On the other hand, non-substitution comparison via simile seems to point up differences, rather than similarities between hierarchically related concepts. For example "his dog was like a horse" alludes to the size difference.

Metonymy, or the referring to an accompaniment rather than the object itself, most often involves knowledge which cannot be represented in simple subject-verb-object prototypes. Some examples are:

> The Whitehouse issued a denial.
> The Yankees fear Bench's bat.
> The town heard the trumpet blast.

The last sentence requires knowledge such as (HUMANS LIVE (in TOWNS)) to allow the acceptance of a target prototype such as (HUMANS HEAR SOUNDS).

Synecdoche is a special kind of metonymy which refers to the part instead of the whole or vice-versa. For example:

> our daily bread.
> All hands on deck.
> My wheels broke down.
> Here comes the Navy.

This is a common type of metaphoric language. The frequency of this type of metaphor suggests that the PART-OF relation may require a special status along with the IS-A relation in the development of meanings. This also leads to the possibility of other relations, such as causality, being singled out. Extending this to the limit, and converting all prototypes into network relations, would develop a strong similarity to the system proposed by Collins and Quillian (1972). A more productive direction might be to represent all relations, including IS-A as prototypes.

Discussion and Conclusions

The use of strict requirements for fulfillment of selectional restrictions during the process of recognition of meaning has been shown responsible for a difficulty in dealing with the type of everyday metaphoric word-use which is important to natural language. This paper suggests some simple, uniform operations over an organization of known language-use situations which can provide the basis for understanding those metaphors. Yet the only knowledge stored in the system is literal meanings, and thus the knowledge associated with any particular verb is centralized.

In order to explore those possibilities, several simplifications have been made. The roles that nouns play in meaning have been reduced to only subject and object. The specification of these roles could be extended into a full case system. Processes have been described which

contribute towards the development of meaning for inputs. The fact that these processes operate context-free should not be taken as a belief that sentence meaning can be derived from word meaning alone. However, the identification of a relevant set of known language-use templates could be a useful context-free step.

Several observations can be made concerning the type of organization and use of knowledge in the system of prototypes. These observations point up speculative aspects of the usefulness of such a system.

During the matching process, prototypes not only serve as targets, but also can be used as the knowledge necessary to permit the acceptance of some other target. For example:

```
  input: JAN DRINK CUP
 target: HUMAN DRINK LIQUID
 others: JAN IS HUMAN
         CONTAINERS CONTAIN LIQUID
         CUP IS CONTAINER

  input: BOTTLE CONTAIN BEER
 target: CONTAINER CONTAIN LIQUID
 others: BOTTLE IS CONTAINER
         BEER IS LIQUID
```

Note that CONTAINERS CONTAIN LIQUID appears both as a target and as tracing information. It follows that for any given prototype, new shades of meaning of the verb may be introduced through the addition of knowledge which has no direct relation to the verb itself. This is a seductive idea in that it coincides with intuitive notions about how humans manipulate knowledge.

Prototypes are retained at several levels of generalization. This provides the basis for the development of meaning only to a desired level of instantiation. Clues directing the instantiation could come from either the context, or other parts of the sentence or discourse.

It is not a coincidence that this system provides possibilities for the examination of questions relating to learning. Learning and comprehension are two side of the same coin, and while both problems need not be dealt with at the same time, at least a degree of compatibility should be maintained. Language learning must involve the ability to develop generalized knowledge about the extent of word use from more specific knowledge. Not much can be said about the actual process of generalization, but the types of processes described which use prototypes do contribute towards an understanding of what the goals of such generalization processes might be.

The application of each type of rule for tracing the subjects and objects of an input to a prototype provides successively more obscure understandings[1]. A measure of this obscurity could provide a criterion for the inclusion of inputs as new prototypes. Completely straightforward inputs (for example, an input identical to a prototype) would not need to be included, and inputs requiring very obscure tracing would also not need to be included. However, those falling in the mid-range of being novel, but not too obscure could be incorporated.

The notion of matching an input to a target prototype corresponds well to the Piagetian idea of "assimilation", which is the processing of experiences to conform to existing methods for dealing with the environment (see Piaget 1952, 1954). Moore and Newell's MERLIN (1974) uses a process of matching which permits assimilation of one concept to another, whereas the approach outlined here deals with the assimilation of simplified sentences. In addition, the criterion for incorporating new inputs on the basis of their familiarity also corresponds to the Piagetian notion of "accommodation" (or changing of existing methods), which operates to reduce the effort required in future assimilation tasks.

Primitive semantic categories are often justified in terms of the properties of the objects being categorized. The connections between hierarchies and the system of prototypes suggests that semantic categories are connected to their use in partitioning relations. Thus a direction has been indicated for a more uniform specification of these two aspects of knowledge.

The system described here provides insight

---

[1]A simple weighting scheme was used in the computer model (Browse 1977) for which the match involving the lowest measure usually provided the intuitively best target.

into the possibilities for a natural language understanding system which could accept inputs on the basis of developing their meaning rather than recognizing their meaning. The result is a uniform processor which permits metaphoric language use, while at the same time centralizes the knowledge contained about verbs.

Acknowledgements

I wish to acknowledge the encouragement and valuable criticisms provided by the A.I. community at U.B.C. In particular I wish to thank Richard Rosenberg for his help and guidance.

References

Bobrow, D.G.
1975 Dimensions of Representation. in Bobrow, D.G. and Collins, A. (eds) Representation and Understanding. Academic Press.

Browse, R.A.
1977 A Knowledge Identification Phase of Natural Language Analysis. Unpublished Masters thesis, University of British Columbia, January 1977.

Collins, A.M. and Quillian, M.R.
1972 How to Make a Language User. in Tulving, E. and Donaldson, W. (eds) Organization of Memory. New York, Academic Press.

Moore, J. and Newell, A.
1974 How Can MERLIN Understand? in Gregg, L.W. (ed) Knowledge and Cognition. Potomac, Maryland, Lawrence Erlbaum.

Piaget, J.
1954 The Construction of Reality in the Child. Basic Books, New York.

Piaget, J.
1952 The Origins of Intelligence in Children. International University Press.

Wilks, Y.
1977 Knowledge Structure and Language Barriers. Proc IJCAI-77 PP.151-8.

Woods, W.A.
1968 Procedural Semantics for a Qeustion-Answering Machine. Proc AFIPS Conference Vol 33 pp.457-71.

REPRESENTING AND ORGANISING FACTUAL KNOWLEDGE IN PROPOSITION NETWORKS

Randy Goebel

Computer Science Department
University of British Columbia
Vancouver, British Columbia, CANADA

Nick Cercone

Department of Computing Science
Simon Fraser University
Burnaby, British Columbia, CANADA

ABSTRACT

The use of extended semantic networks (more accurately called proposition networks) for representing and organising factual knowledge in a machine is reported. Proposition networks possess expressive power akin to higher-order and modal logics while retaining the methodological advantages of earlier (semantic) network formalisms. Preliminary steps have been taken to imbue them with an organisational structure which facilitates the efficient concept-based access of facts relevant to an arbitrary query. The notion of a topic predicate will form the basis for the development of a topic hierarchy organisation which can be superimposed on a proposition network in order to classify topically-related propositions into similar categories.

1. Introduction

Extended semantic networks (more accurately called proposition networks) are particularly well suited to serve as a notational basis for a state-based natural language representation paradigm (vide Cercone and Schubert, 1975; Cercone, 1975). Certainly a rather different representation might be adopted to meet specific needs in specific applications of semantic networks. Also a different or more extensive set of propositional operators might be formulated. For example, the following operator ("subconcept") may well be useful in representing the essential meaning relationships among concepts: an n-ary relation P(x1,...,xn) is a SUBCONCEPT of an n-ary relation Q(x1,...,xn) if (Ax1)...(Axn)[P(x1,...,xn) => Q(x1,...,xn)]. Thus "crow" is a subconcept of "bird" (i.e., crows are necessarily birds), "drinks" is a subconcept of "ingests" (i.e., if x drinks y then x necessarily ingests y), etc.. The point is that this operator and other compound operators can be expressed in terms of the basic network notation.

A variety of issues in the representation of informal knowledge could raise additional notational problems. Examples are the handling of vagueness, events, the lexical meanings of complex concepts, and overall knowledge organisation. Beyond these relatively static issues lie the more dynamic issues of actual language interpretation and generation, plausible inference, learning, and the interplay between procedural and factual knowledge. Clearly any questions about representation raised by these problem areas can only be answered in the context of particular approaches towards the solutions of the problems themselves. One such approach is discussed in Cercone (1975).

The construction and interconnections of propositions, the basic structural unit, comprise the network. To attain efficient access of facts relevant to an arbitrary query, the knowledge encoded in proposition networks must be organised. Schank (1975) remarks "We can see at once that a [hierarchical] organisation will not work for verbs or for nouns that are abstract or for nouns that do not easily submit to standard categories (such as teletypes)". Nevertheless "drinks" can surely be thought of as a subconcept of "ingests" and while some categories (teletype) would be difficult to fit into a SINGLE hierarchy, they certainly can be fitted into various hierarchies (machine, transducer, communications system, etc.). We can easily impose a hierarchical (subconcept-superconcept) structure on top of the general concepts in memory (as shown in Figure 1) as a heuristic device.

Our approach to organising the knowledge encoded in proposition networks centers around the concept of a topic predicate. Briefly, topic predicates are higher order predicates which will be used to classify first order predicates into different categories much like first order predicates classify individuals. Topic predicates will be structured into a topic hierarchy which corresponds to a higher order type hierarchy. Whereas individuals (e.g., Clyde, ball37) occupy the lowest position in standard type hierarchies, in the topic hierarchy that position will be occupied by first order predicates. An instance of such a topic hierarchy will be interposed between each network concept and its related propositions in order to provide a secondary indexing scheme for the concept-indexed propositional knowledge. The topic hierarchy approach is illustrated throughout sections 3 and 4.

The Symbol-Mapping problem is described as the problem of providing fast access to propositions relevant to a query, and the advantages of a topic hierarchy solution over some other solutions (e.g., McDermott, 1975b) are discussed in section 3.2. A brief discussion of the notation and representation utilised follows immediately.

2. Representing Knowledge in Proposition Networks

The results of semantic analysis of natural language utterances can be represented in proposition-based semantic networks (Schubert, 1975). The proposition network is a notation for representing meaning. This notation encodes any proposition that can be expressed in natural language. In addition, distinctions between utterances with distinct meanings and between distinct readings of ambiguous utterances are preserved. For example, the sentence "All dogs chase some cat" has two possible meanings; our approach easily encodes either reading (see Schubert, 1975 for details). Furthermore, judgments of truth and falsity about natural language utterances appear to correspond appropriately to truth and falsity in the formal representation. The network notation can be viewed as a computer-oriented logic notation with concept-based indexing of propositions.

Semantic networks have proven utility as propositional representations in language understanding (Cercone 1975; Schubert 1975; Schubert et al. 1978b; Schank 1972), concept learning programs (Winston 1970), deduction (McSkimin and Minker 1977), and in psychological cognitive theories (Anderson and Bower 1973; Wilson, to appear).

2.1 Notation

The nonlinear fashioning of proposition networks presents special problems with respect to the representation of logical connectives, quantifiers, descriptions, modalities, and certain other constructions. These constructions are often useful and sometimes necessary to explicate the meaning of complex concepts. Schubert (1975) has proposed systematic solutions to these problems by extending the expressive power of (more or less) conventional semantic network notation. Woods (1975) independently made a series of similar proposals. Only the elementary part of the Schubert formalism, as much as is needed to clarify any misconceptions that may arise from the figures used in this paper, is explained.

In the network notation, the distinction between labels designating storage locations and labels designating pointers to storage locations requires clarification. This distinction is used by Quillian (1968) to designate "type nodes" (unique storage locations) versus "token nodes". The notation can be made uniformly explicit as in Figure 2. Here "parent-of", which corresponds in some notations to a token node, designates a type node (as suggested by Winston, 1970). All encircled nodes correspond to storage locations and all arrows to addresses of storage locations. What formerly were token nodes are now called "proposition nodes"; they serve as graphical nuclei for propositions as a whole.

The explicit notation of Figure 2 can be uniformly condensed for visual effect as illustrated by Figure 3. Of course such a tactic is unnecessary for machines; explicit propositions underlie this abbreviated form.

In Figure 2, A, B, and PRED are mere distinguishing marks. They are analogous to parentheses or commas in the Predicate Calculus in that they serve to relate denoting terms syntactically; they are non-denotative themselves. Whenever possible they will be chosen to be suggestive, but they could be chosen as numeric labels as well (as McDermott (1976) apparently suggests).

One advantage of the explicit notation of Figure 2 is that it works for n-ary (n>2) predicates. The sentence "John gives the book to Mary" involves "gives" as a three place predicate, as diagrammed in Figure 4. Figure 4 is appealing because of the significance we can attach to the labels - 'agent', 'object', and 'recepient'. Nevertheless, Figure 4 is not a graphical analogue of "case-structured" grammars. Cases are not viewed as "conceptually primitive binary relations" as Fillmore (1968) and researchers influenced by him, notably Schank (1972), view them. See Cercone and Schubert (1975) and Bartsch and Vennemann (1972) for a further discussion of cases.

In the proposition network time can be used in two modes - 'instantaneous' and 'interval'. In the instantaneous mode a proposition may have either a fixed or variable moment (duration) of time. The interval mode replaces a "moment" with a time interval. A time interval may be omitted in contexts where it would normally appear in order to simplify propositions representing more enduring properties (e.g., being a girl, a car, etc.). The omission is a matter of notational expediency; any change involving metamorphosis, such as a girl becoming a woman or a caterpillar becoming a butterfly, would require explicit rendering of time dependencies.

Temporal relations can be established using these definitions of instantaneous and interval time. Tenses are built from more elementary temporal relations. If we restrict our view of time to a set of elements (time points) and a relation that partially orders them, we can define binary temporal relations similiar to those of Bruce (1972) or Schank et al. (1973).

The use of time in the representation of states, actions, and events is briefly discussed in the next section. Temporal considerations concerning the representation of complex concepts and the meaning of words are presented in Cercone (1975).

Some final notational conventions by way of introduction need to be made. To avoid confusion, predicate names will be designated by lower case letters and markers by upper case letters. Other conventions that are used include: solid loops for propositions, individual concepts, and existentially quantified concept nodes; broken loops for universally quantified concept nodes; solid lines to link propositional constituents to a proposition node; dotted lines for scope dependency links joining universally quantified nodes with dependent existentially quantified nodes; and dashed lines for linking logical operators and connectives with proposition nodes. Observe the performance of proposition networks in handling states, actions, events, and intentionality.

2.2 States, Events, Actions, Causes, and Intentions

Recognising that "actions" in Schank's (1972) sense are essentially states rather than events is important, since it leads to a uniform view of all (true) events as sequences of states. This recognition has been well documented in Cercone and Schubert (1975). The issue of identifying "actors" of events does not arise, nor is it necessary to delineate the spurious boundary between "passive" and "dynamic" states.

Temporal expressions are important to the meaning of utterances, serving for example to structure events. Nevertheless in other representations, particularly Schank's (1972) Conceptual Dependency networks, aspects of time are handled as a quantificational apparatus. Schank's use of the conceptual tenses 'timeless' and 'continuing' illustrate this. Drawing inferences can be problematic if time is allowed to assume this protean character. For example, the use of the conceptual tense 'timeless' to denote "habitual actions" as in the statement "John sells cars" fails to acknowledge the progressive aspect "is selling" which is concerned with the state and not the disposition. Surely there may have been a time when John did not sell cars and a time when he may not sell cars. We fare no better, unfortunately, if we say "John now sells cars" reporting an incident of behaviour instead of an evolving pattern of behaviour since "now" sets no temporal boundaries (vide Strawson, 1959). Within a particular approach to representation we can, however, artificially contrive boundaries for "now" or, at worst, defend some doctrine for the indefiniteness of ordinary language with regard to time (the latter being unsatisfactory for our purposes).

In the state-based approach time is regarded as the only situational (cf. McCarthy and Hayes, 1969) or contextual variable that needs to be added to action propositions. This is in contrast to Anderson and Bower (1973), Rumelhart et al. (1972), and Schank et al. (1973), who add locale as well as time to the basic dimensions of events. But locale is not a property of events as a whole, but rather a (frequently time-dependent) property of the participants in an event. For example, in "John is watching a circling hawk" it is John and the hawk who have locations, not the event.

We now illustrate our representation of states and events. We regard any condition which can hold momentarily (blue, moving, running, etc.) as a "state". Accordingly, any atomic proposition which is based on a time-dependent predicate is a "state proposition". Figure 5 shows two concurrent state propositions: something (the redness of the sun) was increasing throughout some time interval and something else (the distance between the sun and the horizon) was decreasing throughout the same time interval. Actually there are two additional state propositions concerned with the existence of unique values of redness and distance at all moments of time within the time interval of interest, these have not been made explicit since they can be taken to be implicit in the redness and distance relations.

"Events" involve a change in state as "the last leaf fell from the tree" illustrates. The definitive characteristic of state changes is the following: if a system has property A at time t1, and property B at time t2, then A-->B is a change of state if and only if A and B are mutually exclusive properties, e.g., A=solid, B=liquid; A=round, B=rectangular. In fact a state attribute such as colour which can assume various values may be consistently defined as a set of mutually exclusive properties, each member of the set being regarded as a value of the attribute. This admits both qualitative attributes such as colour as well as quantitative attributes such as location. Figure 6 shows a simple event involving a single change of state of a "system" with one component (Mary). The time relation "then" implies immediate succession of the two time intervals. Our representation of one of Schank's standard sentences is shown in Figure 7. An explanatory paraphrase is the following. "Some unknown mode of behaviour of John caused some object to move quickly toward Mary. Subsequently the object reached Mary and exerted a force on her." Note that we have a state and an event here, viz. John's unknown state and the event of the object moving toward Mary and striking her. The causal connections between John's state and the ensuing event does not make John's state part of that event. Only exclusive and successive states of a particular system of objects form events. A natural inference would be that John intentionally hit Mary, i.e., that the missing state of John is that he was trying to bring about the event in question. We would represent "trying" by the state predicate "x has active goal y at time t".

An important consequence of our very broad conception of states is that new complex states (modes of behaviour) can be defined in terms of events involving primitive or already defined states. The time of occurrence of these events may extend some distance backward and forward from the moment at which the new state is defined to hold. Complex dynamic states such as walking, running, dancing, tumbling, flickering, etc., can be constructed in terms of more elementary states. The constructions are necessarily as complex as the states they describe. Complexity can result from the intricate coordination of several simultaneous activities (e.g. "rolling" expresses rotation and translation at coordinated rates), or from complex time dependencies (e.g. flickering), or from both (e.g. "walking" or even "building a snowman").

In previous representations (e.g., Schank, 1972; Wilks, 1973) complex concepts, such as walking, were incompletely "defined" in dictionaries. Presumably these dictionary entries were not intended to capture full meanings as we seem to understand them, but only those aspects which are most essential to language understanding and (immediate) inference. Nevertheless, much more information will surely be required to match the human ability to describe concepts and reason about them or even to adequately comprehend "ordinary" discourse. The state-based formalism clearly allows for the formation of more complete meaning representations. At the same time, it is capable of accomodating large amounts of information about complex concepts without loss of computational efficiency in the use of those concepts. For a

detailed example illustrating this point, see Cercone (1975).

3. Organising Knowledge in Proposition Networks

3.1 Semantic Network Organisations

As originally conceived by Quillian (1968), the characteristic concept-centred organisation of semantic networks does not address representation issues but rather focuses primary concern with organising knowledge for effective use. Subsequent semantic network notations have been developed in an independent and application specific manner (e.g., Anderson and Bower, 1973; Winston, 1970) often indicating a disdain for classical propositional knowledge representations such as the predicate calculus. The resulting efforts have generally been found to be epistemologically inadequate (in the sense of McCarthy and Hayes, 1969) and expressively weak with respect to standard logical representations. Moreover, they have often blurred the important distinction between the representational and organisational aspects of network devices.

Early efforts by Shapiro (1971) to imbue networks with increased logical power explicitly documented this distinction by contrasting "system relations", "items", and "item relations". In Shapiro's MENS (MEmory Network Structure) data structure, system relations were user defined pointers used for structuring items and item relations into propositions and for indexing propositions via their item participants. Schubert (1975) further clarified the distinction by demonstrating that a logical representation couched in network form could offer the advantages of a classical propositional representation (e.g., be formally interpretable and expressively adequate) while retaining the methodological advantages of the associative network organisation. In addition, his notation clearly indicated that an "intelligent" indexing scheme coupled with a database of logical formulae could indeed be considered to be a kind of semantic network.

The basic distinction between the propositional content of a knowledge database and the access mechanism to that content has recently been noted by Bobrow and Winograd (1977). They state: "In most existing AI systems (and models of human memory) there is an underlying assumption that there is a single set of data linkages, used both for retrieval and for matching and deduction...". They go on to say "We believe that the presence of 'associative links' for retrieval is an additional dimension of memory structure which is not derivable from the logical structure being associated".

We wish to emphasize the organisational aspects of proposition networks, in the tradition of Quillian (1968) and in the spirit of Hayes (1977b). As Hayes (1977b) writes, "If someone argues for the superiority of semantic networks over logic, he must be referring to some other property of the former than their meaning". The correspondence between proposition networks and logic has been made; the meaning of a given network is identical with the meaning of the equivalent logical expression. The object of our immediate attention is that structure which remains after paring the propositional content from a proposition network, i.e., the indexing structure which provides concept-based proposition access.

With the general focus on the organisational aspects of proposition networks, we continue with a brief survey of some recent organisational trends within AI and then elaborate on the structure and use of a topic hierarchy mechanism.

The recent fervour to develop organisational theories of knowledge (e.g., Abelson, 1973; Schank and Abelson, 1975; Minsky, 1975; Mylopoulos et al., 1975; Bobrow and Winograd, 1977; Goldstein and Roberts, 1977) is characterised by the desire to cluster related knowledge into "chunks". Ideally, these "chunks" should reduce the computation required to isolate knowledge relevant in a particular context.

Anticipating such higher-level organisations, Bobrow (1975) asserts: "Predicate calculus and semantic network representions tend to impose only a local organisation on the world". However, both the concept-centred organisation of networks and many of the logical tools of predicate calculus are evident (albeit implicitly) in many of the recent knowledge organisation systems (e.g., Bobrow & Winograd, 1977). For example, in reference to the GUS system (Bobrow et al., 1977) Kay (1976) reports "... now the contents of these slots in the dialog frames (and in lots of other frames that exist in the system) are typically other frames. These structure recurse to great depth. Of course they are not simply tree structures, but they are circular and point to one another; they're networks". Further, Hayes (1977a) provides a translation of the "main features" of KRL-0 into a many-sorted predicate logic, which he takes to be the "external meaning" of KRL expressions.

The remaining salient feature of frame-like systems is simply the idea of grouping pieces of knowledge which may be useful for understanding a particular concept or situation. Hayes (1977a) explains that a frame may be viewed as an n-ary relation between itself and its slots, which themselves may be viewed as binary relations and unary predicates (vide Bundy & Wielinga, 1978). One could therefore represent a frame within the proposition network notation. The major difference between the "frames" view and the network view is one of function versus structure, as noted by Schubert et al. (1978a): "A memory structure is regarded as a frame because of the kinds of knowledge and capabilities attributed to it, rather than because of any specific structural properties".

Of course deciding what knowledge to associate with a frame can be done only in the context for which that organisation will be used. Below we develop an organisation of concept properties in the general context of the so-called "symbol-mapping problem" (named by Fahlman, 1975). A topic hierarchy organisation will provide a meaningful structure for associating each concept with knowledge "about" that concept.

3.2 Symbol-Mapping in the Proposition Network

The ubiquitous notion of structuring general properties of concepts to facilitate their inheritance by related concepts and individuals has been noted by many writers (e.g., Reiter, 1975; McDermott, 1975a; Moore, 1975; Mylopoulos et al., 1975). The main issue of effectively accessing all known concept properties via a newly instantiated individual is inherent in the question posed by Fahlman (1975): When told a fact like "Clyde is an Elephant", how can a system quickly and efficiently provide access to all known Elephant properties via the newly created concept "Clyde"?

Fahlman (1975) proposes a novel solution in which an efficient inheritance of properties scheme is based on a network of parallel hardware elements. Each element represents either a relation (e.g., "IS-A") or a concept (e.g., Clyde, Elephant) and is capable of storing "marker bits" which can be propagated through the network in parallel. The relations shared by two nodes can be found by "marking" the nodes in question, and then broadcasting the "markers" through the network and noting which relation nodes receive intersecting "marker" signals.

Those who have not despaired of a serial solution have concentrated on concept-based indexing to provide efficient access of concept properties. For example, McDermott (1975b) suggests organising concept properties into "packets" or "context" vis a vis the programming language constructs available in CONNIVER (Sussman and McDermott, 1974) or QA4 (Rulifson et al., 1972). When the assertion "Clyde is an Elephant" is made, a new index entry is created which links "Clyde" to the packet containing all known "Elephant" properties. Of course access to the "Elephant" packet does not ensure efficient access of an arbitrary "Elephant" property within the packet; McDermott (1975a) notes that the issue of how to access packets internally is related to the issue of "shallow" versus "deep" binding (vide Moses, 1970).

Moore (1975) proposes a scheme which structures asserted properties around a hierarchy of types (e.g., Elephant IS-A Mammal IS-A Animal ..., etc.). A list of subsuming types is attached to each instantiated constant or variable, e.g., the elephant named "Clyde" might have the attached type list

PHYSOB - ALIVE - ANIMAL - MAMMAL - ELEPHANT.

The pattern matcher is augmented with a type checker which ensures that an individual matches only those properties which may be legally inherit from subsuming types. In this case, since "Clyde" is an instance of type "Elephant", he may inherit any property true of Elephants, Mammals, etc. Moore recognises that the key to the pattern matcher's efficient operation depends on how the data base of properties is indexed. He tentatively suggests that assertions be grouped in hierarchical buckets indexed by type. This corresponds to the subconcept-superconcept organisation for networks, suggested in Section 1.

Symbol-mapping in a proposition network is greatly aided by imposing a subconcept-superconcept hierarchy on the network concepts. Within the proposition network, one concept is a superconcept of another concept if the set of properties attached to the former is a subset of the properties attached to the latter. Therefore "mammal" is a superconcept of "elephant" since the set of "mammal" properties is a subset of the set of "elephant" properties.

After asserting "Clyde is an elephant", the colour of Clyde can be found as follows: the concept node "Clyde" is accessed, and the attached propositions are scanned sequentially for one which indicates the colour of Clyde. Should one not be found, Clyde's immediate successor in the subconcept-superconcept hierarchy is accessed (e.g., elephant), and its attached properties are again searched sequentially for a colour proposition. This process continues until either all the existing superconcepts' propositions have been checked, or a colour proposition has been found. Notice that the subconcept-superconcept structure simply guides an exhaustive search for a colour property attached to each of the superconcept classes of which Clyde is a member. It does not increase the efficiency of locating relevant information about the colour of Clyde. Certainly we would rather ask the question "Is there a colour proposition attached to Clyde?", and if the answer is no, proceed up the subconcept-superconcept hierarchy asking the same question of each successive superconcept.

3.3 Topic Organisations

Subconcept-superconcept hierarchies provide an organisation for associating knowledge with a concept, but that knowledge itself remains unstructured. In order to efficiently answer queries like "What is the colour of Clyde?", the knowledge about each concept must also be organised.

We propose that the first order knowledge about a concept (i.e., attributes or properties) be classified by second order predicates called topic predicates. Subtopic-supertopic relationships between topic predicates will define a topic hierarchy which will classify first order predicates in the same way that standard subconcept-superconcept hierarchies classify individuals. For example, consider the "APPEARANCE" topic illustrated in Figure 8. Except for the leaves, all nodes of the hierarchy are viewed as second order predicates. The first order predicate "striped" is classified as an instance of "PATTERN" which in turn is a subtopic (i.e., second order subconcept) of "APPEARANCE". The first order predicates appearing as leaves inherit the properties of their superiors in the topic hierarchy.

In Figure 8, notice that the first order predicate "shiny" appears as an instance of both "TEXTURE" and "COLOUR", thus providing two viewpoints of the same predicate. In general, the topic organisation need not be strictly hierarchical in order that predicates may be classified under multiple topics. Our first

implementation of topic hierarchies (Goebel, 1977) permitted predicates to be ranked as to their "degree of relevance" in the topics under which they were classified. In an attempt to combine a fuzzy logic model of vagueness with a probabilistic model of uncertainty, relevance rankings were specified as cumulative probability distributions over degrees of relevance. Although it is clear that some measure of relevance is useful in specifying which of several possible viewpoints is most relevant, we are now skeptical of the approach of Goebel (1977). An alternate approach for representing degrees of relevance is provided in Schubert (1978).

Topic hierarchies are designed to impose a classification on each network concept's associated knowledge in order to provide efficient access of topically-relevant propositions for an arbitrary query. To implement such an indexing scheme, an instance of the topic hierarchy called a topic access skeleton will be attached to each individual or type concept in the network. Only those topics under which knowledge has been classified will be instantiated in any particular topic access skeleton. For example, Figure 9 gives the "APPEARANCE" topic access skeleton for "Clyde" after asserting "CLyde is pink" and "Clyde is spotted". Additions to a concept's topic access skeleton are signalled by the appearance of a topically classified predicate in an input proposition. When a proposition about "Clyde" involves a predicate appearing as a leaf of the "APPEARANCE" topic hierarchy, that proposition is inserted in the corresponding position in Clyde's "APPEARANCE" topic access skeleton.

Now reconsider the symbol-mapping problem assuming that the propositions attached to each concept are classified by an appropriate topic hierarchy. A search for the colour of Clyde begins by accessing the "Clyde" concept, but rather than looking at each proposition sequentially, the "COLOUR" topic of Clyde can immediately be checked for a colour proposition. If this fails, each of Clyde's superconcept nodes (i.e., elephant, mammal, etc.) are searched in exactly the same fashion. If the topic access skeletons attached to each concept are approximately balanced, the access time for a classified proposition about a particular concept will be approximately proportional to the logarithm of the number of propositions "known" about that concept. The combined organisational power of the subconcept-superconcept hierarchy and the topic hierarchy should provide for a significant reduction in proposition access time.

4. Utilising Knowledge in Proposition Networks

Once a proposition is obtained from the interpretative program (Cercone, 1975), the concept nodes in the proposition are created unless they are found in the system's internal concept dictionary. An unquantified argument is assumed to be the name of an individual concept, and if it does not exist, it is created as specified. All quantified arguments are entered as new variables of the appropriate type (i.e., existential or universal). Predicates may be optionally created as recognised, or predefined and verified upon input. A functional notation is provided which permits reference to an individual concept by its participation in an atomic network proposition. For example, the expression

(father-of Fred)

is a functional reference to Fred's father, whoever he may be. This facility works only for existing nodes; unresolved functional references do not cause the creation of new nodes. In general, the decision to create a new node will require a cautious approach, since indiscriminant creation of new nodes could quickly lead to data base inconsistency. For example, if the proposition

(Sally sister-of Fred)

has been asserted, the functional references

(father-of Sally), (father-of Fred)

refer to the same father, but reasoning is required to recognise this fact in order to refrain from creating two new individual concepts.

The system facilitates the use of arbitrary topic hierarchies by permitting the explicit insertion of suptopic-supertopic relations. For example, the "APPEARANCE" topic hierarchy of Figure 8 would be defined as follows:

[APPEARANCE supertopic-of COLOUR TEXTURE PATTERN]
[COLOUR supertopic-of yellow pink green]
[TEXTURE supertopic-of shiny rough smooth]
[PATTERN supertopic-of striped spotted checked].

The experimental implementation of Goebel (1977) uses a simple proposition classification scheme based on the recognition of individuals (e.g., Clyde, peanut37) and type concepts (e.g., elephant, mammal, animal) in input propositions. For example, the assertion "Elephants are grey" would be classified under the "COLOUR" topic in the "APPEARANCE" topic access skeleton for the "elephant" concept. A detailed description of an automatic classification mechanism based on the logical form of propositions is given in Schubert et al. (1978b).

A retrieval request for topically classified propositions is specified as a concept name followed by a topic name. The request

Clyde;APPEARANCE

would retrieve all the classified "APPEARANCE" propositions about "Clyde". Clearly, this facility could form the basis for a sophisticated question-answering system.

5. Discussion

We conclude with a short assessment of the most salient features of this research and a brief indication of what we consider to be promising directions for further research. We have argued elsewhere (see Schubert et al., 1978b) that primitive representations for factual knowledge are cumbersome and unneccessary, since the further the

reduction to primitives is carried, the more computation bound the resulting representations become. Clearly, ours is a nonprimitive representation and the mini-implementations developed in Cercone (1975) and Goebel (1977) lend credence to that claim.

Although we have repeatedly stressed the distinction between representations and organisations, we now note that a topic hierarchy is simply a relational structure which can of course be represented within the proposition network. This approach would allow a system to reason ABOUT topics. We believe that existing knowledge representations which do not initially (or ever) make such a distinction (e.g., McCalla, 1977) can avoid needless complexity with such an approach.

One cannot overstress the advantages of a logical approach to developing and investigating knowledge representations and organizations. For a further example, observe the clarity of the approach given to the inheritance of properties by Reiter (1975) in comparison to those of McDermott (1975b) or Moore (1975). Also note that the second order nature of topic organisations provide a well-defined notion of "knowledge about knowledge". This should provide important clues to the problem of identifying and using different "levels" of knowledge.

We have concentrated on examples of topic hierarchies for knowledge of attributes or properties, but similar organisations could be constructed for other kinds of knowledge, e.g., knowledge about actions. Note that an organisation similar to a topic hierarchy has been employed by Schank (1972) to organise primary inferences associated with his primitive action concepts. For example, Schank's abstract transfer primitive "ATRANS" may be viewed as a supertopic of the first order predicates "buy", "sell", "trade", etc.. One of the "pieces" of knowledge associated with "ATRANS" which is inherited by the first order concepts is the fact that some object has changed possession.

Hopefully, the logical flavour of the topic hierarchy approach will provide a unifying framework in which to view current and future organisations of propositional knowledge.

ACKNOWLEDGEMENTS

We are indebted to Len Schubert for contributing to the ideas herein; we are always reminded of his guidance, creativity, and example. Also, thanks are due Ray Reiter for his many accurate observations about the logical structure of topic hierarchies.

REFERENCES

Abelson, R. (1973) "The Structure of Belief Systems", in COMPUTER MODELS OF THOUGHT AND LANGUAGE, Schank, R. & Colby, K. (eds.), W. H. Freeman, San Francisco, pp 287-339.

Anderson, J. & Bower, G. (1973). HUMAN ASSOCIATIVE MEMORY, Winston & Sons, Washington, D.C.

Bartsch, R. & Vennemann, T. (1972). SEMANTIC STRUCTURES, Athenaum Verlag, Frankfurt, Germany.

Bobrow, D. (1975). "Dimensions of Representation", in REPRESENTATION AND UNDERSTANDING, Bobrow, D. & Collins, A. (eds.), Academic Press, New York, pp 1-34.

Bobrow, D. & Winograd, T. (1977). "An Overview of KRL, a Knowledge Representation Language", Cognitive Science, 1, pp 3-46.

Bobrow, D., Kaplan, R., Kay, M., Norman, A., Thompson, H. & Winograd, T. (1977). "GUS, A Frame Driven Dialog System", Artificial Intelligence, 8, pp 155-173.

Brady, J. M. & Wielinga, R. (1977). "Reading the Writing on the Wall", in COMPUTER VISION SYSTEMS, Riseman, E. & Hansen, A. (eds.), Academic Press, New York, in press.

Bruce, B. (1972). "A Model for Temporal References and its Application in a Question-Answering Program", Artificial Intelligence, 3, pp 1-26.

Cercone, N. & Schubert, L. K. (1975). "Toward a State-Based Conceptual Representation", Proceedings IJCAI4, Tbilisi, USSR, pp 83-90.

Cercone, N. (1975). "Representing Natural Language in Extended Semantic Networks", TR75-11, Department of Computing Science, University of Alberta, Edmonton, Alberta.

Fahlman, S. (1975). "A System for Representing and Using Real World Knowledge", AI Lab Memo 331, MIT, Cambridge, Massachusetts.

Fillmore, C. J. (1968). "The Case for Case", in UNIVERSALS IN LINGUISTIC THEORY, Bach, E. & Harm, R. (eds.), Holt, Reinhart, and Winston, New York, pp 1-88.

Goebel, R. (1977). "Organizing Factual Knowledge in a Semantic Network", TR77-8, Department of Computing Science, University of Alberta, Edmonton, Alberta.

Goldstein, I. & Roberts, R. (1977). "NUDGE, A Knowledge-Based Scheduling Program", Proceedings IJCAI5, Cambridge, Massachusetts, pp 257-263.

Hayes, P. (1977a) "In Defense of Logic", unpublished draft version of Hayes (1977b).

Hayes, P. (1977b). "In Defence of Logic", Proceedings IJCAI5, Cambridge, Massachusetts, pp 559-565.

Kay, M. (1976). "Xerox's GUS (Genial Understander System)", Proceedings Symposium on Advanced Memory Concepts, SRI, Menlo Park, California, microfich 5, pp 351-360.

McCalla, G. (1977). "An Approach to the Organization of Knowledge for the Modelling of Conversation", Ph.D. Thesis, Department of Computer Science, University of British Columbia, Vancouver, British Columbia.

McCarthy, J. & Hayes, P. (1969). "Some Philosophical Problems from the Standpoint of Artificial Intelligence. MACHINE INTELLIGENCE, 4, Meltzer, B. & Michie, D. (eds.), American Elsevier, New York, pp 463-502.

McDermott, D. (1975a). "Symbol-Mapping: a technical problem in PLANNER-like systems", SIGART Newsletter 51, April, pp 4-5.

McDermott, D. (1975b). "A Packet Based Approach to the Symbol Mapping Problem", SIGART Newsletter 53, August, pp 6-7.

McDermott, D. (1976). "Artificial Intelligence Meets Natural Stupidity", SIGART Newsletter 57, April, pp 4-9.

McSkimin, J.R. & Minker, J. (1977). "The Use of a Semantic Net in a Deductive Question-Answering System", Proceedings IJCAI5, Cambridge, Massachusetts, pp 50-58.

Minsky, M., (1975). "A Framework for Representing Knowledge", in THE PSYCHOLOGY OF COMPUTER VISION, Winston, P. (ed.), McGraw-Hill, New York, pp 211-277.

Moore, R. (1975). "A Serial Scheme for the Inheritance of Properties", SIGART Newsletter 53, August, pp 8-9.

Moses, J. (1970). "The Function of FUNCTION in LISP", AI Lab Memo 199, MIT, Cambridge, Massachusetts.

Mylopoulos, J., Cohen, P., Borgida, A. & Sugar, L. (1975). "Semantic Networks and the Generation of Context", Proceedings IJCAI4, Tbilisi, USSR, August, pp 134-142.

Quillian, M., (1968). "Semantic Memory", in SEMANTIC INFORMATION PROCESSING, Minsky, M. (ed.), MIT Press, Cambridge, Massachusetts, pp 227-270.

Reiter, R. (1975). "Formal Reasoning and Language Understanding Systems", Proceedings TINLAP Workshop, Cambridge, Massachusetts, pp 175-179.

Rulifson, J. F., Derksen, J. & Waldinger, R. J. (1972). "QA4: A Procedural Calculus for Intuitive Reasoning", AI Technical Note 73, Computer Science Department, Stanford University, Stanford, California.

Rumelhart, D., Lindsay, P. & Norman, D. (1972). "A Process Model for Long Term Memory", in ORGANIZATION OF MEMORY, Tulving, E. & Donaldson, W. (eds.), Academic Press, New York, pp 198-221.

Schank, R. (1972). "Conceptual Dependency: A Theory of Natural Language Understanding", Cognitive Psychology, 3, 552-631.

Schank, R. (1975). "The Role of Memory in Language Processing", in THE STRUCTURE OF HUMAN MEMORY, Cofer, C. (ed.), W. H. Freeman, San Francisco, pp 162-189.

Schank, R. & Abelson, R. (1975). "Scripts, Plans, & Knowledge", Proceedings IJCAI4, Tbilisi USSR, pp 151-157.

Schank, R., Goldman, N., Rieger, C. & Riesbeck, C. (1973). "Margie: Memory, Analysis, Response Generation and Inference on English", Proceedings IJCAI3, Stanford, California, pp 255-261.

Schubert, L. K. (1975). "Extending the Expressive Power of Semantic Networks", Proceedings IJCAI4, Tbilisi, USSR, pp 158-164.

Schubert, L. K. (1976). "Extending the Expressive Power of Semantic Nets", Artificial Intelligence, 7, pp 163-198.

Schubert, L. K. (1978). "On the Representation of Vague and Uncertain Knowledge", submitted to COLING 78.

Schubert, L. K., Cercone, N. & Goebel, R. (1978a) "The Structure and Organization of a Semantic Net for Comprehension and Inference", TR78-1, Department of Computing Science, University of Alberta, Edmonton, Alberta.

Schubert, L. K., Goebel, R., & Cercone, N. (1978b). "The Structure and Organization of a Semantic Net for Comprehension and Inference", to appear in ASSOCIATIVE NETS - THE REPRESENTATION AND USE OF KNOWLEDGE BY COMPUTERS, Findler, N. V. (ed.).

Shapiro, S. (1971). "A Net Structure for Semantic Information Storage, Deduction, and Retrieval", Proceedings IJCAI2, London, England, pp 512-523.

Strawson, P. (1959). INDIVIDUALS, Methuen, London, England.

Sussman, G. J. & McDermott, D. (1974). "CONNIVER Reference Manual", AI Lab Memo 259A, MIT, Cambridge, Massachusetts.

Wilks, Y. (1973). "Preference Semantics", Stanford AI Project, AIM-206, Stanford University, Stanford California.

Wilson, K.V. (1977). FROM ASSOCIATION TO STRUCTURE, (to appear).

Winston, P. (1970). "Learning Structural Descriptions from Examples", Ph.D. Thesis, MAC-TR-76, MIT, Cambridge, Massachusetts.

Woods, W. (1975). "What's in a Link: Foundations for Semantic Networks", in REPRESENTATION AND UNDERSTANDING, Bobrow, D. & Collins, A. (eds.), Academic Press, New York, pp 35-82.

FIGURES

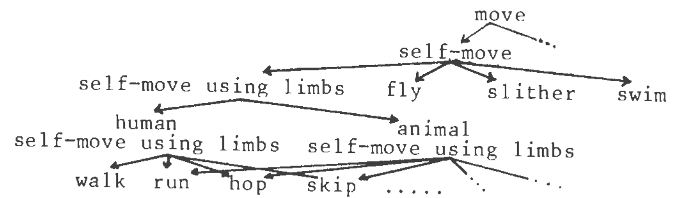


Figure 1. Superimposed Hierarchical Structure

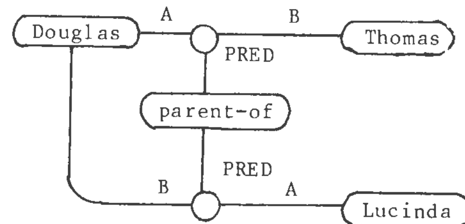


Figure 2. "Douglas is the parent of Thomas. Lucinda is the parent of Douglas."

Figure 3. "Douglas is the parent of Thomas."

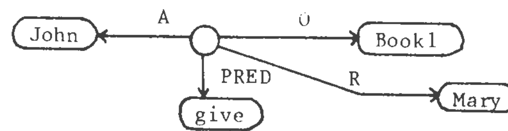


Figure 4. "John gives the book to Mary"

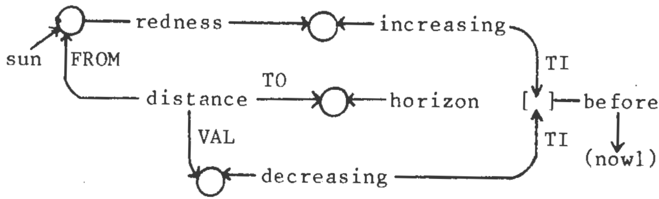


Figure 5. "The sun was getting redder and approaching the horizon."

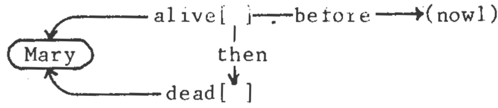


Figure 6. "Mary died."

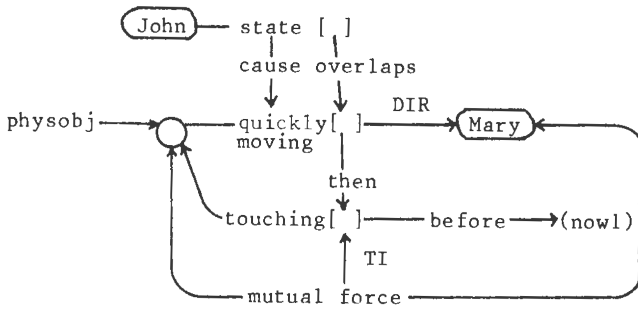


Figure 7. "John hit Mary"

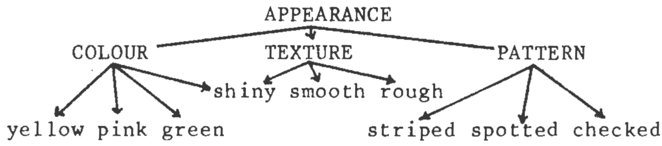


Figure 8. The "APPEARANCE" topic hierarchy.

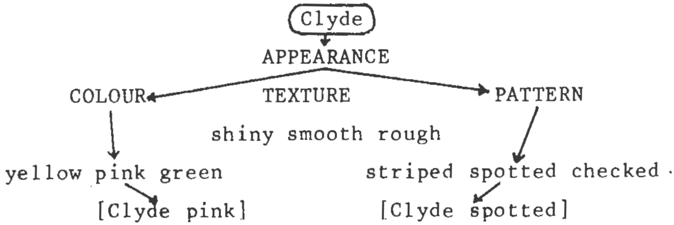


Figure 9. Clyde's partial topic access skeleton.

Capturing Linguistic Generalizations in a Parser for English

Mitchell Marcus
MIT Artificial Intelligence Laboratory

This paper will discuss some aspects of a grammar interpreter called PARSIFAL, which is based upon the hypothesis that a natural language parser needn't simulate a nondeterministic machine. This "Determinism Hypothesis" claims that natural language can be parsed by a computationally simple mechanism that uses neither backtracking nor pseudo-parallelism, and in which all grammatical structure created by the parser is "indelible" in that it must all be output as part of the structural analysis of the parser's input. Once built, no grammatical structure can be discarded or altered in the course of the parsing process.

The paper will focus on one aspect of PARSIFAL in particular: that simple rules of grammar can by written for this interpreter which elegantly capture the significant generalizations behind such constructions as passives (including constructions involving "raising"), *yes/no* questions, imperatives, and sentences with existential *there*. (The first two of these constructions will be discussed in this paper; the latter two will not.) These rules are reminiscent of the sorts of rules proposed within the framework of the theory of generative grammar, despite the fact that the rules presented here must recover underlying structure given only the terminal string of the surface form of the sentence. The component of the grammar interpreter which allows such rules to be formulated is motivated by the Determinism Hypothesis; thus, the ability to write such rules provides indirect evidence for the hypothesis. This result also depends in part upon the use within a computational framework of the closely related notions of *annotated surface structure* and *trace theory*, which derive from the recent work of Noam Chomsky; e.g. [Chomsky 73].

This ability to capture generalizations, coupled with the fact that the grammar rules for PARSIFAL are written in an English-like formal language called PIDGIN, has an important practical implication: that a grammar of English written for this parser can be highly perspicuous.

This parser serves as part of a natural language understanding system which serves as front end for the PAL personal assistant program at the MIT AI Laboratory, which is an extension of the NUDGE system documented in [Goldstein & Roberts 77]. The first version of the PAL system has just been completed, it is documented in [Marcus forthcoming], [Bullwinkle 77]. The PAL system is a prototype appointment scheduler; the current version can handle requests such as "I want to schedule a meeting with Ira in his office at 2 p.m. next Wednesday. It should end at 4 o'clock."

The remainder of this paper will sketch the structure of the parser, discuss how that structure is motivated by the "Determinism Hypothesis", and sketch several examples of grammatical generalizations that can be captured within this framework.

Before proceeding with the body of this paper, two other important properties of the parser should be mentioned which will not be discussed here; they are discussed at length in [Marcus 77]:

1)The structure of the grammar interpreter constrains its operation in such a way that, by and large, grammar rules cannot parse sentences which violate either of two constraints on rules of grammar currently proposed by Chomsky as universals of human language, the Specified Subject Constraint and the Subjacency Principle.

2)The grammar interpreter provides a simple explanation for the difficulty caused by "garden path" sentences, such as "The cotton clothing is made of grows in Mississippi." Rules can be written for this interpreter to resolve local structural ambiguities which might seem to require nondeterministic parsing; the power of such rules, however, depends upon a parameter of the mechanism. Most structural ambiguities can be resolved, given an appropriate setting of this parameter, but those which typically cause garden paths cannot.

The Structure of PARSIFAL

PARSIFAL maintains two major data structures: a pushdown stack of incomplete constituents called *the active node stack*, and a small three-place *constituent buffer* which contains constituents which are complete, but whose higher level grammatical function is as yet uncertain.

Figure 1 below shows a snapshot of the parser's data structures taken while parsing the sentence "John should have scheduled the meeting.". Note that the active node stack in shown growing *downward*, so that the structure of the stack reflects the structure of the emerging parse tree. At the bottom of the stack is an auxiliary node labelled with the features *modal*, *past*, etc.,

which has as a daughter the modal "should". Above the bottom of the stack is an S node with an NP as a daughter, dominating the word "John". There are two words in the buffer, the verb "have" in the first buffer cell and the word "scheduled" in the second. The two words "the meeting" have not yet come to the attention of the parser. (The structures of form "(PARSE-AUX CPOOL)" and the like will be explained below.)

```
            The Active Node Stack
             S1 (S DECL MAJOR S) / (PARSE-AUX CPOOL)
                   NP : (John)
            AUX1 (MODAL PAST VSPL AUX) / (BUILD-AUX)
                   MODAL : (should)

            The Buffer
 1 :        WORD3 (*HAVE VERB TNSLESS AUXVERB PRES
                  V-3S) : (have)
 2 :        WORD4 (*SCHEDULE COMP-OBJ VERB INF-OBJ
                  V-3S ED=EN EN PART PAST ED) : (scheduled)

Yet unseen words:   the meeting .
```

Figure 1 - PARSIFAL's two major data structures.

The constituent buffer is the heart of the grammar interpreter; it is the central feature that distinguishes this parser from all others. The words that make up the parser's input first come to its attention when they appear at the end of this buffer after morphological analysis. Triggered by the words at the beginning of the buffer, the parser may decide to create a new grammatical constituent, create a new node at the bottom of the active node stack, and then begin to attach the constituents in the buffer to it. After this new constituent is completed, the parser will then pop the new constituent from the active node stack; if the grammatical role of this larger structure is as yet undetermined, the parser will insert it into the first cell of the buffer. The parser is free to examine the constituents in the buffer, to act upon them, and to otherwise use the buffer as a workspace.

While the buffer allows the parser to examine some of the context surrounding a given constituent, it does not allow arbitrary look-ahead. The length of the buffer is strictly limited; in the version of the parser presented here, the buffer has only three cells. (The buffer must be extended to five cells to allow the parser to build NPs in a manner which is transparent to the "clause level" grammar rules which will be presented in this paper. This extended parser still has a window of only three cells, but the effective start of the buffer can be changed through an "attention shifting mechanism" whenever the parser is building an NP. In effect, this extended parser has two "logical" buffers of length three, one for NPs and another for clauses, with these two buffers implemented by allowing an overlap in one larger buffer. For details, see [Marcus 77].)

Note that each of the three cells in the buffer can hold a *grammatical constituent* of any type, where a constituent is any tree that the parser has constructed under a single root node. The size of the structure underneath the node is immaterial; both "that" and "that the big green cookie monster's toe got stubbed" are perfectly good constituents once the parser has constructed a subordinate clause from the latter phrase.

The constituent buffer and the active node stack are acted upon by a grammar which is made up of pattern/action rules; this grammar can be viewed as an augmented form of Newell and Simon's production systems [Newell & Simon 72]. Each rule is made up of a pattern, which is matched against some subset of the constituents of the buffer and the accessible nodes in the active node stack (about which more will be said below), and an action, a sequence of operations which acts on these constituents. Each rule is assigned a numerical *priority*, which the grammar interpreter uses to arbitrate simultaneous matches.

The grammar as a whole is structured into *rule packets*, clumps of grammar rules which can be activated and deactivated as a group; the grammar interpreter only attempts to match rules in packets that have been activated by the grammar. Any grammar rule can activate a packet by associating that packet with the constituent at the bottom of the active node stack. As long as that node is at the bottom of the stack, the packets associated with it are active; when that node is pushed into the stack, the packets remain associated with it, but become active again only when that node reaches the bottom of the stack. For example, in figure 1 above, the packet BUILD-AUX is associated with the bottom of the stack, and is thus active, while the packet PARSE-AUX is associated with the S node above the auxiliary.

The grammar rules themselves are written in a language called PIDGIN, an English-like formal language that is translated into LISP by a simple grammar translator based on the notion of top-down operator precedence [Pratt 73]. This use of pseudo-English is similar to the use of pseudo-English in the grammar for Sager's STRING parser [Sager 73]. Figure 2 below gives a schematic overview of the organization of the grammar, and exhibits some of the rules that make up the packet PARSE-AUX.

A few comments on the grammar notation itself are in order. The general form of each grammar rule is:

{Rule <name> priority: <priority> in <packet>
<pattern> --> <action>}

Each pattern is of the form :

[<description of 1st buffer constituent>] [<2nd>] [<3rd>]

The symbol "=", used only in pattern descriptions, is to be read as "has the feature(s)". Features of the form "*<word>" mean "has the root <word>", e.g. "**have*" means "has the root "have"". The tokens "1st", "2nd", "3rd" and "C" (or "c") refer to the constituents in the 1st, 2nd, and 3rd buffer positions and the current active node (i.e. the bottom of the stack), respectively. (These tags will also be used in the text below as names for their respective constituents.) The symbol "t" used in a pattern description is a predicate that is true of any node, thus "[t]" is the simplest always true description. Pattern descriptions to be

matched against the current active node and the current S are flagged by "**C" appearing at the beginning of an additional pattern description. The PIDGIN code of the rule patterns should otherwise be fairly self-explanatory.

```
Priority       Pattern                          Action
               Description of:
        1st    2nd    3rd    The Stack
                    PACKET1
5:     [   ] [   ] [   ]                --> ACTION1
10:    [   ]                  [   ]     --> ACTION2
10:    [   ] [   ] [   ] [   ]          --> ACTION3
                    PACKET2
10:    [   ] [   ]                      --> ACTION4
15:    [   ]                  [   ]     --> ACTION5
```

(a) - The structure of the grammar.

{RULE START-AUX PRIORITY: 10. IN PARSE-AUX
[=verb] -->
Create a new aux node.
Label C with the meet of the features of 1st and pres, past, future, tnsless.
Activate build-aux.}

{RULE TO-INFINITIVE PRIORITY: 10. IN PARSE-AUX
[=*to, auxverb] [=tnsless] -->
Label a new aux node inf.
Attach 1st to C as to.
Activate build-aux.}

(b) - Some grammar rules that initiate auxiliaries.

Figure 2

The parser (i.e. the grammar interpreter interpreting some grammar) operates by attaching constituents which are in the buffer to the constituent at the bottom of the stack; functionally, a constituent is in the stack when the parser is attempting to find its daughters, and in the buffer when the parser is attempting to find its mother. Once a constituent in the buffer has been attached, the grammar interpreter will automatically remove it from the buffer, filling in the gap by shifting to the left the constituents formerly to its right. When the parser has completed the constituent at the bottom of the stack, it pops that constituent from the active node stack; the constituent either remains attached to its parent, if it was attached to some larger constituent when it was created, or else it falls into the first cell of the constituent buffer, shifting the buffer to the right to create a gap (and causing an error if the buffer was already full). If the constituents in the buffer provide sufficient evidence that a constituent of a given type should be initiated, a new node of that type can be created and pushed onto the stack; this new node can also be attached to the node at the bottom of the stack before the stack is pushed, if the grammatical function of the new constituent is clear when it is created.

This structure is motivated by several properties which, as is argued in [Marcus 77], any "non-nondeterministic" grammar interpreter must embody. These principles, and their embodiment in PARSIFAL, are as follows:

1) *A deterministic parser must be at least partially data driven.* A grammar for PARSIFAL is made up of pattern/action rules which are triggered when constituents which fulfill specific descriptions appear in the buffer.

2) *A deterministic parser must be able to reflect expectations that follow from the partial structures built up during the parsing process.* Packets of rules can be activated and deactivated by grammar rules to reflect the properties of the constituents in the active node stack.

3) *A deterministic parser must have some sort of constrained look-ahead facility.* PARSIFAL's buffer provides this constrained look-ahead. Because the buffer can hold several constituents, a grammar rule can examine the context that follows the first constituent in the buffer before deciding what grammatical role it fills in a higher level structure. The key idea is that the size of the buffer can be sharply constrained if each location in the buffer can hold a single complete constituent, regardless of that constituent's size. *It must be stressed that this look-ahead ability must be constrained in some manner, as it is here by limiting the length of the buffer; otherwise the "determinism" claim is vacuous.*

The General Grammatical Framework - Traces

The form of the structures that the current grammar builds is based on the notion of *Annotated Surface Structure*. This term has been used in two different senses by Winograd [Winograd 71] and Chomsky [Chomsky 73]; the usage of the term here can be thought of as a synthesis of the two concepts. Following Winograd, this term will be used to refer to a notion of surface structure *annotated by the addition of a set of features* to each node in a parse tree. Following Chomsky, the term will be used to refer to a notion of surface structure annotated by the addition of an element called *trace* to indicate the "underlying position" of "shifted" NPs.

In current linguistic theory, a trace is essentially a "phonologically null" NP in the surface structure representation of a sentence that has no daughters but is "bound" to the NP that filled that position at some level of underlying structure. In a sense, a trace can be viewed as a "dummy" NP that serves as a placeholder for the NP that earlier filled that position; in the same sense, the trace's binding can be viewed as simply a pointer to that NP. It should be stressed at the outset, however, that a trace is indistinguishable from a normal NP in terms of normal grammatical processes; a trace *is* an NP, even though it is an NP that dominates no lexical material.

There are several reasons for choosing a properly annotated surface structure as a primary output representation for syntactic analysis. While a deeper analysis is needed to recover the predicate/argument structure of a sentence (either in terms of Fillmore case relations [Fillmore 68] or Gruber/Jackendoff "thematic relations" [Gruber 65; Jackendoff 72]), phenomena such as focus, theme, pronominal reference, scope of quantification,

and the like can be recovered only from the surface structure of a sentence. By means of proper annotation, it is possible to encode in the surface structure the "deep" syntactic information necessary to recover underlying predicate/argument relations, and thus to encode in the same formalism both deep syntactic relations and the surface order needed for pronominal reference and the other phenomena listed above.

Some examples of the use of trace are given in Figure 3 immediately below.

(1a) What did John give to Sue?
(1b) What did John give *t* to Sue?
|_______________|
(1c) John gave *what* to Sue.

(2a) A book was given Sue.
(2b) A book was given Sue *t*.
|_______________|
(2c) ∇ gave Sue *a book*.

(3a) John was believed to be happy.
(3b) John was believed [$_S$ *t* to be happy].
|_______________|

Figure 3 – Some examples of the use of trace.

One use of trace is to indicate the underlying position of the wh-head of a question or relative clause. Thus, the structure built by the parser for 3.1a would include the trace shown in 3.1b, with the trace's binding shown by the line under the sentence. The position of the trace indicates that 3.1a has an underlying structure analogous to the overt surface structure of 3.1c.

Another use of trace is to indicate the underlying position of the surface subject of a passivized clause. For example, 3.2a will be parsed into a structure that includes a trace as shown as 3.2b; this trace indicates that the subject of the passive has the underlying position shown in 3.2c. The symbol "∇" signifies the fact that the subject position of (2c) is filled by an NP that dominates no lexical structure. (Following Chomsky, I assume that a passive sentence in fact has *no underlying subject*, that an agentive "by NP" prepositional phrase originates as such in underlying structure.) The trace in (3b) indicates that the phrase "to be happy", which the brackets show is really an embedded clause, has an underlying subject which is identical with the surface subject of the matrix S, the clause that dominates the embedded complement. Note that what is conceptually the underlying subject of the embedded clause has been passivized into subject position of the matrix S, a phenomenon commonly called "raising". The analysis of this phenomenon assumed here derives from [Chomsky 73]; it is an alternative to the classic analysis which involves "raising" the subject of the embedded clause into object position of the matrix S before passivization (for details of this later analysis see [Postal 74]).

Some Captured Generalizations

The remainder of this paper will sketch a few examples of grammar rules that explicitly capture, on nearly a one-to-one basis, the same generalizations that are typically captured by classical transformational rules. The central point of what follows is that the availability of the buffer as a workspace, in conjunction with a grammar written in the form of pattern-action rules, makes possible several techniques for writing simple, concise grammar rules that have the net effect of explicitly "undoing" many of the generative grammarian's transformations with much the same elegance.

One *caveat* should be stated at the outset: Not all grammatical processes which are typically expressed as single rules within the generative framework can be so captured within the grammar for this parser, or, I believe, any other. Such processes include the general phenomenon of "WH-movement", which accounts for the structure of WH-questions and relative clauses (at the least), and the problem of prepositional phrase attachment. Thus, while there is a wide range of grammatical generalizations that can be captured within a parsing grammar, it must be conceded that there are important generalizations that cannot be captured within this framework.

There are several techniques made possible by the buffer that will be used repeatedly to capture linguistic phenomena within fairly simple rule formulations. They are:

1) *The ability to remove some constituent other than the first from the constituent buffer*, compacting the buffer and reuniting discontinuous constituents. In natural language, it is often the case that some third structure intervenes between two parts of what is intuitively one constituent. In most parsers, special provisions must be made in the grammar for handling such situations. As will be demonstrated below, the buffer mechanism makes this unnecessary.

2) *The ability to place a trace by inserting it into the buffer* rather than by directly attaching it to a tree fragment. As will be sketched below, this yields a simple analysis of passivization and "raising".

3) *The ability to insert specific lexical items into the buffer*, thereby allowing one set of rules to operate on only superficially different cases. As figure 4 below shows, many grammatical constructions in natural language are best analyzed as slight variants of other constructions, differing only in the occurrence of an additional specific lexical item or two. Given the buffer mechanism, such constructions can be easily handled by doing a simple insertion of the appropriate lexical items into the shorter form of the construction, "transforming" the shorter form into the longer form, allowing both cases to then be handled by the same grammar rule.

1(a) all the boys
(b) all *of* the boys

2(a) I helped John pick it up.
(b) I helped John *to* pick it up.

Figure 4

In what follows below, examples will be given which illustrate points (1) and (2).

Example 1 - Yes/No Questions

In the grammar for this parser, the analysis of a yes-no question differs from the analysis of the related declarative only in the execution of two rules for each sentence type: declaratives trigger the two rules shown in figure 5a below, and yes-no questions trigger the two rules shown in 5b. The differences between the rules for declaratives and yes-no questions are underlined in fig. 5.

{Rule DECL-S in SS-START [=np] [=verb] --> Label C decl, major. Deactivate ss-start. Activate parse-subj.}	{Rule Y/N-Q in SS-START [=auxverb] [=np] --> Label C ynquest, major. Deactivate ss-start. Activate parse-subj.}
{RULE UNMARKED-ORDER IN PARSE-SUBJ [=np] [=verb] --> Attach 1st to c as np. Deactivate parse-subj. Activate parse-aux.}	{RULE AUX-INVERSION IN PARSE-SUBJ [=auxverb] [=np] --> Attach 2nd to c as np. Deactivate parse-subj. Activate parse-aux.}
DECLARATIVES (a)	YES-NO QUESTIONS (b)

Figure 5 - Rules for yes-no questions and declaratives.

What is surprising about these rules is that they obviate the need for the grammar to contain special provisions to handle the discontinuity of the verb cluster in yes-no questions. Consider the following sentence,

(i) Has John scheduled the meeting for Wednesday?

One of the auxiliary parsing rules that should be triggered during the course of the analysis of this sentence is the rule PERFECTIVE, shown below in figure 6. This rule will attach any form of "have" to the auxiliary and label the auxiliary with the feature *perfective* if the *following* word (implicitly a verb) has the feature *en*. It would seem that some provision must be made for the fact that in a yes-no question these two words might be separated by the subject NP, as in (i) above where the verb "scheduled" (which does carry the feature *en*, since *en* is morphologically realized with this verb as "-ed") does not follow "has", but rather follows an intervening NP. As we shall see immediately below, however, no special patch is needed to handle this discontinuity at all.

{RULE PERFECTIVE PRIORITY: 10. IN BUILD-AUX
[=*have] [=en] --> Attach 1st to c as perf. Label c perf.}

Figure 6 - The PERFECTIVE rule requires contiguous verbs.

Let us now trace through the initial steps of parsing (i) and see why it is that no changes to the auxiliary parsing rules are required to parse yes-no questions.

We begin with the parser in the state shown in figure 7a below, with the packet SS-START active. The rule Y/N-Q matches and is executed, labelling S17 with features that indicate that it is a yes-no question, as shown in figure 7b below. (The buffer, not shown again, remains unchanged.) This step of the parsing process is quite analogous to the analysis process for declaratives.

The Active Node Stack (0. deep)
C: S17 (S) / (SS-START)

The Buffer
1 : WORD134 (*HAVE VERB AUXVERB PRES V3S) : (Has)
2 : NP43 (NP NAME NS N3P) : (John)
3 : WORD136 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)

Yet unseen words: a meeting for Wednesday ?

(a) - Before Y/N-Q has been executed.

The Active Node Stack (0. deep)
C: S17 (S QUEST YNQUEST MAJOR) / (PARSE-SUBJ)

(b) - The Active Node Stack after Y/N-Q is executed.

Figure 7

The packet PARSE-SUBJ is now active, and rule AUX-INVERSION matches and is executed; it attaches NP43 to S17. After AUX-INVERSION has been executed, the grammar interpreter notices that NP43 is attached and *it therefore removes NP43 from the buffer.* But now that the subject of the clause has been removed from the buffer, the pieces of the verb cluster "has scheduled" are no longer discontinuous, as the word "has" is now in the 1st buffer cell, and "scheduled" is in the 2nd cell. This is shown in figure 8 below. In effect, the rule AUX-INVERSION, merely by picking out the subject of the clause, has "undone" the subject/auxiliary "inversion" which signals the presence of a question.

The Active Node Stack (0. deep)
C: S17 (S QUEST YNQUEST MAJOR) / (PARSE-AUX)
NP : (John)

The Buffer
1 : WORD134 (*HAVE VERB AUXVERB PRES V3S) : (Has)
2 : WORD136 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)

Yet unseen words: a meeting for Wednesday ?

Figure 8 - After AUX-INVERSION has been executed.

From this simple example, we see that the ability to attach constituents in other than the first place in the buffer to the current active node, in conjunction with the fact that attachment causes a node to be removed from the buffer, compacting the remaining contents of the buffer, allows a key generalization to be captured. Most interestingly, the removal of the subject NP in this case was *not* specifically stipulated by the grammar rule, which merely specified that the second NP in the buffer was to be attached to the dominating S. The deletion followed instead from *general principles of the grammar interpreter's operation.* This latter point is crucial; given a simple statement of the structure of yes-no questions in English, the proper behavior follows from much more general

principles.

Example 2 - Passives and Raising

In this section, I will very briefly sketch a grammatical solution to the phenomena of passivization and "raising" [Postal 74], sentences in which what seems to be the subject of an embedded complement is passivized into the subject position of the higher clause. This analysis, I believe, is simpler than that demonstrated by Woods within his classic paper on the ATN formalism [Woods 70], in that (1) nothing like the register mechanism of the ATN and the related SENDR and LIFTR mechanisms are needed for this solution; and (2) the register resetting involved in Woods' solution is not needed here.

Let us begin with the parser in the state shown in figure 9 below, in the midst of parsing the following sentence:

The meeting was scheduled for Wednesday.

The analysis process for the sentence prior to this point is essentially parallel to the analysis of any simple declarative with one exception: the rule PASSIVE-AUX in packet BUILD-AUX (shown in figure 11) has decoded the passive morphology in the auxiliary and given the auxiliary the feature *passive* (although this feature is not visible in figure 9). At the point we begin our example, the packet SUBJ-VERB is active.

```
        The Active Node Stack ( 1. deep)
         S21 (S DECL MAJOR) / (SS-FINAL)
                NP : (The meeting)
                AUX : (has been)
                VP : ↓
C:      VP17 (VP) / (SUBJ-VERB)
                VERB : (scheduled)

        The Buffer
1 :     PP14 (PP) : (for Wednesday)
2 :     WORD162 (*. FINALPUNC PUNC) : (.)
```

Figure 9 - Partial analysis of a passive sentence: after the verb has been attached.

The packet SUBJ-VERB contains, among other rules, the rule PASSIVE, shown in figure 10 below. As I will show in the next section, this rule by itself is sufficient to account for many of the phenomena that accompany clause-level passivization including the phenomenon of raising. The pattern of this rule is fulfilled if the auxiliary of the S node dominating the current active node (which will always be a VP node if packet SUBJ-VERB is active) has the feature *passive*, and the S node has not yet been labelled *np-preposed*. (The notation "** C" indicates that this rule matches against the two accessible nodes in the stack, not against the contents of the buffer.) The action of the rule PASSIVE simply creates a trace, sets the binding of the trace to the subject of the dominating S node, and then drops the new trace into the buffer.

```
{RULE PASSIVE IN SUBJ-VERB
[** c; the aux of the s above c is passive;
        the s above c is not np-preposed] -->
Label the s above c np-preposed.
Create a new np node labelled trace.
Set the binding of c to the np of the s above c.
Drop c.}
```

Figure 10 - Six lines of code captures np-preposing.

The state of the parser after this rule has been executed, with the parser previously in the state in figure 9 above, is shown in figure 11 below. S21 is now labelled with the feature *np-preposed*, and there is a trace, NP53, in the first buffer position. NP53, as a trace, has no daughters, but is bound to the subject of S21.

```
        The Active Node Stack ( 1. deep)
         S21 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
                NP : (The meeting)
                AUX : (has been)
                VP : ↓
C:      VP17 (VP) / (SUBJ-VERB)
                VERB : (scheduled)

        The Buffer
1 :     NP53 (NP TRACE) : bound to: (The meeting)
2 :     PP14 (PP) : (for Wednesday)
3 :     WORD162 (*. FINALPUNC PUNC) : (.)
```

Figure 11 - After PASSIVE has been executed.

Now rules will run which will activate the two packets SS-VP and INF-COMP, given that the verb of VP17 is "schedule". These two packets contain rules for parsing simple objects of non-embedded Ss, and infinitive complements, respectively. Two such rules, each of which utilize an NP immediately following a verb, are given in figure 12 below. The rule OBJECTS, in packet SS-VP, picks up an NP after the verb and attaches it to the VP node as a simple object. The rule INF-S-START1, in packet INF-COMP, triggers when an NP is followed by "to" and a tenseless verb; it initiates an infinitive complement and attaches the NP as its subject. (An example of such a sentence is "We scheduled John to give a seminar next week".) The rule INF-S-START1 must have a higher priority than OBJECTS because the pattern of OBJECTS is fulfilled by any situation that fulfills the pattern of INF-S-START1; if both rules are in active packets and match, the higher priority of INF-S-START1 will cause it to be run instead of OBJECTS.

```
{RULE OBJECTS PRIORITY: 10 IN SS-VP
[=np] -->
Attach 1st to c as np.}

{RULE INF-S-START1 PRIORITY: 5. IN INF-COMP
[=np] [=*to,auxverb] [=tnsless] -->
Label a new s node sec, inf-s.
Attach 1st to c as np.
Activate parse-aux.}
```

Figure 12 - Two rules which utilize an NP following a verb.

While there is not space to continue the example here in detail, note that the rule OBJECTS will trigger with the parser in the state shown in figure 11 above, and will attach NP53 as the object of the verb "schedule. OBJECTS is thus totally indifferent both to the fact that NP53 was not a regular NP, but rather a trace, and the fact that NP53 did not originate in the input string, but was placed into the buffer by grammatical processes. Whether or not this rule is executed is absolutely unaffected by differences between an active sentence and its passive form; the analysis process for either is identical as of this point in the parsing process. Thus, the analysis process will be exactly parallel in both cases after the PASSIVE rule has been executed. (I remind the reader that the analysis of passive assumed above, following Chomsky, does *not* assume a process of "agent deletion", "subject postposing" or the like.)

Example 3 - Passives in Embedded Complements

The reader may have wondered why PASSIVE drops the trace it creates into the buffer rather than immediately attaching the new trace to the VP node. As we will see below, such a formulation of PASSIVE also correctly analyzes passives like 3b above which involve "raising", but with no additional complexity added to the grammar, correctly capturing an important generalization about English. To show the range of the generalization, the example which we will investigate in this section, sentence (1) in figure 13 below, is yet a level more complex than 3a above; its analysis is shown schematically in 13.2. In this example there are two traces: the first, the subject of the embedded clause, is bound to the subject of the major clause, the second, the object of the embedded S, is bound to the first trace, and is thus ultimately bound to the subject of the higher S as well. Thus the underlying position of the NP "the meeting" can be viewed as being the object position of the embedded S, as shown in 13.3.

(1)The meeting was believed to have been scheduled for Wednesday.

(2)The meeting was believed [$_S$ *t* to have been scheduled t for Wednesday]

(3) ∇ believed [$_S$ ∇ to have scheduled *the meeting* for Wednesday].

Figure 13 - This example shows simple passive and raising.

We begin our example, once again, right after the rule MVB has been executed, attaching "believed" to VP20, the current active node, as shown in figure 14 below. Note that the AUX node has been labelled *passive*, although this feature is not shown here.

```
        The Active Node Stack ( 1. deep)
          S22 (S DECL MAJOR) / (SS-FINAL)
                  NP : (The meeting)
                  AUX : (was)
                  VP : ↓
C:      VP20 (VP) / (SUBJ-VERB)
                  VERB : (believed)

        The Buffer
1 :     WORD166 (*TO PREP AUXVERB) : (to)
2 :     WORD167 (*HAVE VERB TNSLESS AUXVERB
                         PRES ...) : (have)
```

Figure 14 - After MVB has been executed.

The packet SUBJ-VERB is now active; the PASSIVE rule, contained in this packet now matches and is executed. This rule, as stated above, creates a trace, binds it to the subject of the current clause, and drops the trace into the first cell in the buffer. The resulting state is shown in figure 15 below.

```
        The Active Node Stack ( 1. deep)
          S22 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
                  NP : (The meeting)
                  AUX : (was)
                  VP : ↓
C:      VP20 (VP) / (SUBJ-VERB)
                  VERB : (believed)

        The Buffer
1 :     NP55 (NP TRACE) : bound to: (The meeting)
2 :     WORD166 (*TO PREP AUXVERB) : (to)
3 :     WORD167 (*HAVE VERB TNSLESS AUXVERB
                         PRES ...) : (have)

Yet unseen words:   been scheduled for Wednesday .
```

Figure 15 - After PASSIVE has been executed.

The rule SUBJ-VERB is now triggered, and deactivates the packet SUBJ-VERB and activates the packet SS-VP (which contains the rule OBJECTS) and, since "believe" takes infinitive complements, the packet INF-COMP (which contains INF-S-START1), among others. Now the patterns of OBJECTS and INF-S-START1 will both match, and INF-S-START1, shown above in figure 27, will be executed by the interpreter since it has the higher priority. (Note once again that a trace is a perfectly normal NP from the point view of the pattern matching process.) This rule now creates a new S node labelled infinitive and attaches the trace NP55 to the new infinitive as its subject. The resulting state is shown in figure 16 below.

The Active Node Stack (2. deep)

```
         S22 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
                   NP : (The meeting)
                   AUX : (was)
                   VP : ↓
         VP20 (VP) / (SS-VP THAT-COMP INF-COMP)
                   VERB : (believed)
C:       S23 (SEC INF-S S) / (PARSE-AUX)
                   NP : bound to: (The meeting)
```

The Buffer

```
1 :      WORD166 (*TO PREP AUXVERB) : (to)
2 :      WORD167 (*HAVE VERB TNSLESS AUXVERB
                          PRES ...) : (have)
```

Yet unseen words: been scheduled for Wednesday .

Figure 16 - After INF-S-START1 has been executed.

We are now well on our way to the desired analysis. An embedded infinitive has been initiated, and a trace bound to the subject of the dominating S has been attached as its subject.

The parser will now proceed, exactly as in earlier examples, to build the auxiliary, attach it, and attach the verb "scheduled" to a new VP node. After the rules that accomplish this have been executed, the parser is left in the state depicted in figure 17 below. (Note that for the sake of brevity, only the 3 bottommost nodes in the active node stack will be shown in this and all successive diagrams.) The infinitive auxiliary has been parsed and attached, and VP21 is now the current active node, with the verb "scheduled" as main verb of the clause. Again, the auxiliary has been assigned the feature *passive* by the auxiliary parsing rules, although this is not shown in the figure below.

The Active Node Stack (3. deep)

```
         .......
         VP20 (VP) / (SS-VP THAT-COMP INF-COMP)
                   VERB : (believed)
         S23 (SEC INF-S S) / (EMB-S-FINAL)
                   NP : bound to: (The meeting)
                   AUX : (to have been)
                   VP : ↓
C:       VP21 (VP) / (SUBJ-VERB)
                   VERB : (scheduled)
```

The Buffer

```
1 :      PP15 (PP) : (for Wednesday)
2 :      WORD174 (*. FINALPUNC PUNC) : (.)
```

Figure 17 - After parsing the auxiliary and main verb.

The packet SUBJ-VERB, containing the rules PASSIVE and SUBJ-VERB, is now active. Once again PASSIVE's pattern matches and this rule is executed, creating a trace, binding it to the subject of the clause, (which is in this case itself a trace), and dropping the new trace into the buffer. This is shown in figure 18 below. Note that in this figure, as in earlier figures, the lexical NP which is the *transitive closure* of the binding relationship is shown for each trace.

The Active Node Stack (3. deep)

```
         .......
         VP20 (VP) / (SS-VP THAT-COMP INF-COMP)
                   VERB : (believed)
         S23 (NP-PREPOSED SEC INF-S S) / (EMB-S-FINAL)
                   NP : bound to: (The meeting)
                   AUX : (to have been)
                   VP : ↓
C:       VP21 (VP) / (SUBJ-VERB)
                   VERB : (scheduled)
```

The Buffer

```
1 :      NP57 (NP TRACE) : bound to: (The meeting)
2 :      PP15 (PP) : (for Wednesday)
3 :      WORD174 (*. FINALPUNC PUNC) : (.)
```

Figure 18 - After PASSIVE has run on the lower clause.

The remainder of the parsing process proceeds in a fashion similar to the simple passive example discussed above; the rule OBJECTS will attach the trace NP57 as the object of VP21, and the parse will then be completed by grammatical processes which will not be discussed here. The tree structure which results is shown in figure 19 below. (For the sake of brevity, most features have been deleted from this tree.) A trace is indicated in this tree by giving the terminal string of its ultimate binding in parentheses.

```
(NP-PREPOSED S DECL MAJOR)
  NP: (MODIBLE NP DEF DET NP)
     DET: The
     NBAR: (NS NBAR)
        NOUN: meeting
  AUX: (PAST V13S AUX)
          PASSIVE: was
  VP: (VP)
     VERB: believed
     NP: (NP COMP)
        S: (NP-PREPOSED SEC INF-S S)
           NP: (NP TRACE)
                 (bound* to: The meeting)
           AUX: (PASSIVE PERF INF AUX)
              TO: to
              PERF: have
              PASSIVE: been
           VP: (VP)
              VERB: scheduled
              NP: (NP TRACE)
                    (bound* to: The meeting)
              PP: (PP)
                 PREP: for
                 NP: (NP TIME DOW)
                    NOUN: Wednesday
  FINALPUNC: .
```

Figure 19 - The final tree structure.

We have seen that the simple formulation of the PASSIVE rule presented above, interacting with other simply formulated grammatical rules for parsing objects and initiating embedded infinitives, allows a trace to be

attached either as the object of a verb or as the subject of an embedded infinitive, whichever is the appropriate analysis for a given grammatical situation. The PASSIVE rule is formulated in such a way that it drops the trace it creates into the buffer, rather than attaching the trace somewhere in particular in the tree. Because of this, later rules, already formulated to trigger on an NP in the buffer, will analyze sentences with NP-preposing exactly the same as those without a preposed subject. Once again, we see that the availability of the buffer mechanism is crucial to capturing this generalization; such a generalization can only be stated by a parser with a mechanism much like the buffer used here.

Conclusion

This paper has demonstrated that the structure of PARSIFAL, in conjunction with the computational usage of the notion of traces, allows the formulation of linguistic generalizations within the context of the grammar interpreter, resulting in simple, perspicuous rules of grammar which handle complex linguistic phenomena. The perspicuity of the grammar is increased by the fact that each of these rules conflates different linguistic situations; thus, after the passive rule has run the parsing of actives and passives is identical, after the aux-inversion rule has run, the parsing of declaratives and yes/no questions is identical. And finally, the perspicuity of these rules is increased by the fact that they are expressed in a concise pseudo-English grammar language.

In conclusion, let me briefly recapitulate the major points made above:

Of the structures that make up the grammar interpreter, it is the constituent buffer which is most central to the results that are presented in this document. For example, because the buffer automatically compacts upon the attachment of the constituents that it contains, the parsing of a yes/no question and the related declarative will differ in one rule of grammar, with the key difference restricted to the rule patterns and one line of the rules' actions. The yes/no question rule explicitly states only that the NP in the second buffer cell should be attached as the subject of the clause. Because the buffer will then compact, auxiliary parsing rules that expect the subconstituents of the verb cluster to be contiguous will then apply without need for modification.

Another important source of power is the use of traces, especially in conjunction with the use of the buffer. Especially important is the fact that a trace can be dropped into the buffer, thereby indicating its underlying position in a factorization of the terminal string without specifying its position in the underlying tree. From this follows a simple formulation of passive which accounts for the phenomenon of "raising". The essence of the passive rule - create a trace, bind it to the subject of the current S, drop it into the buffer - is noteworthy in its simplicity. Again, the availability of the buffer yields a very simple solution to a seemingly complex linguistic phenomenon.

Acknowledgments

This paper summarizes one result presented in my Ph.D. thesis; I would like to express my gratitude to the many people who contributed to the technical content of that work: Jon Allen, my thesis advisor, to whom I owe a special debt of thanks, Ira Goldstein, Seymour Papert, Bill Martin, Bob Moore, Chuck Rieger, Mike Genesereth, Gerry Sussman, Mike Brady, Craig Thiersch, Beth Levin, Candy Bullwinkle, Kurt VanLehn, Dave McDonald, and Chuck Rich.

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 Advanced Research Projects Agency of the Department of Defence under Office of Naval Research Contract N00014-75-C-0643.

BIBLIOGRAPHY

Bullwinkle, C. [1977] "The Semantic Component of PAL: The Personal Assistant Language Understanding Program," Working Paper 141, MIT Artificial Intelligence Laboratory, Cambridge, Mass.

Chomsky, N. [1973] "Conditions on Transformations", in S. Anderson and P. Kiparsky, eds., *A Festschrift for Morris Halle*, Holt, Rinehart and Winston, N.Y.

Fillmore, C. J. [1968] "The Case for Case" in *Universals in Linguistic Theory*, E. Bach and R. T. Harms, eds., Holt, Rinehart, and Winston, N.Y.

Goldstein, I. P. and R. B. Roberts [1977] "NUDGE, A Knowledge-Based Scheduling Program," Memo 405, MIT Artificial Intelligence Laboratory, Cambridge, Mass.

Gruber, J. S. [1965] *Studies in Lexical Relations*, unpublished Ph.D. thesis, MIT.

Jackendoff, R. S. [1972] *Semantic Interpretation in Generative Grammar*, MIT Press, Cambridge, Mass..

Marcus, M. P. [1977] *A Theory of Syntactic Recognition for Natural Language*, unpublished Ph.D. thesis, MIT.

Marcus, M. P. [forthcoming] *A Progress Report on the Syntax and Semantics of PAL*, Memo ??, MIT Artificial Intelligence Laboratory, Cambridge, Mass.

Newell, A. and H.A. Simon [1972] *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, N.J.

Postal, P. M. [1974] *On Raising*, MIT Press, Cambridge, Mass.

Pratt, V. R. [1973] "Top-Down Operator Precedence", in the proceedings of *The SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, Boston, Mass.

Pratt, V. R. [1976] "CGOL - An Alternative External Representation For LISP Users", Working Paper 121, MIT Artificial Intelligence Laboratory, Cambridge, Mass.

Rustin R., ed. [1973] *Natural Language Processing*, Algorithmics Press, N.Y.

Sager, N. [1973] "The String Parser for Scientific Literature", in [Rustin 73].

Winograd, T. [1971] *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*, Project MAC-TR 84, MIT, Cambridge, Mass.

Woods, W. A. [1970] "Transition Network Grammars for Natural Language Analysis", *Communications of the ACM* 13:591.

SEMANTIC NETWORKS AND THE DESIGN OF INTERACTIVE INFORMATION SYSTEMS

John Mylopoulos
Harry K.T. Wong
Philip A. Bernstein*

Department of Computer Science
University of Toronto

Abstract

Interactive Information Systems (IISs) such as credit-card verification and airline reservation can be viewed as knowledge-based systems that use their knowledge for question-answering and reasoning. This paper describes TAXIS, a language for the design of such systems, that is based on semantic networks and enables the designer of an IIS to tightly integrate data and procedures.

1. Introduction

We are interested in the design of interactive information systems (IISs) which are characterized by their requirement for handling large volume of transactions that are short, very predictable and update intensive. Common examples of such systems include credit-card verification, airline and hotel reservation, point-of-sale inventory control and electronic funds transfer. A main feature of such systems is their use of a database (usually in secondary memory) for maintaining and accessing information.

Our approach to ISS design is to integrate useful concepts from Database Management and Programming Languages using AI techniques related to the problems of knowledge representation and the design of knowledge-based systems. Specifically, from Database Management, we have selected the relational model of data. From Programming Languages, we have adopted the idea of abstract data types. Most importantly, from AI, we have used the techniques and constructs of semantic networks and in particular, the procedural semantic network model developed in [Levesque 77]. The product of our work is a language for IIS design that is simple in structure and (we believe) will be efficient when implemented.

It is true that semantic networks have been used before in modelling databases [Mylopoulos et al. 76], [Roussopoulos 76], [Wong & Mylopoulos 77]. What has been lacking is an integration of procedural aspects of application programming with data structuring aspects of data modelling. Performing this integration is a major theme of our work.

Designing an IIS primarily involves creating a model for an enterprise. A program in this language consists of a definition of the database (i.e. the information) that the system will maintain for the enterprise over long periods of time and a definition of transactions (i.e., procedures) that can be invoked to alter the information in the database.

Our language is called TAXIS (pronounced tak'-siss), a programming language for designing interactive information systems.† In the next section, we sketch the main features of TAXIS; the reader is referred to [Mylopoulos et al. 78a] for a full description. In sections 3 and 4, we explore the relationships between semantic networks and the design of IISs.

2. A Language for IIS Design

2.1 Classes, Tokens and Properties

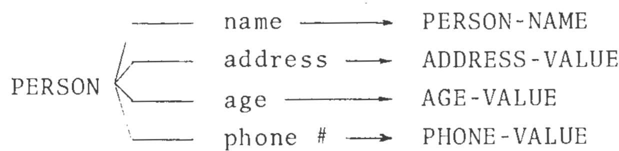
A class is a collection of objects sharing common properties. The instances of a class are its tokens. Collectively, classes and tokens are called objects. One way to describe objects is by their properties. A property is a named relation from a subject to a value and is used to relate classes or tokens. The name of the property is its attribute. For illustrative purposes, we will use a

* Aiken Computation Lab., Harvard University.
† "TAXIS" (ταξισ) is a Greek noun that means order, as in "Law and order", or class as in "social class" or "university class". The word "taxonomy" is a derivative of "taxis".

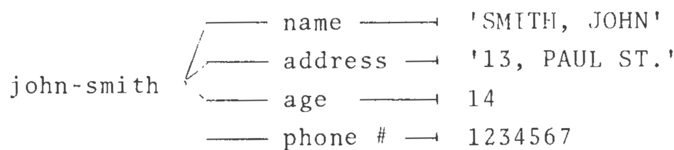
graphical notation to specify properties:

subject $\xrightarrow{\text{attribute}}$ value

For example, a class such as PERSON can have properties such as name, address, age and phone #.



Note that the values of the properties name, address, age and phone # are themselves classes. A token (instance) of PERSON now must have the four properties defined for PERSON.



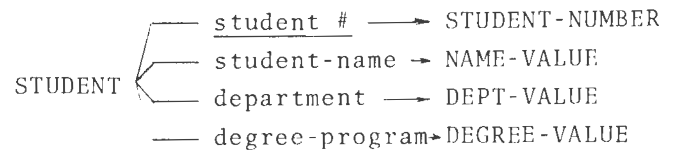
Note that the values of john-smith's properties are (and must be) tokens of the value classes of PERSON.

This example illustrates a very important difference between properties of a class and the corresponding properties of its tokens. In the former, the properties provide information about the structure of its instances, while the latter specify facts about a particular instance. We call the properties of classes <u>definitional</u> and those of tokens <u>factual</u>; <u>notationally</u>, definitional properties will have arrow links while factual ones will have flat arrow links.

TAXIS has seven "types" of classes, each of which can only have certain "types" of properties. Below, we describe three of these class types in detail: Relation, Transaction and Exception.

2.2 Relation

A <u>Relation</u> class resembles a database relation (à la Codd) and has tuples as tokens. It can have three types of properties: key, r-attributes and operations. For example, we might choose to model students as a Relation class, called STUDENT. Suppose STUDENT has r-attributes student #, student-name, department and degree-program. The key property of a Relation class is a subset of r-attributes that uniquely identifies all tokens in the class, for example, student # in STUDENT. Operation properties are transactions that can be applied to tokens of the class. Every Relation class has available three sets of relational operators to manipulate tuples, sequences of tuples and entire relations; these operators are predefined as operation properties of every Relation class. Additional operation properties can be defined by users (an example appears later). For Relation class STUDENT, then, we have



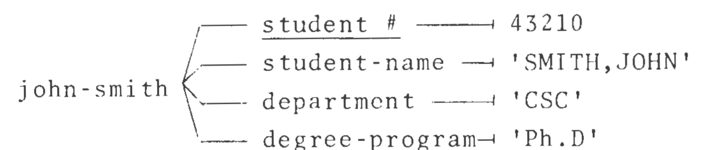
The classes STUDENT-NUMBER, NAME-VALUE, DEPT-VALUE and DEGREE-VALUE are all of type <u>Domain</u>. A Domain class is a collection of atomic tokens with some associated operations. For example, INTEGER is a Domain class, whose tokens are all the integers and whose operations are the usual arithmetic operations. The underlined r-attribute, student # is selected as the key property of STUDENT.

The syntactic program representation of STUDENT and its properties are:

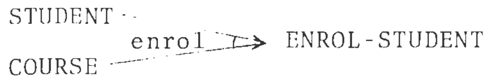
```
relation STUDENT with
   key: student #: STUDENT-NUMBER
   r-attributes
      student #: STUDENT-NUMBER;
      student-name: NAME-VALUE;
      department: DEPT-VALUE;
      degree-program: DEGREE-VALUE;
end
```

For conciseness, we will not present the syntax for classes and properties, but will use the graph notation throughout.

A particular token of STUDENT is an instance of the Relation class with factual properties corresponding to the definitional properties.



As described so far, properties can only be binary relationship. TAXIS actually permits properties to have more than one subject. Such properties are called <u>complex</u>. For example, suppose that in addition to STUDENT, we define a Relation class called COURSE. To specify the property "students can be enrolled in courses through the transaction ENROL-STUDENT", we may associate it with the class STUDENT and/or with the class COURSE. That is, ENROL-STUDENT is no less a property of STUDENT than of COURSE. We use the term <u>complex property</u> to denote a property with multiple subjects (in this case, 2).

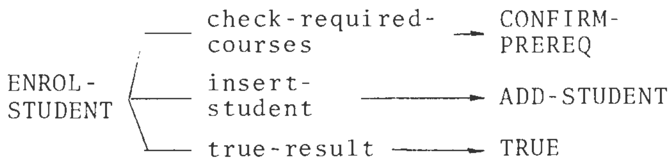


The property enrol is of type "operation". Operation properties associate operations with the data in the database very much in the style of abstract data types. However, the mechanism of complex properties does not require the objects to be partitioned (often artificially) into independent abstract types, a notable disadvantage of languages such as CLU [Liskov and Zilles 74] and ALPHARD [Wulf et al. 76].

2.3 Transaction

Transaction classes are essentially procedures. The tokens of a transaction class are invocations of the transaction, like procedure activations.

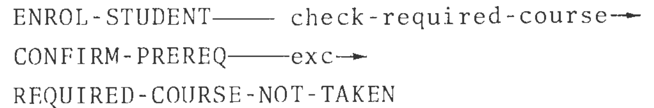
There are three main types of properties that make up the body of a transaction: prerequisite, action and result. In addition, a transaction has as properties the usual components of a procedure such as parameters, local variables and returned values. Prerequisites, action and result properties take as values Expression classes. Standard expression constructs much like EL1 [Wegbreit 74] are provided; among them are repeat-loop, if-then-else, exit constructs and boolean expressions. A transaction is executed as follows: first, all prerequisites are checked; if all are satisfied, then the actions are executed; finally, all results are tested and if they are satisfied, then the transaction returns. For example, the ENROL-STUDENT transaction above might have a prerequisite check-required-courses to check if the student has taken all the required courses of the course s/he wants to take. The action is to incorporate the enrollment into the database, and the result (say) always returns true (no check).



CONFIRM-PREREQ, ADD-STUDENT and TRUE are Expression classes which specify the codes of the prerequisite, action and result properties of ENROL-STUDENT.

2.4 Exceptions

An Exception class in an abnormal condition that can be detected during the execution of a transaction. The only type of properties an Exception class can have is status properties which describe the state of the system when the exception is raised. For example, an Exception class called REQUIRED-COURSE-NOT-TAKEN can be defined for the situation where a student did not take all the required prerequisite courses of a course; the status properties of this class are (say) the student, the course s/he wants to take and the course s/he did not take but was required to take. A token of an exception class is created when a prerequisite or a result of a transaction returns an object other than true. For example, if the prerequisite check-required-courses of transaction ENROL-STUDENT has failed, then an exception token of REQUIRED-COURSE-NOT-TAKEN is raised.

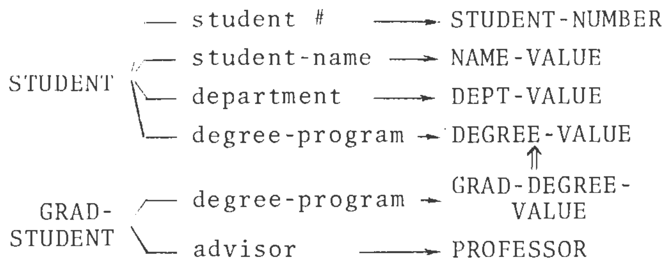


The responsibility of handling the exception raised belongs to the invoker of the transaction that discovers the exception. In the example, exception REQUIRED-COURSE-NOT-TAKEN is handled by the caller of ENROL-STUDENT (another transaction or a secretary). Exception handlers are transactions that decide what to do with the exception (ABORT is the default).

2.5 The ISA Hierarchy

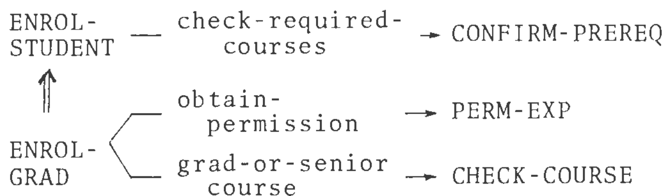
The main facility for organizing classes is the ISA hierarchy. ISA is a binary relation that captures similarities among classes. The statement, X ISA Y (sometimes written $x \leq y$) means every instances of class X is also an instance of class Y. For example, suppose we define a class GRAD-STUDENT ≤ STUDENT. The important characteristic of ISA is its rules for inheriting properties: GRAD-STUDENT ≤ STUDENT implies GRAD-STUDENT has the same five properties as STUDENT. If desired, we can re-define some of the properties of STUDENT for GRAD-STUDENT. For example, since graduate students only receive graduate degrees, we might re-define the degree-program of GRAD-STUDENT to be GRAD-DEGREE-VALUE. This kind of re-definition is restricted by the rule that the new property value must be ISA related to the original value. In our example, it must be that GRAD-DEGREE-VALUE ≤ DEGREE-VALUE.

Additional properties can be defined for GRAD-STUDENT, that are not defined at all for STUDENT. For example, GRAD-STUDENT can have an additional property advisor, whose value is PROFESSOR.



In our notation, we use ⇑ to represent the ISA relationship. If A ≤ B, then we say that A is a specialization of B and B is a generalization of A. The restriction on redefinition of properties of specializations is the same for all class types.

The ISA hierarchy provides the structure for the organization of a TAXIS program. Suppose we want to organize ENROL transactions. We begin, say, with the general transaction, ENROL-STUDENT, which enrols a student in a course. We might then produce a specialization of ENROL-STUDENT, ENROL-GRAD, which enrols a graduate student in a course. Since ENROL-GRAD ≤ ENROL-STUDENT, the prerequisite, action and result properties of ENROL-STUDENT are inherited by ENROL-GRAD. As in the case of Relation classes, properties of transactions can be redefined and/or augmented in specializations. An example is that ENROL-GRAD can have an additional prerequisite to check that the student is only taking graduate level or senior level courses. Also, assume that a graduate student must obtain permission from his/her supervisor:



The prerequisite check-required-courses is augmented in ENROL-GRAD by two more prerequisites.

The ISA relationship between Transaction classes (and Expression classes) is defined by the language (not by the programmer, as in other class types); essentially, if T and T' are Transaction (Expressions) classes, then T ≤ T' if T has "at least as many side-effects" as T'. The ISA relationship between Transaction (and Expression) classes are very interesting and we have only partial solution to the problem of defining it.

Exception classes can also be related through ISA. For example, we might have REQUIRED-COURSE-NOT-TAKEN ≤ ENROL-FAILURE. An exception handler can handle an instance of REQUIRED-COURSE-NOT-TAKEN or it can "generalize" the exception and treat it as an instance of ENROL-FAILURE, in which case, a more 'general' repair action is attempted. Hence, organizing exception classes into a hierarchy gives the designer the flexibility of handling an exception at different level of abstraction.

2.6 Other Features of TAXIS

Space limitations prevent us from describing the use of all of the classes and their properties. Below we list a few more features.

In addition to classes and tokens, we define a third level of description, called metaclasses whose instances are classes. So, for example, Relation is a metaclass whose instances are all the relation classes. Commands are defined for traversing the system at the level of metaclasses. An IIS design can use these commands to explore the definitional properties of a class and the ISA hierarchy of classes. For example, SUB[C] returns a sequence whose elements are all the immediate specializations of class C. PROP[C] returns a sequence whose elements are all the attributes of properties with subject class C. TAXIS also allows the existence of variables which take property attributes as values and can therefore be used along with PROP to explore a TAXIS model in run-time.

The reader is referred to [Mylopoulos et al. 78a] for a complete description of TAXIS and an extended example of an IIS. A condensed version of this document will appear in [Mylopoulos et al. 78b].

3. Semantic Networks and TAXIS

TAXIS treats an IIS as a collection of classes interrelated through properties (binary relationships) and organized into an ISA hierarchy. In this sense, TAXIS is nothing but a "toned-down" language for the creation, modification and search of semantic networks. Because of its specialized application area (design of IISs) TAXIS is in several ways different from a general-purpose semantic network formalisms. Below we list some of the most important differences between TAXIS and a particular semantic network formalism (hereafter referred to as SNF) described in [Levesque 77], [Levesque and Mylopoulos 78], which served as the starting point for the design of TAXIS.

(a) In the SNF a semantic network is allowed to have an arbitrary number of description levels. In other words, a semantic network may include an object A_1 which is an instance of an object A_2 which, in turn, is an instance of an object A_3 etc. In TAXIS we have restricted the

number of levels of description to three (tokens-classes-metaclasses). This was done partly in order to enhance the efficiency of IISs designed in TAXIS and partly because it was felt that it is rarely useful to have more than three levels.

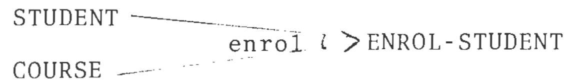
(b) SNF distinguishes two types of definitional properties: structural and assertional ones. Structural properties describe the internal structure of a class, e.g. the r-attributes of PERSON mentioned earlier would all be treated as structural properties. Because of their use, structural properties do not have their own semantics (i.e. are not classes with their own properties and position on the ISA hierarchy). Assertional properties, on the other hand, define binary relationships between classes but are themselves treated as classes (e.g. PARENT-OF could be treated as a binary relationship with PERSON as domain and co-domain). In TAXIS all definitional properties are structural, and assertional properties can only be defined indirectly through relation classes.

(c) New classes can be created dynamically in SNF. This is not allowed in TAXIS where the collection of classes, metaclasses, their properties and their ISA hierarchy are all fixed at run-time. This means that users of the IIS cannot modify its basic structure. If the designer of the IIS wants to modify it, s/he can do so only by augmenting the definitions of new classes to those that already exist and then recompiling the entire system. Of course, this restriction to TAXIS was introduced with run-time efficiency and conceptual simplicity in mind.

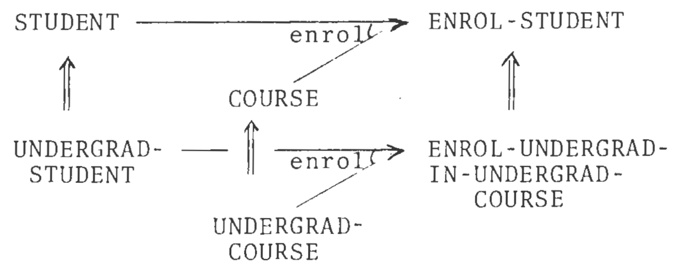
(d) SNF treats expression classes like all other classes by defining an internal structure for them. In TAXIS every expression class is treated as atomic and its semantics are determined by the expression associated with that class. This restriction simplifies the conceptual framework of TAXIS as well as its implementation.

(e) In TAXIS, unlike SNF, there exist different types of classes (e.g. domain, relation, transaction, exception) and properties (e.g. key, r-attribute, operation). These types were introduced primarily in order to aid the designer of an IIS describe his system in TAXIS. Thus, the designer can take it for granted that s/he will use domain, relation, transaction and exception classes which have particular features that make them suitable for representing different aspects of an IIS. If SNF was used instead of TAXIS, the concepts of domain, relation, transaction and exception class would probably have to be defined before proceeding to the details of the design. SNF is a "general purpose" representation language whereas TAXIS is a "special purpose" one.

(f) A complex property allows one to relate more than one subject to a common property. This adds expressive power to other semantic networks where only binary relations are allowed. More importantly, ISA hierarchies of a complex property's subject classes can be used to induce an ISA hierarchy of the property's value class. In a STUDENT-ENROL-COURSE model, we have an operation property "enrol" with two subjects, its value being a transaction ENROL-STUDENT.



One of the prerequisites of ENROL-STUDENT is to check that the student has taken all the required prerequisite courses s/he wants to take. Suppose we now specialize STUDENT and COURSE by adding subclasses UNDERGRAD-STUDENT and UNDERGRAD-COURSE. By the inheritance rule, the property enrol should also be a property of UNDERGRAD-STUDENT and UNDERGRAD-COURSE. Now suppose that an undergraduate student cannot take more than 5 undergraduate courses. In TAXIS, such special cases can be modelled by adding an additional prerequisite to the property value (a transaction) associated to the particular subclasses in the ISA hierarchies of the subjects. In our example, the ENROL-STUDENT transaction associated with UNDERGRAD-STUDENT and UNDERGRAD-COURSE is supplemented by an additional prerequisite ("a student cannot take more than 5 courses"). That is, a new transaction (call it ENROL-UNDERGRAD-IN-UNDERGRAD-COURSE) is created which is the same as ENROL-STUDENT but with an additional prerequisite.



In general, given a complex property among N subject classes associated with a property value class V, an ISA hierarchy of V is induced whose appearance reflects the cross product of the ISA hierarchies of the N subject classes. This induced hierarchy of V can be constructed quite easily, because at any one time, one need only look at a certain combination of the subclasses of the subjects and specialize the inherited V class by adding and/or redefining the associated properties. Using these concepts, TAXIS provides structuring mechanisms that control the explosion of detailed specification of an application, because, at any one time, a special case

always has a unique proper spot on the induced ISA hierarchy.

(g) TAXIS offers exception-handling facilities based on control structures described in [Wasserman 77] which have been integrated into the framework of classes, properties and the ISA relationship.

The concept of exception organization and handling can make the operations of semantic networks more robust and provide a way of ensuring reliability of the knowledge base. Organizing exceptions in ISA hierarchies allows one to deal with errors in a flexible manner. For example, suppose we have the following ISA hierarchy of exceptions.

REQUIRED-COURSE-NOT-TAKEN ≤
ENROL-FAILURE ≤ INTEGRITY-VIOLATION

The amount of exception handling in TAXIS can vary from none at all to every possible kind of exception. In the former case, every exception is generalized to an instance of INTEGRITY-VIOLATION; the system default (e.g. ABORT) is used uniformly. In the latter case, a handler is associated with every exception class and handling of these exceptions can presumably be very different from one to another. To the designer of IISs, this organized way of exception handling is obviously useful, since in many large applications, as much as 30% or more of the actual code is dedicated to their detection and handling.

(h) SNF factors programs into a single prerequisite, a single action and a single result. TAXIS extends this idea by treating prerequisite, action and result as property types and therefore allowing for programs with several prerequisite, action and result properties. This capability helps the programmer factor out a transaction into as many units as s/he sees fit, with each unit being a property of the transaction. Once defined, these units can be inherited and/or refined by classes lower down on the ISA hierarchy of transactions.

4. Conclusions

A programming language offers its user a point of view in helping him/her formulate his/her ideas into a coherent program. TAXIS offers a view based on the idea that an IIS is a knowledge-based system, not just a program. This view is realized by the framework of classes, properties and the ISA relationship in terms of which all TAXIS constructs are described. We believe that the result is a programming language particularly helpful to its users in the design, debugging and maintenance phases of their projects.

Because TAXIS is designed with reliable IISs in mind, we are forced to be quite careful and relatively formal in defining it. We believe it is possible to formalize the set of constructs in TAXIS to the point of proving "correctness" of a model (in the spirit of [Hoare 69]); we have started a project on formalization of TAXIS, see [Wong 78]. In the future, we may be able to define the notions of "correctness" of semantic networks quite rigorously.

TAXIS is defined as an applications programming language, hence its compilability (without run-time interpretation of a semantic network) is an important goal. A similar goal may be applied to other semantic network formalisms. These semantic network 'compilers' should be much more efficient than the intrinsically slow interpretive way of using semantic networks - the traditional approach. While this compilability property has the advantage of efficient execution, it also has the drawback of limiting the ability to create new classes 'on the fly'. This compilability requirement has forced us to consider only those applications areas where there is no need to create new classes in run-time. IIS design is such an application area. TAXIS is aimed at one particular application area (here we don't mean a particular 'microworld' such as Blocksworld, but application types such as IISs). Hence this allows us to pick useful classes and constructs to model such an area. The use of a semantic network formalism for the specification of particular kinds of knowledge bases has enabled us to refine such formalisms by introducing particular types of classes which can only have particular types of properties. This specialization process allows us to design a small language (i.e., with a small set of constructs), at the expense of expressibility (i.e., lack of generality). However, this appears to be an important step in using representations of knowledge in particular application areas. We feel that there is a lot to be learned from such applications. They provide challenges to a representation formalism that lead to refinements and enrichments of the formalism itself. As we have shown, these enrichments often are as valuable to the general knowledge representation research community as they are to the application area that spawns them.

Acknowledgements

We would like to thank the following people for their constructive comments: Robin Cohen, Hector Levesque, Gord McCalla and Ray Perrault.

We also wish to thank Teresa Miao for her excellent typing.

References

[Hoare 69]
Hoare, C.A.R., "An Axiomatic Basis for Computer Programming", CACM 12:10, 1969.

[Levesque 77]
Levesque, H., "A Procedural Approach to Semantic Network", TR-105, Dept. of Computer Science, Univ. of Toronto, 1977.

[Levesque & Mylopoulos 78]
Levesque, H., Mylopoulos, J., "A Procedural Semantics for Semantic Networks", to appear in Associative Networks - the Representation and Use of Knowledge in Computers, Academic Press, edited by N.V. Findler, 1978.

[Liskov & Zilles 74]
Liskov, B., Zilles, S., "Programming with Abstract Data Types", SIGPLAN Notices, 9, 4, 1974.

[Mylopoulos et al. 76]
Mylopoulos, J., Borgida, A., Cohen, P., Roussopoulos, N., Tsotsos, J., Wong, H., "TORUS: A Step Toward Bridging the Gap Between Databases and the Casual User", Information Systems, vol.2, no.2, 1976.

[Mylopoulos et al. 78a]
Mylopoulos, J., Bernstein, P.A., Wong, H.K.T., "A Preliminary Specification of TAXIS: A Language for Designing Interactive Information Systems", TR-78-02, Computer Corp. of America, Jan. 1978.

[Mylopoulos et al. 78b]
Mylopoulos, J., Bernstein, P.A., Wong, H.K.T., "A Language Facility for Designing Interactive Database - Intensive Applications", to appear in TODS, presented in ACM-SIGMOD 78 Conference, Austin, Texas, May 1978.

[Roussopoulos 76]
Roussopoulos, N., "A Semantic Network Model of Databases", TR-104, Dept. of Computer Science, Univ. of Toronto, 1976.

[Smith & Smith 77]
Smith, J.M. Smith, D.C.P., "Data Base Abstractions: Aggregation and Generalization", ACM TODS, vol.2, 1977.

[Wasserman 77]
Wasserman, A.I., "Procedure-Oriented Exception Handling", TR #27, Lab. of Medical Information Science, Univ. of California, San Francisco, 1977.

[Wegbreit 74]
Wegbreit, B., "The Treatment of Data Types in EL1", CACM, vol.17, no.5, 1974.

[Wong & Mylopoulos 77]
Wong, H.K.T., Mylopoulos, J., "Two Views of Data Semantics: A Survey of Data Models of Database Management and Artificial Intelligence", INFOR, vol.15, no.3, Oct. 1977.

[Wong 78]
Wong, H.K.T., Ph.D. thesis, to appear.

[Wulf et al. 76]
Wulf, W., London, R., Shaw, M., "Abstraction and Verification in Alphard: Introduction to Language and Methodology", Tech. Report, Dept. of Computer Science, Carnegie-Mellon University, 1976.

Organization of Knowledge for a Procedural Semantic Network Formalism

Peter F. Schneider
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada

1.0 ABSTRACT

An investigation of some of the issues involved in the organization of knowledge in semantic networks is presented. The investigation is in terms of a procedural semantic network formalism developed by H. Levesque [Levesque 77a] although the ideas put forward have application in most other semantic network formalisms. The main ideas include the generalization of the ISA hierarchy, inheritance in this hierarchy, the treatment of programs as examinable objects that can participate in an ISA hierarchy, and the retention of consistency in the definition of classes.

2.0 INTRODUCTION

Representation of knowledge is a major problem in many areas of Artificial Intelligence. Based upon research by the author in [Schneider 78], this paper investigates certain problems of a particular representation of knowledge, the procedural semantic network (PSN) formalism developed by H. Levesque [Levesque 77a] [Levesque and Mylopoulos 78]. The problems generally involve inadequacies of the tools for organizing knowledge (ISA hierarchies, structures, etc.) in the PSN formalism. They include the inability of ISA hierarchies to form acyclic graphs instead of just trees, the problem of inheritance in such extended hierarchies, the inability to ensure that classes and relations are consistent, the inability to assign an object to a class depending upon its attributes, and the inability to inherit fragments of programs along the ISA hierarchy.

These problems are at least partially overcome through modifications and extensions to the PSN formalism which together produce a new formalism called the extended PSN (EPSN) formalism. These modifications and extensions are, in general, designed to increase the power of the EPSN formalism to organize knowledge into hierarchies, classes, and structures. They further retain the desirable properties of the PSN formalism including consistency of the formalism, flexibility, extensibility, self-examinability, modifiability, and modularity.

Although the notions in the EPSN formalism are based specifically on the PSN formalism, most of them are also relevant to any semantic network formalism and, in fact, to representation languages such as KRL [Winograd and Bobrow 76] and FRL [Roberts and Goldstein 77].

3.0 THE PSN FORMALISM

The PSN formalism is an approach to the representation of knowledge that uses a procedural framework to define the semantics of semantic network based systems. That is, programs define actions in the formalism and represent aspects of the domain being modelled. What follows is a brief discussion of the PSN formalism. More detail can be found in [Levesque 77a] or [Levesque and Mylopoulos 78].

All entities in the PSN formalism are called objects. There are <u>definite objects</u>, akin to constants; <u>indefinite objects</u>, akin to variables; and <u>assertions</u>, used in relations. There are four important subsets of definite objects, namely <u>classes</u>, <u>relations</u>, <u>programs</u>, and <u>contexts</u>.

A class is a collection of definite objects sharing common properties. The objects in the class are the class's <u>instances</u> and may themselves be classes. The semantics of a class are defined by four programs attached to the class. The class's <u>instantiator</u> program makes an object an instance of the class. Its <u>terminator</u> removes an object from the class. Its <u>generator</u> generates all the instances of the class. Finally, the class's <u>recognizer</u> checks whether an object is an instance of the class.

A relation is a mapping from one class, its domain, to another, its range. Relations have assertions which form the extension of the relation. There are four programs attached to relations, called the asserter, eraser, accessor, and tester, which perform roles analogous to the four programs attached to classes.

A program is a special kind of class whose instances (called processes) are program activations and can be executed in the formalism. These programs and processes are used to define the semantics of classes and relations. A context is a special type of relation which is used to define the visibility of objects and to associate indefinite objects with their values.

A class may define structural attributes for its instances (e.g. students have an age) and then instances of this class have a value for this structural attribute (e.g. John is 25 years old). Structural attributes and their values form the major part of the definition of objects. A class may also define assertional attributes for its instances (e.g. students take courses) and then instances of this class may have values for this assertional attribute (e.g. John takes CSC374F and CSC334S). Assertional attribute values are assertions between the object and other objects. They do not form part of the definition of objects but are used for incidental properties.

In the PSN formalism every object is an instance of some class or relation. Those classes whose instances are classes or relations are called metaclasses. Metaclasses are very important because they define the attributes of their instances (namely the classes and relations) which, in turn, define the attributes of all objects. The basic metaclasses are "CLASS", "RELATION", "PROGRAM", and "CONTEXT" whose instances are the classes, relations, programs, and contexts of the formalism respectively. The attributes defined in metaclasses are given values in classes or relations, thus allowing classes to both have attribute values (e.g. average age of students) and define attributes for their instances (e.g. a student's age).

Classes and relations participate in an ISA hierarchy which forms a tree. An instance (assertion) of a class (relation) is also an instance (assertion) of that class's (relation's) ISA ancestors. Attribute definitions of both kinds are inherited by ISA descendents because of the subset property of the ISA hierarchy mentioned above. These definitions may be modified in ways that restrict the possibilities allowed for instances of the subclass.

The structural attribute values of a class help define the class. Thus it makes sense for its subclasses to inherit these values. However, to allow for flexibility in the PSN formalism these inherited values may be overridden by the subclass. On the other hand assertional attribute values are not part of the class's or relation's definition and thus there is no reason to inherit them along the ISA hierarchy.

Thus the ISA hierarchy serves as an abstraction mechanism, where ISA children are specializations of their ISA parents. Abstraction is also provided by the structural attributes where the details of objects are hidden in their structural attribute values.

Programs form a large part of the PSN formalism since they are used to define the semantics of classes and relations. Programs are classes and thus they can define structural attributes for their instances (processes). These attributes are used as program variables by the processes. There are four structural attributes defined in "PROGRAM" which are used to define programs. These all take values which are program forms and are defined in the formalism. First, there is the prerequisite which is a logical expression which is executed when a process of the program is created and must evaluate to "true" or the process will not be executed and the call which created it will fail. Second, there is the body which constitutes the main actions or calculations executed by processes of the procedure. Third, there is the effect which contains actions to be executed if the execution of the body does not fail. Fourth, there is the complaint which contains actions or remedies to be taken if the body fails.

Since programs are classes they participate in the ISA hierarchy. This means that programs inherit the four structural attributes defined in "PROGRAM" from their ISA parents. This aids in constructing new programs especially for use as programs attached to a class or relation. Often the programs attached to a subclass differ only in one part from those programs attached to its ISA parent and making them subprograms of the parent's programs allows these changes to be easily made.

Programs are divided into three basic types. These are procedures, whose bodies perform actions such as creating or destroying objects; functions, whose bodies return objects or sequences of objects as value; and predicates, whose bodies return "true", "false", or "unknown" as value. These three types are set up by having three subclasses of "PROGRAM", namely "PROCEDURE", "FUNCTION", and "PREDICATE" which contain the

necessary modifications to the definitions in "PROGRAM".

Now the four programs attached to classes and relations are just structural attributes defined in "CLASS" and "RELATION" respectively. Instantiators, terminators, asserters, and erasers are all procedures; generators and accessors are functions; and recognizers and testers are predicates. This means that the semantics of classes and relations (except for a few basic programs) can be defined, accessed, and modified in the formalism.

Contexts form another large portion of the PSN formalism. A context is a special type of relation which is used to associate indefinite objects with their values. As mentioned above, contexts are also used to create a context hierarchy (as in [Sussman and McDermott 72] or [Hendrix 75]) which controls the visibility of objects and actions. If an object is created or any other action is performed this is only visible or effective in the context in which it was performed and its descendents in the context hierarchy.

A context can give values to those indefinite objects which are visible in the context which it interprets, called its structure. Furthermore, contexts have a default which is another context and objects which can be given values in the context inherit values from this default context. The structural attributes of a class use this mechanism in the following way: if "C" is a class then it will have an attached context, called its part context, where the objects which form its structural attribute definitions are created. If "C" is a subclass of another class, "B", then its part context will be a child of "B"'s part context in the context hierarchy so the objects defined in "B"'s part context are visible in "C"'s part context.

Then an instance of "C", say "c", will have a local context whose structure is "C"'s part context. The structural attribute values for "c" are just the values in its local context. The default for "c"'s local context is "C"'s part context so "c" inherits defaults from "C". Further, if "c" was a class or relation and had a subclass or subrelation "d", then "d" would have a local context whose structure would be "C"'s structure and whose default would be "c"'s local context so that "d" would inherit "c"'s structural attribute values. This is how inheritance along the ISA hierarchy actually works in the formalism.

In summary, the PSN formalism is an object centred formalism based upon semantic networks which uses procedural attachment to define the semantics of classes or relations. Since each class or relation is ultimately completely responsible for its own semantics, this leads to an ACTOR [Hewitt 73] style representation tempered by an inheritance mechanism which makes it easier to construct new objects and share information.

4.0 THE EPSN FORMALISM

4.1 Hierarchies And Inheritance

The EPSN formalism extends the PSN formalism in a number of places, particularly by allowing the ISA hierarchy to be a directed acyclic graph. Thus a class in the EPSN formalism can be an ISA child of two or more other, unrelated classes.

The inheritance of structural attributes along the ISA hierarchy is a major part of the PSN formalism and the generalization of the ISA hierarchy produces several problems with respect to this inheritance. The definition of a structural attribute for a class is the definition of an indefinite object (which is akin to a variable) in a special context associated with the class. As shown in Figure 1 (where "sex" is a structural attribute definition (SAD) for

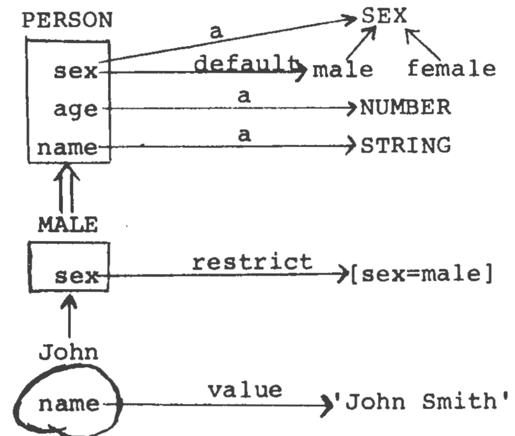


Figure 1*

----

* In the figures INSTANCE-OF relationships are represented by single, unlabelled arrows; ISA relationships are represented by double, unlabelled arrows; and other relationships or links are represented by single, labelled arrows. Also rectangular boxes enclose structural attribute definitions of classes or objects defined in a structure and irregular boxes enclose the structural attributes of objects and objects used in valuors.

"PERSON") such objects have an identifier, (here "sex"); have an associated class (here "SEX") which defines the values the attribute may have in instances of the class; may have a default (here "male") which is used in instances if no value is given; and may have one or more restrictions (as in the class "MALE" of Figure 1) which further restrict the possible values in instances. It is possible for ISA children (for example "MALE") to modify the associated class or default and to add new restrictions.

Now suppose a class is an ISA child of two other classes (as "CHILD-TAXPAYER" in Figure 2 is an ISA child of both "DEPENDENT-TAXPAYER" and "CHILD"). This class should inherit SADs from both of its ISA parents without favouring either. This works out fine for SADs present in only one of the parents (such as "net-income" which is present only in "DEPENDENT-TAXPAYER"). In these classes this SAD is inherited as is and may be modified as outlined above.

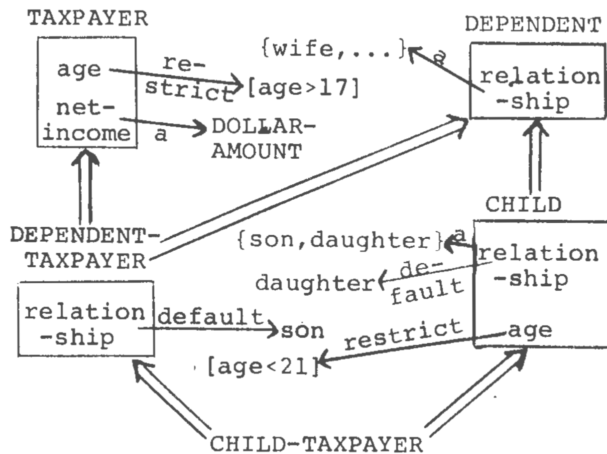


Figure 2

However, if the parent classes have SADs that have the same identifier but are not identical then problems arise. The child class must inherit a SAD which is some combination of the SADs in the parents. For SADs with differing restrictions this combination is easy since it is natural to just further restrict the possible values for the SAD of the ISA child by accumulating the restrictions. Thus the SAD "age" for "CHILD-TAXPAYER" has two restrictions; one restricting the age to be less than 21 and one restricting it to be greater than 17. As above, the ISA child may add new restrictions.

For SADs with differing associated classes much the same sort of solution is required. For example, the SAD "relationship" of "CHILD-TAXPAYER" should have the associated class "{son,daughter}" since this is the intersection of the associated classes ({son,daughter} and {wife,husband,son,daughter}) in its ISA parents. However, most classes are not sets (i.e. cannot be described by simply listing their extension) and in these cases some construct similar to intersection must be used.

For this and other purposes the EPSN formalism defines the meet of a set of objects from an ISA hierarchy as that object which is an ISA descendent of all of the objects in the set and is an ISA ancestor of all other objects with this property. Such classes are the greatest (in the partially ordered set defined by treating the ISA links as a cover relation) class less than the classes being combined and thus are the ISA hierarchy equivalent of meets in a lattice. Since the meet is an ISA descendent of the classes being combined it inherits attributes from all of them. However, this does not mean that a meet automatically has as instances the intersection of the instances of the classes that form the meet.

In the EPSN formalism meets are given names that consist of the names of the classes forming the meet concatenated together with '&'s separating them. Thus in Figure 3 "DEPENDENT&TAXPAYER" is the meet of "DEPENDENT" and "TAXPAYER" and "CHILD&TAXPAYER" is the meet of "CHILD" and "TAXPAYER". It is not necessary to set up these meet classes beforehand; they are created, if necessary, when referenced by their name or needed for the inheritance rules. This significantly reduces the work required to create large and deep ISA hierarchies.

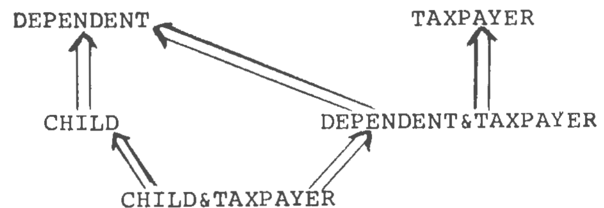


Figure 3

Thus the associated class of an ISA child's SAD is the meet of the associated classes of the SADs in its ISA parents. Of course, the ISA child may modify the associated class to be an ISA descendent of the inherited associated class thus

further restricting its possible values.

The situation for defaults is a bit different. Suppose the defaults in the ISA parents are objects that do not participate in an ISA hierarchy (i.e. are neither classes nor relations). In this case there is no reasonable way of producing a compromise default if the defaults in the parents are not identical. It seems that the only solution is not to inherit any default at all. Thus, in Figure 2, there is no default for "relationship" in "CHILD-TAXPAYER" since the defaults in its ISA parents are "son" (in "DEPENDENT-TAXPAYER") and "daughter" (in "CHILD") which are non-identical and do not participate in an ISA hierarchy. If a default is desired then a default can be given in the ISA child (e.g. "son" could be used as the default in "CHILD-TAXPAYER").

If the defaults in the ISA parents do themselves participate in the ISA hierarchy then there is a logical default to be inherited by the ISA child. This is the meet of the defaults in the parents. This makes sense since the meet is analogous to the intersection of the defaults and follows the general idea of restricting the SADs in ISA children. If this is not what was desired then it can be changed to any other valid value.

So far the discussion has centred on inheritance of SADs by subclasses of classes. Instances of classes can also inherit information; in particular, an instance of a class inherits the SADs of the class as structural attributes which can be given values (SAVs). If no value is given then the default of the SAD is used as a value and if there is no default then "unknown" is used. For example, "John" in Figure 1 has SAV 'John Smith' for his structural attribute "name", inherits the default "male" for his "sex", and has an "unknown" "age". It must be noted that before an object can have a SAV it must inherit the corresponding structural attribute by virtue of being an instance of a class that has a corresponding SAD.

Now the instances of a class may themselves be classes or relations and participate in the ISA hierarchy. Since the SAVs of an object constitute the major portion of its definition they should be inherited along the ISA hierarchy. This inheritance can become very complicated when the ISA hierarchy becomes complicated. For example, consider Figure 4 where "RELATION", that class in the EPSN formalism whose instances are relations is partially defined. The instances of "RELATION" have a "tester" structural attribute whose value is a predicate which determines if two objects are related by the relation. "RELATION" gives a standard default, "stdtester", for this structural attribute. Also shown in Figure 4 is "SYMMETRIC-RELATION", an ISA child of "RELATION", whose instances are supposed to be symmetric relations. "SYMMETRIC-RELATION" accomplishes this by changing the default for "tester" to "symtester" which contains the testing appropriate for symmetric relations. "RELATION" also defines the structural attributes "domain" and "range" which are the domain and range of relations.

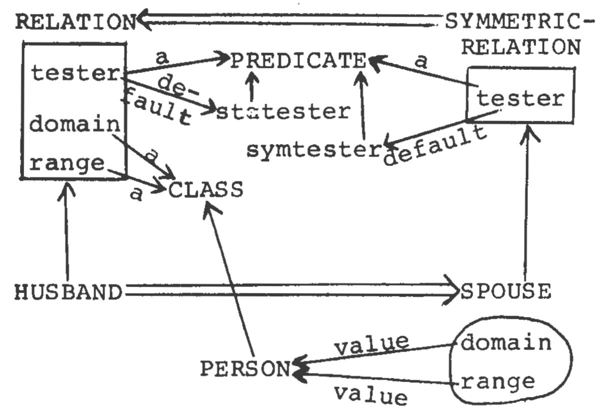


Figure 4

This looks fairly straightforward; instances of "RELATION" will be relations and instances of "SYMMETRIC-RELATION" will be symmetric relations. However, consider the situation for the relations "SPOUSE" and "HUSBAND" as shown in Figure 4. "SPOUSE" is a symmetric relation and "HUSBAND" is a non-symmetric sub-relation of "SPOUSE". Since "HUSBAND" is a sub-relation of "SPOUSE" it should inherit "SPOUSE"'s "domain" and "range". However, "HUSBAND" is not a symmetric relation so it should not inherit "SPOUSE"'s "tester" but should instead inherit "stdtester" from "RELATION". The problem is to devise an inheritance scheme which will perform correctly in this and other complicated cases.

The EPSN formalism solves this problem by using two SAVs. The inheritable SAV is derived in the same manner as the defaults of SADs. It is specified by the "value" links in the figures, is inherited along the ISA hierarchy in exactly the same way as the defaults for SADs, and can be modified by "value" links in the ISA descendents in the same fashion. Thus "SPOUSE" has no inheritable SAV for "tester" and thus "HUSBAND" also has no inheritable SAV for "tester" whereas "SPOUSE" has the inheritable SAV "PERSON" for both "domain" and "range" and these are inherited by "HUSBAND" so that it has the correct domain and range.

The actual SAV is the SAV that is used in the object. If there is no inheritable SAV (as determined above) in an object for one of its structural attributes then the actual SAV in that object becomes the default from the class of which the object is an instance or becomes "unknown" if there is no such default. Thus "SPOUSE" has as actual SAV for "tester" the symmetric tester "symtester" from "SYMMETRIC-RELATION" and thus "HUSBAND" has the regular tester "stdtester" from "RELATION", as required.

If there is an inheritable SAV for the structural attribute and this SAV participates in the ISA hierarchy then the actual SAV is the meet of the inheritable SAV and the default for the structural attribute. This rule allows modifications to be made to "symtester" in "SPOUSE" by putting the changes into its inheritable SAV and then in "HUSBAND" having these changes inherited and applied to "stdtester". This should be possible since "stdtester" and "symtester" should be similar programs. For details on how this could work out see the section on programs.

Finally, if there is an inheritable SAV which does not participate in an ISA hierarchy then this becomes the actual SAV no matter what the default is. This is the case for objects such as numbers or other simple objects.

The idea behind these complicated inheritance rules is not only to solve the problem given above but also to make it possible to construct very complicated ISA hierarchies with a significant amount of interaction. An EPSN implementation could presumably take care of such complex processing automatically, leaving the user with a relatively simple and natural task.

4.2 Structures And Valuors

The EPSN formalism redefines the contexts of the PSN formalism as two separate concepts. In the EPSN formalism <u>structures</u> are used to support the visibility of objects and <u>valuors</u> are used to give values to indefinite objects. For a valuor to be able to give a value to an indefinite object this object must be defined in the structure which is linked to the valuor via an "interpret" link.

For example, "Structure-1" in Figure 5 defines two objects, "age" and "John". Because "Valuor-1" is linked to "Structure-1" via an "interpret" link it can give a value to the indefinite object "age" (and does so, giving it the value "29").

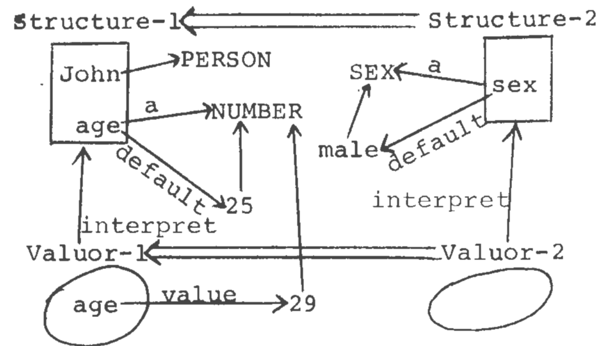


Figure 5

Both valuors and structures participate in an ISA hierarchy. These hierarchies replace the context hierarchy of the PSN formalism, and inheritance of definitions of objects and values for objects along them is performed in exactly the same manner as for SADs and SAVs, respectively. Thus "Valuor-2" in Figure 5, since it interprets a structure which is an ISA descendent of "Structure-1", can give a value to "age". Further, since it as an ISA descendent of "Valuor-1", it inherits the value "29" from "Valuor-1".

In fact, structures and valuors are used to formally implement the inheritance of SADs and SAVs. (So it is actually the case that inheritance of SADs and SAVs is defined in terms of inheritance in structures and valuors and not vice versa.) The main reason for separating the two uses of contexts is to support this inheritance in a cleaner manner. Even though contexts are split into two parts when dealing with classes, if a full PSN context is required it can be created since a context is defined to be a structure plus a valuor that interprets the structure.

4.3 Programs

There are also important extensions and modifications in the EPSN formalism to the programs of the PSN formalism. Programs are basic to both formalisms since they are used to define the semantics of classes and relations. One of the modifications to programs in the EPSN formalism comes as a result of a restriction on indefinite objects. In the EPSN formalism the value of an indefinite object may not be changed <u>in a valuor</u> once it has been given some value in that valuor. Since indefinite objects are used as program variables and processes each have a valuor to hold their values this means that program variables can only be

given one value which may not be later changed. This may seem to be a serious restriction but all that it means is that EPSN programs have to be written in a style closely akin to functional or applicative programming [Tennant 76]. In this style binding variables to values is very closely associated with flow of control. A style similar to that necessitated in the EPSN formalism is advocated in [Levesque 77b].

However, the most significant modification to programs is the change in the method of constructing forms (the constructions which make up the four parts which comprise a program). In the PSN formalism these four forms are each a single block of code which can be only inherited by subprograms without modification. Thus, although the division of programs into four parts allows inheritance of each part separately, to modify one of these parts usually requires that the entire part be rewritten.

In the EPSN formalism a form is a context in which several blocks of code are defined. This results in programs similar to those in [Mylopoulos et al 78]. For example, "FORM-7" in Figure 6 contains three blocks of code identified by "first", "second", and "third". Note that "second" is special in that it is an indefinite object. To execute this form the three blocks of code are executed in the order determined by the "subsequently" links (which need not form a total order).

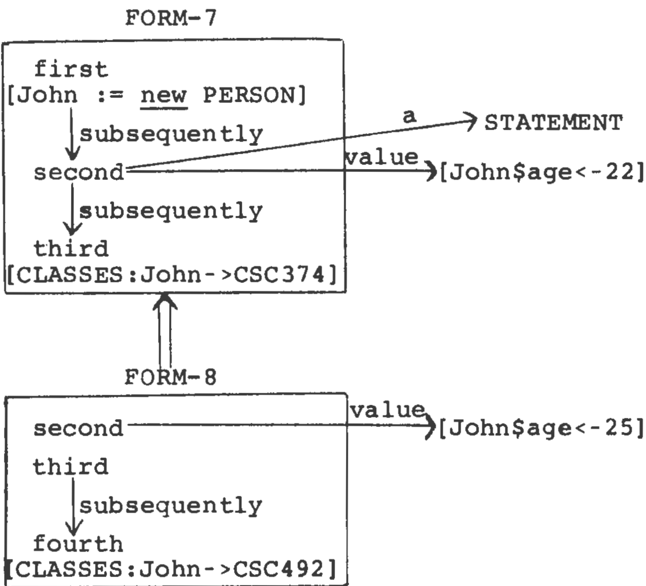


Figure 6

In the case of "FORM-7", which is a statement (i.e. a form that performs actions), this is all that is required to specify how to execute the form. In the case of conjectures (forms that return truth values) the value of the form is the result of 'anding' together the values of the enclosed blocks of code. Again the order of execution, since it is possible for the blocks of code to have side-effects, is specified by "subsequently" links. In the case of expressions (forms that return objects or sequences of objects) the enclosed code blocks each contribute one object to the sequence, the order being determined by the "subsequently" links.

This division allows more complex inheritance of program parts to take place. The inheritance is precisely that described above for inheriting objects defined in contexts and values for them along the ISA hierarchy. Thus "FORM-8" in Figure 6 has four parts. The parts with identifiers "first" and "second" are inherited without change from "FORM-7". The part with identifier "second" is also inherited from "FORM-7" but it is an indefinite object and has been given a new value in "FORM-8" (namely "[John$age<-25]"). The part with identifier "fourth" is created in "FORM-8" and not inherited. "FORM-8" also has three "subsequently" links, two inherited from "FORM-7" and one created in "FORM-8", so that its parts will be executed in the order "first", "second", "third", and "fourth".

This inheritance scheme allows for easy modification and addition to forms and thus allows easy construction of slight modifications of programs. These slightly modified programs are especially useful in constructing the programs that define the semantics of classes and relations.

4.4 Consistency

One of the major problems in the PSN formalism is the possibility of creating a class or relation and designing the four associated programs that define its semantics in such a way that the class or relation is inconsistent. For example, it is possible to write instantiator and recognizer programs for a class in such a way that the recognizer will not recognize objects that were supposedly made instances of the class by the instantiator as instances of the class.

The EPSN formalism partially alleviates this problem by including several metaclasses that define particular types of consistent classes. The two that are the most prominent are stored classes, classes that explicitly store data

allowing them to recognize as instances those objects which have been instantiated into the class or are instances of the class's ISA sons, and intensional classes, classes that have as instances those objects which belong to the class's ISA parent and also satisfy some boolean condition. A class that belongs in either of these types of classes can be constructed very easily and with no chance of creating inconsistencies. A stored class is just made an instance of the metaclass "STORED-CLASS", placed into the ISA hierarchy, and later given instances while an intensional class is made an instance of the metaclass "INTENSIONAL-CLASS", placed into the ISA hierarchy, and given a boolean condition which specifies its instances.

However, many classes do not fall into either of these categories and are therefore not guaranteed consistent in the EPSN formalism. Nonetheless, most of these are modifications of the above categories and thus can be easily constructed with little possibility of creating inconsistencies.

Another possible cause of problems in the semantics of classes and relations occurs when an ISA descendent's associated programs do not correspond with those in the ISA ancestor. For example, if one piece of the instantiator has to be changed then, in the PSN formalism, a large chunk of the code may have to be rewritten and this may introduce an inconsistency. The problem is alleviated in the EPSN formalism (but certainly not eliminated) by allowing such changes in parts of programs to be easily and naturally specified, thus reducing the possibilities of errors and resultant inconsistencies.

5.0 CONCLUSION

The EPSN formalism is an attempt to solve some of the problems in the PSN formalism, primarily to increase its powers in the organization of knowledge. To this extent it appears to be fairly successful. It contains solutions to several problems of the PSN formalism. The ideas present in these solutions can also be used to advantage in the organization of knowledge in other formalisms based upon semantic networks, especially those which attempt to include procedures in the formalism and use them.

The main contribution of the EPSN formalism is its generalization of the ISA hierarchy of the PSN formalism and its specification of inheritance rules for structural attribute definitions and values in this generalized hierarchy. This makes it much easier to create large, complex hierarchies and to create modifications of existing objects in the hierarchy. The idea of creating meets in such hierarchies further aids in the creation of complicated hierarchies by allowing the user to specify only the basic classes in the hierarchy and have the dependent classes available when needed.

Programs have been modified in the EPSN formalism so that a program is composed of four parts (as in the PSN formalism) which are then each composed of several pieces that are executed in a specified order. These pieces are inherited by ISA children of the program and more pieces can be added and certain pieces may be changed in the ISA child. This allows easier manipulation of programs and much easier organization of programs in hierarchies so that the definition of semantics via programs is easier for complex ISA hierarchies than was the case in the PSN formalism. Programs are defined within the formalism and thus the interpreter can be specified almost entirely within the formalism (as in LISP or the PSN formalism).

However, there are some areas that should receive further research. The area of consistency in the definition of the semantics of classes and relations is not solved adequately. Although the EPSN formalism does not allow exceptions in any of its inheritance rules, often a controlled violation of these rules is desired and an investigation of how to allow this is definitely desirable. Also, an actual implementation of the EPSN formalism, probably in LISP, would be a useful task and has already been started on a tentative basis.

In spite of these unsolved problems, the EPSN formalism has made several advances especially in the organization of knowledge. And, because the formalism, like LISP, allows extensions and revisions to its basics, it can be easily extended or amended to include the solutions developed in subsequent work.

6.0 ACKNOWLEDGEMENTS

I would like to thank Professors John Mylopoulos and Gord McCalla for their efforts in the supervision of this work. Hector Levesque and Professor Ray Perrault also provided valuable input into several sections of the investigation. The National Research Council of Canada provided generous support via a 1967 Science Scholarship.

7.0 BIBLIOGRAPHY

[Hendrix 75] Hendrix, G. "Expanding the utility of semantic networks through partitioning". Proceedings 4th IJCAI, Tbilisi, U. S. S. R., 1975.

[Hewitt 73] Hewitt, C. "A universal ACTOR formalism for Artificial Intelligence". Proceedings 3rd IJCAI, Stanford University, 1973.

[Levesque 77a] Levesque, H. J. "A Procedural Approach to Semantic Networks". Technical Report No. 105, Department of Computer Science, University of Toronto, 1977.

[Levesque 77b] Levesque, H. J. "Functional Programming in LISP". Department of Computer Science, University of Toronto, 1977.

[Levesque and Mylopoulos 78] Levesque, H. J. and J. Mylopoulos. "A Procedural Semantics for Semantic Networks". AI Memo 78-1, Department of Computer Science, University of Toronto, 1978.

[Mylopoulos et al 78] Mylopoulos, J., P. Bernstein, and H. K. T. Wong. "A Preliminary Specification of TAXIS: A Language for Designing Interactive Information Systems". Technical Report CCA-78-02, Computer Corporation of America, Cambridge, Mass., 1978.

[Roberts and Goldstein 77] Roberts, B. P. and I. P. Goldstein. "The FRL Primer". AI Lab Memo 408, M. I. T., 1977.

[Schneider 78] Schneider, P. F. "Organization of Knowledge in a Procedural Semantic Network Formalism". Technical Report 115, Department of Computer Science, University of Toronto, 1978.

[Sussman and McDermott 72] Sussman, G. J. and D. V. McDermott. "From PLANNER to CONNIVER: A genetic approach". Proceedings FJCC, Vol. 41, Part 2, 1972.

[Tennant 76] Tennant, R. D. "The denotational semantics of programming languages". Communications of the ACM, Vol. 19, No. 8, 1976.

[Winograd and Bobrow 76] Winograd, T. and D. G. Bobrow. "An Overview of KRL, a Knowledge Representation Language". Technical Report CSL-76-4, Xerox Palo Alto Research Center, 1976.

ON STRUCTURING A FIRST ORDER DATA BASE

Raymond Reiter

Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1W5

ABSTRACT

This paper is concerned with how a first order data base might be structured in order to guarantee finite computations at query evaluation time. It turns out that this question is intimately connected with the following: In the design of a data base, which knowledge should be represented intensionally, and which extensionally? This paper provides a criterion for extensional representations which, if fulfilled, assures finite computations during the evaluation of queries.

The paper also explores ways of proving that all queries will yield finite computations. This leads to an appropriate notion of proving the correctness of a data base.

1. INTRODUCTION

The concern of this paper is with deductive question-answering over first order data bases, and how such data bases might be structured in order to provide efficient evaluation of queries. There appear to be two distinct notions of "structuring" current in AI:

1. Structuring as indexing. For example, a data base is so structured if one can readily access all relevant information about block33 (perhaps with respect to some context). Semantic nets typically provide for this kind of structuring.
2. Structuring as the appropriate choice of relations with which to model some domain of interest. For example, if the relations chosen form a hierarchy of some sort, then this hierarchical structure can be exploited to reduce redundancy in the representation e.g. relations lower down in the hierarchy will typically inherit all properties of those above them. IS-A hierarchies are a classic example of this kind of structuring. Similarly, for "primitivists" like Schank [Schank 1973] the choice of a particular set of primitives will impose a certain structure on the data base.

There is a third notion of structure which appears not to have been articulated within the AI community, and which this paper explores. This notion concerns the possibility of infinite search paths during deductive question-answering. For a variety of reasons, such infinite paths are undesirable:

1. In those cases where all answers to a query are to be returned, the entire proof tree must be searched [Reiter 1977, 1978a]
2. Even when only one answer suffices, the entire proof tree must be searched in those cases where a query has no answers.

In this paper, we adopt as a data base structuring principle that no infinite search paths be allowed to arise.

A different notion of structuring concerns the distinction between intensional and extensional representations of facts in a data base. Loosely speaking, an extensional fact is something quite specific, like "Block1 supports block2" while an intensional fact has some generality to it, like "All men are mortal", or "Every supplier of parts supplies all their subparts". This distinction is by no means limited to first order representations. Under the so-called "procedural representation of knowledge", procedures correspond to intensional facts. Once this distinction between intensional and extensional representations is made, a natural question arises when designing a data base for a given domain: Which information should be represented intensionally, and which extensionally? As we shall see, there is an intimate relationship be-

tween this structuring question and the issue of structuring a data base so that infinite deductive paths cannot arise. Specifically, provided that certain sub-extensions of suitably designated relations are extensionally represented, all deductive paths will be finite.

Finally, we shall see that there is an appropriate notion of proving the correctness of a data base, where "correct" is taken to mean "all deductive paths will be finite". What this involves is proving that certain recursive intensional facts, which might conceivably lead to infinite deductive searches, actually yield finite search trees.

2. FORMAL PRELIMINARIES

We shall be dealing with a first order language having the usual logical symbols (quantifiers, propositional connectives), predicate signs, constant signs, but no function signs. The formulae of interest are called twffs (typed well formed formulae) which are just like ordinary first order formulae except that all variables are typed. For example, in an inventory domain, such a twff might be

$$(x/MANUFACTURER)(y/PART)(z/PART)MANUFACTURES(x,y) \wedge SUBPART(z,y) \supset SUPPLIES(x,z) \qquad (2.1)$$

i.e. Every manufacturer of a part supplies all its subparts. The restricted universal quantifier (y/PART) may be read "For every y which is a part". The restrictions MANUFACTURER and PART are called types, and are distinguished monadic predicates. If τ is such a type, then $(x/\tau)W$ is an abbreviation for $(x)\tau(x) \supset W$. We shall also require the notion of a restricted existential quantifier (Ex/τ) which may be read "there is an x in τ". $(Ex/\tau)W$ is an abbreviation for $(Ex)\tau(x) \wedge W$. We denote by $|\tau|$ the set of all constants which satisfy the type τ. Thus, |PART| might be {gadget1, widget3, bolt49,...}. In general, a twff has the form $(q_1x_1/\tau_1)\ldots(q_nx_n/\tau_n)W$ for $n\geq 0$ where (q_ix_i/τ_i) is (x_i/τ_i) or (Ex_i/τ_i), W is any quantifier-free ordinary first order formula with free variables $x_1,\ldots,x_n$ (containing no function signs) and $\tau_1,\ldots,\tau_n$ are types. Notice that twffs may not contain function signs. They may, and usually will, contain constant signs.

A data base is any set of universally quantified twffs. If DB is a data base, let EDB be the set of ground literals (literals with no variables) of DB. EDB will be called the extensional data base. The intensional data base is defined to be IDB = DB - EDB. Intuitively, the EDB is a set of specific facts like "John Doe teaches Calculus 103", while the IDB is a set of general facts like "All widgets are manufactured by Foobar Inc." Notice that only universally quantified twffs are permitted in a data base. We preclude twffs involving existentially quantified variables since such variables lead to the introduction of function signs (Skolem functions) and it is not clear how to generalize some of the results of this paper to take into account arbitrary functional terms. Examples of data base twffs are (2.1) above, as well as:

"All widgets are manufactured by Foobar Inc."
(x/WIDGET)MANUFACTURES(foobar,x)
"If block x supports block y, then y does not support x."
$(xy/BLOCK)SUPPORTS(x,y) \supset \neg SUPPORTS(y,x)$
"Acme supplies part33."
SUPPLIES(acme,part33)
"Block A supports block B."
SUPPORTS(A,B)

Of these, (2.1) and the first two twffs above are in the IDB. The last two are EDB twffs.

A query is any existentially quantified twff, for example

"Who supplies widgets?"
(Ex/SUPPLIER)(Ey/WIDGET)SUPPLIES(x,y)
"Which block supports block A?"
(Ex/BLOCK)SUPPORTS(x,A)
"Who manufactures both part33 and blue widgets?"
$(Ex/MANUFACTURER)(Ey/WIDGET)MANUFACTURES(x,part33) \wedge MANUFACTURES(x,y) \wedge BLUE(y)$

Although we treat only existential queries in this paper, it is possible, under suitable conditions, to "reduce" the evaluation of arbitrary queries (i.e. those involving both existential and universal quantifiers) to the evaluation of existential queries [Reiter 1977] so there is no loss in generality in assuming only existential queries.

Although this paper is not concerned with techniques for query evaluation, there are a few observations which must be made:

1. Typically, a query is answered by finding a proof of it using the data base as premises.

The resulting instances of one or more of the existentially quantified variables of the query provide an answer to the query.

2. In general, the data base will be so large that these proofs must all be top down, corresponding to some form of backward chaining or consequent mode of reasoning, beginning with the given query. (In theorem proving jargon, this corresponds to linear deduction with a clause of the query as top clause.) Now a serious problem with such top down reasoning is that certain intensions can lead to infinite deduction paths. For example, a transitive relation like subpart:

$$(xyz/PART)SUBPART(x,y) \wedge SUBPART(y,z) \supset SUBPART(x,z) \quad (2.2)$$

will lead to such an infinite deduction path for any goal of the form SUBPART(a,b) whenever a is not a subpart of b.

Our goal in the next section is to characterize those data bases for which such infinite deductive searches might arise, with the ultimate objective of structuring a data base in such a way as to guarantee finite deductive paths.

3. RECURSIVE DATA BASES

Notice that the intension (2.2), which can lead to an infinite deductive search, has the clausal form[1]

$$\overline{L}_1 \vee L_2' \vee C_1,\ \overline{L}_2 \vee L_3' \vee C_2,\ \overline{L}_3 \vee L_4' \vee C_3, \ldots, \overline{L}_n \vee L_1' \vee C_n \quad (3.1)$$

for literals L_i, L_i' such that $L_i\sigma = L_i'\sigma$ for some typed unifier[2] σ, $i=1,\ldots,n$. Following [Lewis 1975], we call such a sequence of clauses a cycle. We shall say that the IDB is recursive iff it contains a cycle.

To see why cycles can lead to infinite deduction trees, consider an attempted refutation of the literal L_1. By an appropriate sequence of resolution operations on the clauses of (3.1) we can deduce a clause

$$(C_1 \vee C_2 \vee \ldots \vee C_n \vee L_1')\mu$$

where μ is a substitution with σ as an instance of μ. Hence $L_1'\mu$ unifies with L_1 and we might cycle through (3.1) again with no assurance that this cycling cannot continue indefinitely.

It is intuitively clear that if a set of clauses is cycle-free, then no infinite deductions can arise. In [Lewis 1975] just such a result is proved. Now in general we cannot expect the IDB to be cycle-free. In what follows, we shall propose techniques for neutralizing the infinite recursive computations resulting from cycles in the IDB.

4. EXTENSIONAL COMPLETENESS AND RECURSION REMOVAL

In this section we propose a condition which, if satisfied by an appropriate literal of a cycle, has the effect of cutting the recursive deductive searches which would otherwise obtain from that cycle.

4.1 Extensionally Complete Literals

Suppose $L(\vec{x})$ with free variables $\vec{x}=x_1,\ldots,x_n$ is a literal of clause C. $L(\vec{x})$ is said to be extensionally complete with respect to C iff for every tuple of constants $\vec{c} \in |\tau(x_1)| \times \ldots \times |\tau(x_n)|$, either $L(\vec{c}) \in$ EDB or $\overline{L}(\vec{c}) \in$ EDB. Intuitively, if $L(\vec{x})$ is extensionally complete with respect to C, then C can contain no information about $L(\vec{x})$, since all such information is present in the EDB. At best, C specifies new information about some other literal of C in terms of the complete information that we already have about $L(\vec{x})$. Thus, we can expect that in a resolution proof it is redundant ever to resolve upon $L(\vec{x})$ except with a unit of the EDB. The following result confirms this intuition for linear resolution proofs [Loveland 1970], which are most commonly used in deductive question-answering. The reader is assumed familiar with the literature on resolution theorem proving. If Q is a query, denote by $\overline{Q}$ the set of clauses of the negation of Q.

Theorem 4.1

Suppose that a data base DB is satisfiable, and that DB $\cup$ $\overline{Q}$ is unsatisfiable. Then there is a

[1] When twffs are converted to clausal form their quantifiers are removed. Since all information about variable types is contained in the quantifiers, we assume that with each variable of a clause is associated its type. If x is a clausal variable, we shall denote its type by $\tau(x)$.

[2] Since terms (variables and constants) have types associated with them, the ususal unification algorithm [Robinson 1965] must be modified to enforce type consistency. For details see [Reiter 1977].

(typed) linear refutation[1] of DB ∪ $\overline{Q}$ with top clause in $\overline{Q}$ with the property that if a clause C ∈ IDB is used as a far parent in this refutation, and if L ∈ C is extensionally complete with respect to C, then L is not the literal of C resolved upon. Moreover, if L' is a descendant of L in this deduction, then the only resolution operation in which L' is the literal resolved upon is one in which L' is resolved away against a unit of the EDB.

Proof:

Let S = DB ∪ $\overline{Q}$ and let S_G be the set of ground instances of the clauses of S over the Herbrand universe (which consists only of the constant signs of the data base since no function signs are permitted) where each such ground instance is the result of substituting constant signs for variables consistent with the types of these variables. Clearly, S_G is unsatisfiable since DB ∪ $\overline{Q}$ is. Now let Σ_G be obtained from S_G by deleting from S_G each non EDB clause subsumed by a unit of EDB. Σ_G is unsatisfiable and since DB is satisfiable, there is a linear refutation D from Σ_G with top clause a ground instance of a clause of $\overline{Q}$. Now suppose C ∈ IDB and C contains a literal L which is extensionally complete with respect to C. Then if C_G is a ground instance of C and L_G the corresponding ground instance of L, either $L_G \in$ EDB or $\overline{L}_G \in$ EDB. By the construction of Σ_G, it follows that $C_G \in \Sigma_G$ iff $\overline{L}_G \in$ EDB. Moreover, no other clause of Σ_G other than $\overline{L}_G$ itself can contain $\overline{L}_G$. Hence, in the linear deduction D from Σ_G, C_G can serve as far parent only if L_G is not the literal of C_G resolved upon. This establishes the first claim of the theorem in the ground case. Now if C_G serves as a far parent, then L_G will occur in the resolvent so formed. Since no other clause of Σ_G other than $\overline{L}_G$ itself can contain $\overline{L}_G$, it follows that the only way to resolve upon L_G in the rest of the deduction is by resolving it against $\overline{L}_G \in$ EDB. This establishes the second claim of the theorem in the ground case. The general case follows by a suitable lifting argument.

Informally, Theorem 4.1 says that there are top down proofs of a query in which extensionally complete literals need only be resolved against the EDB. Theorem 4.1 can be proved for quite restrictive linear strategies [Reiter 1977].

4.2 Extensionally Normalized IDBs

Suppose that the IDB contains a cycle of the form (3.1). Suppose further that any one of the literals $\overline{L}_i$, L_i i=1,...,n is extensionally complete with respect to the clause in which it occurs. Then by Theorem 4.1 neither that literal nor any of its descendants in a linear deduction need ever be resolved upon except with a literal of the EDB. This means that the recursive chain of resolution operations which, for this cycle, might lead to an infinite deduction tree has been cut!

We shall say that an IDB is extensionally normalized iff for every cycle of the form (3.1) it is the case that one of the literals $\overline{L}_i$, L'_i i=1,...,n is extensionally complete with respect to the clause in which it occurs. We can summarize our observations thus far:
If the IDB is extensionally normalized, then no infinite linear deduction trees can arise, provided extensionally complete literals are resolved only against units of the EDB.

Notice that this result deals only with linear deductions which use only the clauses of IDB. It does not necessarily hold for the clauses of IDB ∪ $\overline{Q}$ since IDB ∪ $\overline{Q}$ may not be extensionally normalized even when IDB is.

Example 4.1

IDB: $(x/\tau)\overline{P}(x) \vee R(x)$
Q = $(Ex/\tau)\overline{P}(x) \wedge R(x)$
Then IDB is extensionally normalized, but
IDB ∪ $\overline{Q}$ = $\{(x/\tau)\overline{P}(x) \vee R(x),\ (x/\tau)P(x) \vee \overline{R}(x)\}$
which is not extensionally normalized.

Theorem 4.2

If IDB is extensionally normalized, and query Q has the form $(Ex_1/\tau_1)\ldots(Ex_n/\tau_n)L$ for some literal L, then no infinite linear deductions can arise in evaluating Q provided extensionally complete literals are

[1] Since all variables are typed, the usual theory of resolution theorem proving must be modified to accommodate typed unification (Footnote [2], section 3). Details for a restricted form of linear resolution may be found in [Reiter 1977]. It should be noted that the proof of Theorem 4.1 requires the "τ-completeness assumption" of [Reiter 1977], namely, for all types τ and constants c, one of τ(c) or $\overline{\tau}$(c) holds i.e. c is known to satisfy or fail to satisfy type τ.

resolved only against units of the EDB.

Proof:

$\overline{Q}$ consists of a unit clause $\overline{L}$. Since the result of adding a unit clause to an extensionally normalized set of clauses is still such a set, IDB $\cup$ $\overline{Q}$ is extensionally normalized.

For the next result, we shall require the notion of query evaluation under the closed world assumption [Clark 1978, Reiter 1978b]. Under the closed world assumption, certain answers are admitted as a result of failure to find a proof. Specifically, if no proof of a positive ground literal exists, then the negation of that literal is assumed true. This can be viewed as equivalent to implicitly augmenting the given data base by all such negated literals. In contrast, the open world assumption corresponds to the usual first order approach to query evaluation e.g. [Minker 1978]: Given a data base DB and a query Q, the only answers to Q are those which obtain from proofs of Q given DB as premises. Under the open world assumption no significance is attached to failure to find a proof. The distinction is closely related to that between logical negation and the "negation" operator of procedural languages for artificial intelligence e.g. PLANNER [Hewitt 1972].

Under fairly general conditions [Reiter 1978b] closed world data bases need explicitly represent only positive facts. Negative facts are not represented, but are inferred, when required, by default. For many domains of application, closed world query evaluation is almost mandatory, since the number of negative facts in overwhelmingly large - too large to be explicitly represented. For example, in a data base for an airline flight schedule, all flights and the cities which they connect will be explicitly represented. Flights and the cities which they do not connect will not be so represented. Instead, failure to find e.g. an entry indicating that Air Canada flight 103 connects Vancouver with New York will permit the inference that it does not.

An atomic query has the form $(Ex_1/\tau_1)\ldots(Ex_n/\tau_n)L$ where L is a positive literal. The basic result we require about closed world query evaluation is the following [Reiter 1978b]:
An arbitrary query Q can be evaluated under the closed world assumption by decomposing it into atomic queries, each of which is evaluated on the given data base under the open world assumption.
This result, when coupled with Theorem 4.2 yields the following:

Corollary 4.3

If the IDB is extensionally normalized, Q is any query, and query evaluation is in closed world mode, then no infinite linear deductions can arise in evaluating Q provided extensionally complete literals are resolved only against units of the EDB.

Corollary 4.3 completely eliminates any concern about infinite computations during query evaluation for closed world data bases, provided the IDB is extensionally normalized. For open world query evaluation this is not the case, except for one literal queries in which case Theorem 4.2 provides the necessary assurance. In general, then, for open worlds and arbitrary queries, extensionally normalized IDBs do not guarantee finite computations. Nevertheless, it is clear that such IDBs reduce the possibility of infinite computations and hence provide a valuable heuristic for open world deductive question-answering.

5. STRUCTURING A DATA BASE: INTENSIONS VS. EXTENSIONS

In principle, at least for some data bases, there is no need for an IDB. All information could be stored in the EDB. What an IDB provides is a space saving mechanism: information which might have been explicitly stored in the EDB is instead implicitly contained in the IDB and must be retrieved by deduction. In general, one would want an intensional representation of certain facts only when the corresponding extensional representation would be unfeasibly large. What we have here is a classical space-time computational trade-off, whereby the more information one stores in the EDB the less time, on the average, one requires to answer queries. There are two extremes on this space-time (or better, extension-intension) spectrum. At one extreme, all information is represented extensionally. At the other, a minimal extension is maintained and most information is represented intensionally. In general, one pays a high price for this latter extreme despite its minimal space requirements since one must then expect a recursive IDB with the attendant infinite computations we have come to expect from such data bases. What seems to be required is an appropriate balance be-

tween both extremes in which there is an optimal division of the information content of the data base into extensional and intensional components. We believe that the concept of extensional completeness, when exploited to cut cycles in the IDB (Section 4.2), provides a handle on this optimal extensional vs. intensional division of information. Specifically, we propose to represent enough information extensionally so as to render the IDB extensionally normalized.

In order to fix these ideas, we shall consider the process of designing a data base. At some point one must choose a set of relations which are to represent the relationships among the individuals in the domain being modeled. For example, if the domain is some form of inventory, then the individuals of the domain will be parts, suppliers, manufacturers, etc. and relations like

PART(x) - x is a part
MANUFACTURER(x) - x is a manufacturer
SUPPLIES(x,y) - supplier x supplies part y
SUBPART(x,y) - part x is a sub-part of part y
etc.

are likely to be of concern, and will all be members of a presumably larger fixed set of relations which are all deemed to be relevant to the class of queries which may be posed for any inventory domain. The next step in the design process is to determine, and appropriately represent, the semantics of the domain i.e. the relationships which hold among the relations like PART, SUPPLIES, etc. Thus, the fact that the relation SUBPART is transitive is part of the semantics of the inventory domain and we represent this by

$$(xyz/PART)SUBPART(x,y) \wedge SUBPART(y,z) \supset SUBPART(x,z) \quad (5.1)$$

The following might also reflect the semantics of a particular inventory domain:

"Every manufacturer of a part supplies all its sub-parts"

$$(x/MANUFACTURER)(yz/PART)MANUFACTURES(x,y) \wedge SUBPART(z,y) \supset SUPPLIES(x,z) \quad (5.2)$$

"Acme manufactures all parts it supplies."

$$(x/PART)SUPPLIES(A,x) \supset MANUFACTURES(A,x) \quad (5.3)$$

When all such semantic properties of the domain have been determined, we have a candidate IDB. The question now arises: What information do we represent in the EDB? While we have no general answer to this question, we can provide a guideline based upon the results of Section 4.2 i.e. we want to represent enough information extensionally so that the IDB is extensionally normalized. This involves the following steps:

1. Determine the cycles of the IDB, a decidable problem.
2. For each such cycle, choose a literal L which, if it were extensionally complete with respect to its clause, would cut the cycle.
3. Suppose L chosen in 2. is a literal in the predicate sign P. Represent extensionally as much of P's extension as is required to render L extensionally complete with respect to its clause in the cycle.

To see how this structuring principle might be applied in practice, consider first the intensions (5.2) and (5.3) above. These form a cycle which can be cut in any of four different ways, namely by extensionally representing the relation MANUFACTURES(x,y) and its negation if the data base is open world) for all manufacturers x and parts y, or by extensionally representing SUPPLIES(x,z) etc. The optimal choice would be to make SUPPLIES(A,x) extensionally complete with respect to (5.3) i.e. extensionally represent the relation SUPPLIES(A,x) - all parts supplied by Acme (together with parts not supplied by Acme if the data base is open world).

The intension (5.1) forms a cycle by itself. To cut it, we must extensionally represent the relation SUBPART(x,y) for parts x and y, in which case (5.1) becomes redundant since then each of its literals will be extensionally complete with respect to (5.1). Of course, one should not take too literally the need to represent the full extension of the subpart relation. In actual fact, it is sufficient to represent a "minimal" extension and to have available procedures which, given parts p_1, and p_2, can decide whether or not SUBPART(p_1,p_2) holds. For example, if SUBPART(p_i,p_{i+1}) holds, $i=1,\ldots,n-1$, then it is sufficient to extensionally represent these n-1 facts. There is no need to explicitly represent, say, the fact SUBPART(p_1,p_n) since one can easily define a procedure to deduce

this from the facts explicitly stored. Moreover, there is no commitment to any particular extensional representation for the subpart relation. Certainly, it need not be as a set of literals, or as an array. More likely, some sort of tree structured representation would be best. In this connection, notice that the subpart relation has additional properties to simple transitivity. It is asymmetric:

$$(xy/PART)SUBPART(x,y) \supset \neg SUBPART(y,x) \quad (5.4)$$

It is irreflexive:

$$(x/PART)\neg SUBPART(x,x) \quad (5.5)$$

All these properties strongly suggest that an optimal extensional representation will be tree structured, coupled with appropriate procedures for inferencing, in which case the intensions (5.4) and (5.5) need not be present in the IDB.

In general, many relations can be expected to possess special properties which, if represented in the IDB, will render it recursive. Our view is that such relations must instead be represented extensionally. Moreover, specialized data representations and inference procedures must be devised for each combination of properties possessed by such a relation. For example, an assymmetric, transitive, irreflexive relation like SUBPART will require quite different data representations and access methods than an equivalence relation. Lindsay, in [Lindsay 1973] makes essentially this point, in describing the work of Elliott [Elliott 1965]. Elliott's thesis considers nine properties, like transitivity, asymmetry etc. which a relation may possess, and classifies relations in terms of all possible meaningful combinations of these nine properties, these being 32 in number. For each of these 32, he proposes suitable extensional representations and access methods.

6. OTHER FORMS OF RECURSION REMOVAL

Although the process of filling in the extension of a suitably chosen predicate sign can always be invoked to cut a cycle in the IDB, this can occasionally be too drastic a remedy. In what follows we shall discuss certain far more economical approaches to the elimination of infinite deductions caused by cycles. While these approaches lack the full generality of that of Section 5, conditions under which they apply can be expected to arise frequently, which is why we feel they merit some attention. In those cases where they fail to apply, the method of Section 5 can be invoked.

6.1 Checking for Duplicate Subgoals

Consider the following possible intensions for the inventory domain:

"All widget suppliers supply gadgets, and vice versa."

$$(x/SUPPLIER)(y/WIDGET)(z/GADGET)SUPPLIES(x,y) \supset SUPPLIES(x,z) \quad (6.1)$$

$$(x/SUPPLIER)(y/GADGET)(z/WIDGET)SUPPLIES(x,y) \supset SUPPLIES(x,z) \quad (6.2)$$

Assuming that widgets are disjoint from gadgets, neither (6.1) nor (6.2) is, by itself, a cycle but the two together define a cycle. For simplicity, assume a definite IDB[1] and closed world assumption, in which case a conventional subgoaling or backchaining proof procedure will do for query evaluation [Reiter 1977]. Consider an attempted proof of SUPPLIES(α,β) where α and β are terms with types SUPPLIER and WIDGET respectively. Then the effect of the cycle (6.1) and (6.2) will be to generate the following infinite deduction sequence, where $\rightarrow$ denotes "current subgoal":

$\rightarrow$ SUPPLIES(α,β) $\tau(\alpha)$ = SUPPLIER $\tau(\beta)$ = WIDGET
$\rightarrow$ SUPPLIES(α,y) $\tau(y)$ = GADGET
$\rightarrow$ SUPPLIES(α,w) $\tau(w)$ = WIDGET
$\rightarrow$ SUPPLIES(α,u) $\tau(u)$ = GADGET
$\rightarrow$ SUPPLIES(α,v) $\tau(v)$ = WIDGET
.
.
.

Clearly there is no need to continue this deduction beyond the third subgoal, since the fourth is simply a renaming of the second.

It follows, in general, that we need only equip the theorem prover with the capacity to detect duplicate subgoals in order to truncate certain infinite deduction paths. Although we omit the details here, it should be clear that there is a simple sufficient condition on cycles which guarantees that a duplicate subgoal detector will truncate the infinite deduction paths which might otherwise arise. Hence, we can determine in advance which cycles of the IDB lead to finite

[1] A data base is definite iff each of its clauses contains exactly one positive literal.

deductions. For these cycles there will be no need to appeal to the extension filling techniques of Section 5.

6.2 Special Cases

In some instances, the particular structure of a cycle, together with certain specialized knowledge that is available about the relations of the cycle can be exploited to prevent infinite decuction paths. As an example, consider the intension

"All parts suppliers also provide sub-parts for those parts."

$$(x/SUPPLIER)(yz/PART)SUBPART(z,y) \wedge SUPPLIES(x,y) \supset SUPPLIES(x,z) \quad (6.3)$$

As before, assume a definite IDB and the closed world assumption so that we can appeal to a conventional subgoaling proof procedure for query evaluation. Consider an attempted proof of $SUPPLIES(\alpha,\beta)$ for terms α and β. Then the effect of the cycle (6.3) will be to generate the following infinite deduction sequence:

$\rightarrow SUPPLIES(\alpha,\beta)$

$\rightarrow SUBPART(\beta,y_1) \wedge SUPPLIES(\alpha,y_1) \quad (6.4)$

$\rightarrow SUBPART(\beta,y_1) \wedge SUBPART(y_1,y_2) \wedge SUPPLIES(\alpha,y_2)$

$\rightarrow SUBPART(\beta,y_1) \wedge SUBPART(y_1,y_2) \wedge SUBPART(y_2,y_3) \wedge SUPPLIES(\alpha,y_3)$

.
.
.

It is clear that the theorem prover is trying to establish a transitive chain $\beta,y_1,\ldots,y_n$ with respect to the subpart relation such that $SUPPLIES(\alpha,y_n)$ holds. Now suppose that, for some $n>1$, one of these subgoals succeeds, i.e. there are parts $p_1,\ldots,p_n$ such that

$\vdash SUBPART(\beta,p_1) \wedge SUBPART(p_1,p_2) \wedge\ldots\wedge SUBPART(p_{n-1},p_n) \wedge SUPPLIES(\alpha,p_n)$

Then, since the relation SUBPART is transitive,

$\vdash SUBPART(\beta,p_n) \wedge SUPPLIES(\alpha,p_n)$

which is an instance of the subgoal (6.4). Hence, we conclude that there is no need to generate any of the subgoals following (6.4) so that the cycle (6.3) will not generate an infinite deduction path. This observation leads to the following special case of recursion removal:

If the IDB contains an intension of the form

$$(x/\tau_1)(yz/\tau_2)(\vec{v}/\vec{\theta})T(z,y,\vec{v}) \wedge P(x,y,\vec{v}) \supset P(x,z,\vec{v})$$

where T is transitive in its first two arguments, then in any subgoaling proof procedure one need never recurse on this intension.

For another example of a special case of recursion removal, consider the following intension: "If an employee belongs to the dental plan, then so does his (her) spouse."

$$(x/EMPLOYEE)(y/HUMAN)DP(x) \wedge SPOUSE(x,y) \supset DP(y)$$

As before, consider a subgoaling proof of $DP(\alpha)$.

$\rightarrow DP(\alpha)$

$\rightarrow DP(x_1) \wedge SPOUSE(x_1,\alpha)$

$\rightarrow DP(x_2) \wedge SPOUSE(x_2,x_1) \wedge SPOUSE(x_1,\alpha)$

.
.
.

It is clear from the semantics of the spouse relation (Everyone has at most one spouse), that the third and subsequent subgoals in this infinite sequence are irrelevant. In general, then, we have the following special case of recursion removal: Suppose the relation R is commutative, and for any x there is at most one y such that $R(x,y)$. If the IDB contains an intension of the form

$$(x/\tau_1)(y/\tau_2)(\vec{v}/\vec{\theta})P(x,\vec{v}) \wedge R(x,y) \supset P(y,\vec{v})$$

then in any subgoaling proof procedure one need never recurse on this intension.

It is easy to see that the same result holds, if, instead, the relation R has the property that if $R(x,y)$ then for no z do we have $R(z,x)$.

As a final example, consider the ubiquitous blocks world, in particular the following recursive axiom which captures part of the definition of the ON relation in terms of the "directly supports" relation:

$$(xyz/BLOCK)ON(x,y) \wedge SUPPORTS(y,z) \supset ON(x,z) \quad (6.5)$$

Assume further that SUPPORTS (not ON) is extensionally complete with respect to this axiom so that for all blocks b_1 and b_2 the system knows whether or not b_1 directly supports b_2. Again, consider a subgoaling proof of $ON(\alpha,\beta)$:

$\rightarrow ON(\alpha,\beta)$

$\rightarrow ON(\alpha,y) \wedge SUPPORTS(y,\beta) \quad (6.6)$

Since SUPPORTS is extensionally complete, we can discharge or fail to discharge the second conjunct $SUPPORTS(y,\beta)$. If the latter, then the attempted

proof of $ON(\alpha,\beta)$ fails and we are done. Otherwise this conjunct can be discharged, say with block b_1 for y and we are left with the subgoal

$\rightarrow ON(\alpha,b_1)$

We can repeat this same process with this subgoal. It is not difficult to see that for a world with n blocks, this process of recursively back chaining into (6.5) must terminate after at most n-1 back chaining operations.

It is important to observe that the proof that (6.5) will not yield an infinite deductive path relies upon the order in which conjuncts in the conjunctive subgoals generated are discharged. Specifically, the proof requires that in conjunctive subgoals like (6.6), the literal $SUPPORTS(y,\beta)$ be discharged first. In other words, termination is dependent on the theorem prover realizing an appropriate control structure. In general, then, we can expect that termination proofs for recursive axioms will be control dependent; we shall need to use our knowledge of the consequences of following certain deduction paths to appropriately control the action of a theorem prover. This observation strongly suggests the need for a suitable control language which would be used to prevent the theorem prover from straying [Hayes 1973]. Such a language, if sufficiently expressive, would provide a uniform facility for representing special cases of termination like those of this section.

7. ON PROVING THE CORRECTNESS OF A DATA BASE

The treatment of special cases of Section 6 suggests an alternate view of the structuring principles which we have been proposing. For what they amount to is proving the correctness of a data base, where "correct" is taken to mean "all deductive paths will be finite". Such proofs may be as simple as appealing to the extensional completeness of certain predicates, as in Section 5, or as complex as the proof that (6.5) terminates. We want now to argue that "termination" is the only appropriate notion of "correct" in this context. For it is difficult to imagine a more succinct and perspicuous specification language than that of first order logic. Accordingly, a first order data base is a specification of some domain and there is no need to prove that it is a correct specification with respect to one in some more elementary specification language. Moreover, by definition, the answer to a query is obtained from a first order proof of that query so, assuming that the underlying theorem prover is correct, the correctness of the answers returned cannot be doubted. These observations leave just one correctness property open to question, and that is whether an answer will always be forthcoming i.e. whether "the data base terminates". One way of viewing this paper then is as an extended argument in favour of proving the correctness of data bases. In addition, we have proposed a few techniques for appropriately structuring a data base so as to make such proofs possible.

ACKNOWLEDGEMENT

This paper was written with the financial support of the National Research Council of Canada under grant A7642. Part of this research was done while the author was visiting at Bolt Beranek and Newman Inc., Cambridge, Mass.

REFERENCES

Clark, K., (1978). "Negation as Failure," in Logic and Data Bases, H. Gallaire and J. Minker (Eds.), Plenum Press, New York, to appear.

Elliott, R.W., (1965). A Model for a Fact Retrieval System, unpublished doctoral dissertation, The University of Texas at Austin, 1965.

Hayes, P.J., (1973). "Computation and Deduction," PROC. Math. Foundations of Computer Science Symposium, Czech. Academy of Sciences, 1973.

Hewitt, C., (1972). Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot, AI Memo No. 251, MIT Project MAC, Cambridge, Mass., April 1972.

Lewis, H.R., (1975). "Cycles of Unifiability and Decidability by Resolution," Aiken Computation Laboratory, Harvard University, Technical Report, 1975.

Lindsay, R.K., (1973). "In Defense of Ad Hoc Systems," in Computer Models of Thought and Language, R.C. Schank and K.M. Colby (Eds.), Freeman and Co., San Francisco, Cal., 1973,372-395.

Loveland, D.W., (1970). "A Linear Format for Resolution," in Lecture Notes in Mathematics 125 (Symposium on Automatic Demonstration), Springer-Verlag, Berlin, 1970, 147-162.

Minker, J., (1978). "An Experimental Relational Data Base System Based on Logic," in Logic and Data Bases, H. Gallaire and J. Minker (Eds.), Plenum Press, New York, to appear.

Reiter, R., (1977). "An Approach to Deductive Question-Answering," Technical Report 3649, Bolt Beranek and Newman Inc., Cambridge, Mass., Sept. 1977, 161 pp.

Reiter, R., (1978a). "Deductive Question-Answering on Relational Data Bases," in Logic and Data Bases, H. Gallaire and J. Minker (Eds.), Plenum Press, New York, to appear.

Reiter, R., (1978b). "On Closed World Data Bases," in Logic and Data Bases, H. Gallaire and J. Minker (Eds.), Plenum Press, New York, to appear.

Robinson, J.A., (1965). "A Machine Oriented Logic Based on the Resolution Principle," J.ACM, 12 (January 1965), 25-41

Schank, R., (1973). "Identification of Conceptualizations Underlying Natural Language," in Computer Models of Thought and Language, R.C. Schank and K.M. Colby (Eds.), W. H. Freeman Press, San Francisco, Cal., 1973.

THE GENETIC GRAPH
A REPRESENTATION FOR THE EVOLUTION OF PROCEDURAL KNOWLEDGE[1,2]

Ira. P. Goldstein
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Ma. 02139

Abstract

I shall describe a theory of the evolution of rule-structured knowledge that serves as a cornerstone of our development of computer-based coaches. The key idea is a graph structure whose nodes represent rules, and whose links represent various evolutionary relationships such as generalization, correction, and refinement. I shall define this graph and describe a student simulation testbed which we are using to analyze different genetic graph formulations of the reasoning skills required to play an elementary mathematical game.

Keywords: Information Processing Psychology, Learning, Knowledge representation, CAI, ICAI, AI.

Outline

1. A Learner-based paradigm for AICAI research is evolving.
2. The genetic graph has evolutionary roots in AICAI research.
3. Wumpus serves as an experimental domain.
4. The genetic graph formalizes the syllabus.
5. Genetic graphs are being explored via student simulations.
6. The genetic graph is a framework for a theory of learning.

1. A Learner-based Paradigm for AICAI is evolving.

The 1970's has seen the evolution of a new generation of computer-aided instructional programs based on the inclusion of AI-based expertise within the CAI system. These systems surmount the restrictive nature of older script-based CAI by supplying "reactive" learning environments which can analyze a wide range of student responses by means of an embedded domain-expert. Examples are AICAI tutors for geography [Car70], electronics [Bro73], set theory [Smi75], Nuclear Magnetic Resonance spectroscopy [Sle75], and mathematical games [Bur76, Gol77a].

However, while the inclusion of domain expertise is an advance over earlier script-based CAI, the tutoring theory embedded within these benchmark programs for conveying this expertise is elementary. In particular, they approach teaching from a subset viewpoint: expertise consists of a set of facts or rules. The student's knowledge is modelled as a subset of this knowledge. Tutoring consists of encouraging the growth of this subset, generally by intervening in situations where a missing fact or rule is the critical ingredient needed to reach the correct answer.

This is, of course, a simplification of the teaching process. It has allowed research to focus on the critical task of representing expertise. But the subset viewpoint fails to represent the fashion in which new knowledge evolves from old by such processes as analogy, generalization, debugging, and refinement.

This paper explores the genetic graph as a framework for representing procedural knowledge from an evolutionary viewpoint,[3] thereby contributing to the movement of AICAI from an *expert-based* to a *learner-based* paradigm.[4]

2. A graph representation of the syllabus has roots in AICAI research.

Scholar [Car70], the earliest of the AICAI tutors, employed a graph (semantic net) representation for declarative facts about geography. The graph, however, encoded only domain specific relationships; it did not embody a series of progressively more refined levels of geography knowledge linked by various evolutionary relationships.

SOPHIE-I [Bro73], the next major AICAI milestone, was an expert-based system for the more complex domain of electronic troubleshooting. SOPHIE-I compared a student's

---

1. This research was supported under NSF grant SED77-19279.
2. This paper has evolved from many fruitful conversations with members of my Cognitive Computing Group at the MIT AI Lab and with members of John Seely Brown's ICAI group at Bolt, Beranek and Newman.
3. A potential confusion in terminology may occur here. The term "genetic" is often equated with heredity. However, I use it here in its older sense, namely, the genetic method is the study of the origins and development of a phenomena. This paper is an exercise in Genetic Epistemology, the study of the origin and development of knowledge. This enterprise has been articulately advocated by Piaget [Pia70], who considers it the foundation on which psychology should be based.
4. There are other dimensions to this paradigm shift that include: (1) more sophisticated modelling of the student's knowledge and learning style [Bur76, Bro77a, Gol78a], (2) widening the communication channel from student to teacher via natural language interfaces [Bur77], and (3) developing a theory of teaching skills [Col75]. Goldstein and Brown [Gol78b] provide an overall perspective.

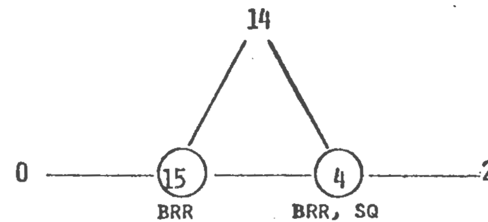
Fig. 1. An Interaction with the Wumpus Game

You are at cave 15 with neighbors 4, 14 and 0. Brrr! There is a draft. You are near a pit. What now?

1> 4

You are at cave 4 with neighbors 15, 14 and 2. Brrr! There is a draft. Squeak! A bat is near. What now?

2> 14



(E1) Mary, it isn't necessary to take such large risks with pits. There is multiple evidence of a pit in cave 14 which makes it quite likely that cave 14 contains a pit. It is less likely that cave 0 contains a pit. Hence, we might want to explore cave 0 instead. Do you want to take back your move?

---

troubleshooting hypotheses for an electronic circuit with that of its embedded expert and offered advice when the student's analysis went astray. It employed a procedural rather than a network representation for its electronics knowledge, but this representation was largely a black box. SOPHIE-I did not have access to a detailed, modular, human-oriented representation of troubleshooting skills. Nor did it have a representation for the genesis of these skills.

SOPHIE-2, now under development, will incorporate a modular, anthropomorphic representation for the expert's knowledge [Dek76]. This structured expertise serves as a better foundation for expert-based tutoring, but still is not a model of how the student evolved to that level of competence.

BUGGY [Bro77a], a program for building procedural models of a student's arithmetic skills, does incorporate both a graph representation for the basic skills and some evolutionary relationships. The basic skill representation is a graph with links representing the skill/subskill relationships. The evolutionary component consists of "deviation" links to "buggy" versions of the various skills.

BIP-II [Wes77], a tutor for programming skills, again employs a network for the basic skill representation, but embodies a different set of evolutionary relationships. There are links for representing analogy, generalization, specialization, prerequisite, and relative difficulty relations. The BIP-II skill network, however, does not include deviation links nor define an operational expert for the programming domain. Rather it employs author-supplied exercises attached to the relevant skills in the network.[5]

The genetic graph is a descendant of these network representations. Its nodes are the procedural skills of players of varying proficiency and its links include the analogy, specialization, generalization and prerequisite relations of BIP-II and the deviation relationships of BUGGY.

3. Wumpus serves as an experimental domain.

Designing coaches for the maze exploration game Wumpus [Yob75] has proven to be a profitable experimental domain because the game exercises basic skills in logic and probability.[6] The player is initially placed somewhere in a warren of caves with the goal of slaying the Wumpus. The difficulty in finding the beast arises from the existence of dangers in the warren -- bats, pits and the Wumpus itself. Pits and the Wumpus are fatal; bats move the player to a random cave elsewhere in the warren. But the player can infer the probable location of dangers from warnings he receives. The Wumpus can be sensed two caves away, pits and bats one cave away. Victory results from shooting an arrow into the Wumpus's lair; defeat if the arrows are fruitlessly exhausted.

In 1976 we developed WUSOR-I [Sta76], an expert-based coach. Skilled play was analyzed in terms of rules such as these:

> *Positive Evidence: A warning implies that a danger is in a neighboring cave.*
>
> *Elimination: If a cave has a warning and all but one of its neighbors are known to be safe, then the danger is in the remaining neighbor.*
>
> *Multiple Evidence: Multiple warnings increase the likelihood that a given cave contains a danger.*

As fig. 1 illustrates, WUSOR-I explained a rule if its employment would result in a better move than the one chosen by the student.

---

5. MALT [Kof75], a tutor for machine language programming, does include an "expert" for problems composed from a limited set of skills and solved in a tutor-prescribed order. However, MALT's syllabus of skills are related only by the probability with which MALT includes them in a system-generated problem, and not by any evolutionary links. Hence, MALT does not have BIP's ability to choose a problem based on its evolutionary relationship to the student's current knowledge state.

6. Our group is also exploring evolutionary epistemologies for other domains ranging from elementary programming to airplane flying.

WUSOR-I was insensitive to the relative difficulty of the various Wumpus skills. In 1977 we took our first steps toward an evolutionary epistemology with WUSOR-II [Carr77a], wherein the rule set was divided into five subsets or phases representing increasing skill at the task.

Phase 1: Rules for visited, unvisited and fringe caves.
Phase 2: Rules for possibly dangerous, definitely dangerous and safe caves.
Phase 3: Rules for single versus multiply dangerous caves.
Phase 4: Rules for "possibility sets", i.e. keeping track of the sources of dangers.
Phase 5: Rules for numerical evidence.

The tutor did not describe the rule of a particular level of play until it believed the student was familiar with the rules of the preceding levels.[7]

These phases constituted a coarse genetic epistemology, better than the completely unordered approach of WUSOR-I, but still far from a detailed platform on which to build new knowledge from old in the student's mind. WUSOR-III, now being implemented, addresses this limitation. It has evolved from WUSOR-II by defining a set of symbolic links between rules that characterize such relationships as analogy, refinement, correction, and generalization. The result is that the "syllabus" of the coach has evolved from an unordered skill set to a *genetic graph* of skills linked by their evolutionary relationships.

4. The genetic graph formalizes the syllabus.

The "genetic graph" (GG) formalizes the evolution of procedural rules by representing the rules as nodes and their interrelationships as links. In this section I discuss four of these relationships -- generalization/specialization, analogy, deviation/correction, and simplification/refinement -- and show the explanatory leverage they supply by allowing variations on the basic Wusor-II explanations.[8]

R' is a *generalization* of R if R' is obtained from R by quantifying over some constant.[9] *Specialization* is the inverse relation. In the Wumpus syllabus, for each trio of specialized rules for bats, pits and the Wumpus, there is usually a common generalization in terms of warnings and dangers.[10] Fig. 2 illustrates such a cluster for rule 2.2 which represents the deduction: "a warning implies that the neighbors of the current cave are dangerous."

R' is *analogous* to R if there exists a mapping from the constants of R' to the constants of R. This is the structural definition employed by Moore and Newell [Moor73]. Of course, not all analogies defined in this fashion are profitable. However, the GG is employed to represent those that are. Fig. 2 illustrates analogy links between the specialization trio of R2.2. The similar nature of the dangers of the Wumpus world make this kind of densely linked cluster common. As fig. 3 shows, identifying such clusters provides teaching leverage by providing multiple methods of explanation (one per link) for each constituent rule.[11]

R' is a *refinement* of R if R' manipulates a subset of the data manipulated by R. *Simplification* is the inverse relation. This relation represents the evolution of a rule to take account of a finer set of distinctions. The Wumpus syllabus contains five major refinements corresponding to the five Wusor-II phases. Fig. 2 illustrates the refinement of the rule R1.1 through phases 1, 2 and 3. Fig. 3 shows a refinement-based explanation.

R' is a *deviation* of R if R' has the same purpose as R but fails to fulfill that purpose in some circumstances. *Correction* is the inverse relation. Deviations arise naturally in learning as the result of simplifications, overgeneralizations, mistaken analogies, and so on. While any rule can have deviant forms, the GG is used to record the more common errors.[12]

A deviant Wumpus rule is: "If there is multiple evidence that a cave contains a pit, then that cave definitely contains a pit." The debugged rule includes the additional condition that there is only <u>one</u> pit in the warren. The deviation has a natural genetic origin: it is a reasonable rule in the early stages of Wumpus play when the game is simplified by the coach to contain only one of each danger.

---

7. [Carr77b] describes the mechanisms by which it estimated the student's position in the syllabus.
8. The GG also supplies modelling leverage [Gol78a]. A procedural model of the student can be constructed in terms of the regions of the GG with which the student has displayed familiarity. The GG does not solve the enormous difficulties which exist in inducing student models (discussed in [Carr77b, Bro77c], but it does provide another source of guidance: the evolutionary links of the known regions suggest which skills the student will acquire next.
9. This is a standard predicate calculus definition, applied here to quantifying over formulas representing rules rather than logical statements.
10. In one version of Wumpus, the wumpus warning propagates only one cave. In this case, bats, pits and the wumpus are exactly analogous. In more complex versions, the Wumpus is no longer exactly analogous. Hence, the analogies to bats and pit rules are in fact restricted cases or outright deviations. We represent this in the GG explicitly, thereby giving the coach an expectation for the traps the student will encounter.
11. There are many difficulties in generating explanations not solved by employing a GG: when should the coach interrupt, how often, how much should be said, which variation should be chosen? The utility of the GG is only to increase the range of possibilities open to the tutor.
12. The deviant skills recorded in the GG account for errors arising from the *<u>correct</u> application of <u>incorrect</u> rules*. There is another class of errors arising from the *<u>incorrect</u> application of <u>correct</u> rules*. These are errors arising from such causes as the occasional failure to check all preconditions of a rule, the misreading of data, or confusion in the bookkeeping associated with a search process. Sleeman [Sle77] explores some errors of this class in his construction of a coach which analyzes a student's description of his algorithm. Sleeman's coach, however, does not have a representation for deviant or simplified versions of the algorithm to be tutored: indeed, he assumes that the student is familiar with the basic algorithm. A possible extension of his system would be to include a GG representing evolutionary predecessors of the skilled expert.

Fig. 2. A Region of the Genetic Graph

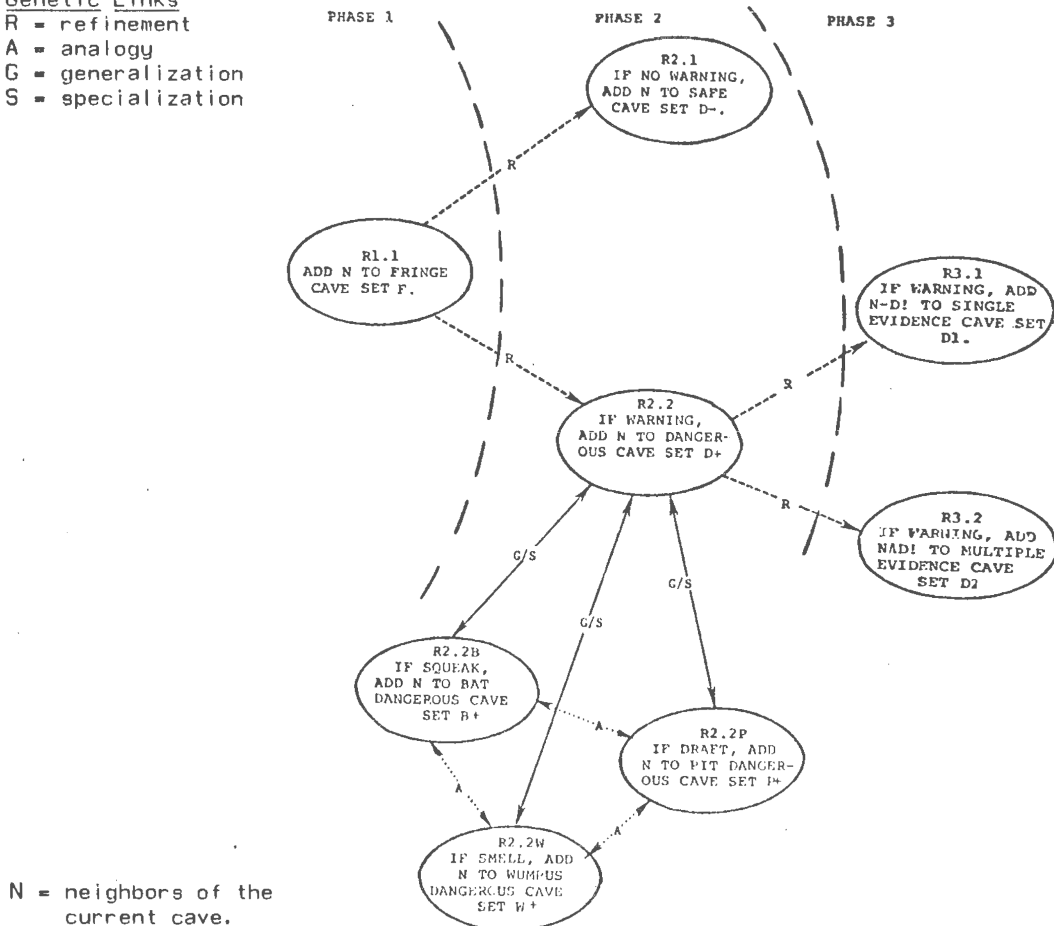
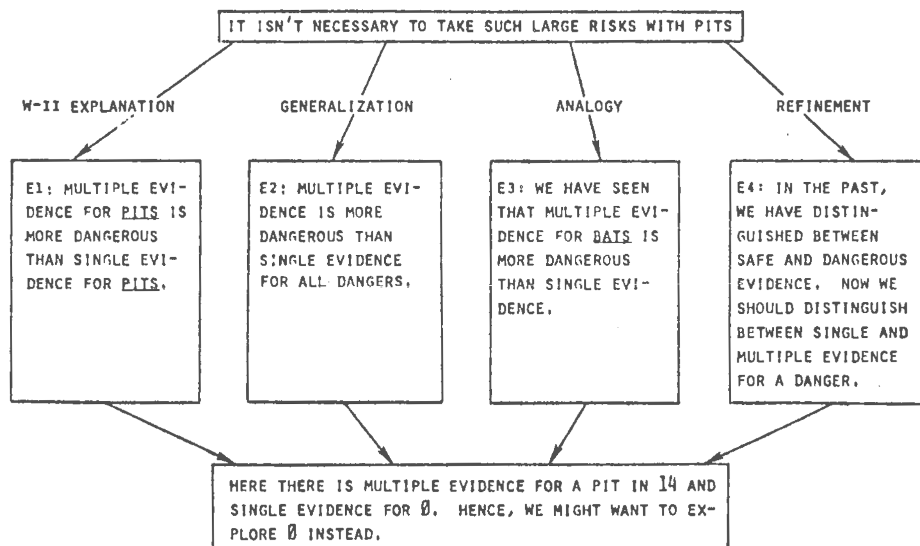


Fig. 3. Variations on an Explanation



E1 is the WUSOR-II explanation triggered by a move to cave 14 of fig. 1. E2, E3 and E4 are proposed WUSOR-III variations generated by explaining a rule in terms of its evolutionary relatives.

Again, the genetic link supplies explanatory power. Suppose the student has just transitioned from games with one pit to games with multiple pits and is in the situation of fig. 4. If he moves to cave 0, E5 is inferior to E6 as an explanation as it fails to address the probable cause of the student's difficulty, namely a belief that cave 0 is in fact safe from pits.

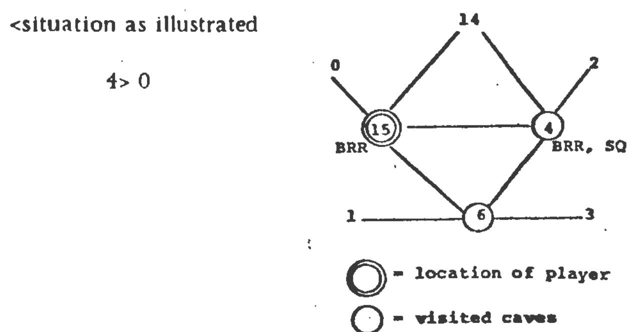
5. Genetic graphs are being explored in a student simulation testbed.

The Wumpus GG currently contains about 100 rules and

Fig. 4. A Correction Based Explanation

<situation as illustrated

4> 0



E5: Ira, we needn't risk a pit. Cave 3 is safe. Hence, we might want to explore cave 3 instead.

E6: Ira, we needn't risk a pit. Cave 3 is safe. *If there were only one pit, cave 0 would also be safe. But in this game there are 2 pits. Therefore we cannot be certain that the pit is in 14 and 0 is safe.* Hence we might want to explore cave 3 instead.

---

300 links.[13] We are currently testing the reasonableness of this graph by means of a "Student Simulation Testbed". [14] In this testbed, the performance of various simulated students, defined in terms of different regions of the GG, is being examined. These students correspond to different evolutionary states. Fig. 5 is the comparative trace of two students corresponding to the mastery of phases 2 and 3 respectively.

Expert-based CAI allows only for the definition of "computer students" formed from subsets of the expert's skills. The power of the GG to broaden the tutor's understanding of the task is evident from the testbed: the GG permits not only the creation of subset students, but also students formed from specializations, deviations, and simplifications of the expert's rules.

6. The genetic graph is a framework for a theory of learning.

This paper has explored the construction of new knowledge in terms of a genetic graph. Implicit in this structure is the following view of learning: new rules are constructed from old in terms of processes corresponding to the individual links. However, the graph does not describe a unique evolutionary path. One learner may rapidly acquire a generalization, another may first build several specializations before constructing the generalization, while a third may never acquire the generalization. Hence, the tutor should encourage this idiosyncratic construction of new knowledge by giving advice appropriate to the learner's current knowledge state (position in the graph) and particular style of learning (preference for particular links). The redesign of the Wumpus coach to employ the guidance of the GG to more closely approximate this ideal tutoring behavior is a major thrust of our current research activity.

The GG is, of course, only a framework for a theory of learning. Hard questions about learning remain to be studied: When should a learning strategy be applied? How are profitable analogies, generalizations and refinements to be detected? When should portions of the graph be forgotten? However, the graph as it stands is sufficient to formalize the relations between items of knowledge in a syllabus so as to improve the tutoring capacity of an AICAI system.

Nevertheless, it must be stressed that the evolutionary relations discussed here remain both underspecified and incomplete. There are many kinds of analogies, generalizations, and corrections. There are also other kinds of evolutionary processes for acquiring knowledge: learning by being told, learning by induction, and learning by recombining pieces of old rules in new ways.

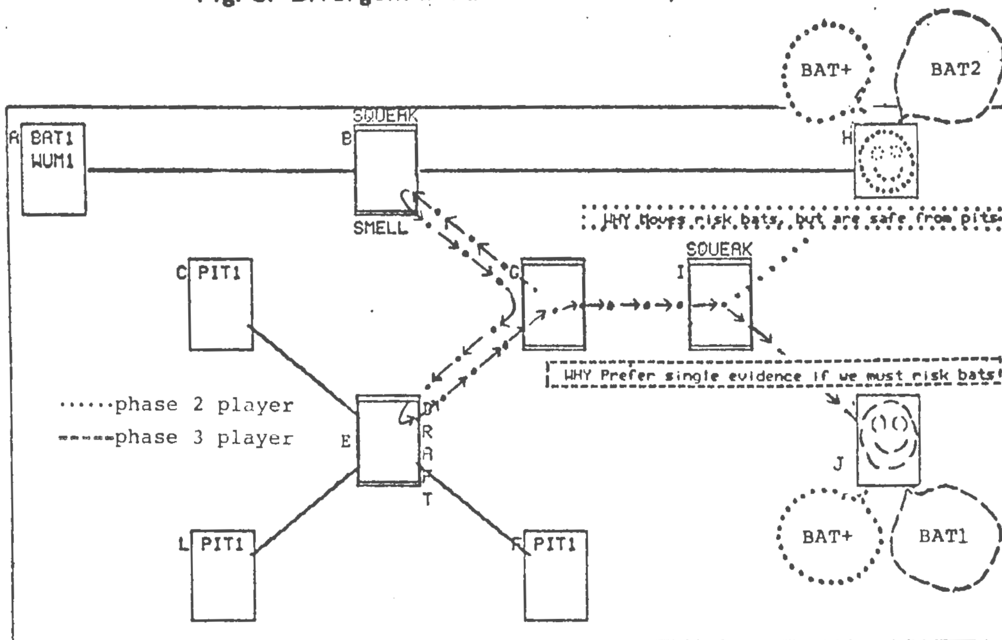
Furthermore a syllabus of procedural skills should contain various kinds of knowledge orthogonal to the evolutionary links between rules including (1) *meta-knowledge* of the relations between rules, e.g. planning knowledge regarding their sequencing, (2) *foundational* knowledge regarding the declarative

---

13. These statistics are based on an explicit representation of each generalization, its specializations and their common deviations. It is possible for the graph to be less extensive if procedures for generating common deviations and specializations are supplied. This is the approach we shall eventually employ: Specializations are simple to generate. Deviations are suggested by the common bug types enumerated by such work as my own analysis of Logo programs [Gol75], Sussman's analysis of Blocks world programs [Sus75], and Stevens and Collins's study of bugs in causal reasoning [Ste77]; or they can be induced, for simple cases, by analyzing the student's performance [Sel74, Gol75, Bro77b]. However, my current research strategy has been to make the graph explicit, in order to understand its form. The next stage will include the extension to expanding the graph dynamically.

14. The testbed serves other purposes as well. Computer students can be used to test the modelling and tutoring of teaching systems [Carr77a, Sel77, Wes77]. They can also serve as models of real students, and hence can yield insight for a human teacher observing their performance [Bro77a, Gol77b].

Fig. 5. Divergent Behavior of Two Computer Students



The "WHY" messages are printed by the student simulator as the rules defining a student are executed. The comments inside cave boxes represent hypotheses of the simulated student regarding that cave. The balloons reflect the differing hypotheses of the two students regarding bat evidence for caves J and H.

The phase 2 student (dotted path) does not know the multiple evidence heuristic. Hence, he does not realize that cave J is to be preferred over cave H. While he understands that they both risk bats, he makes no further distinction. Thus, he randomly selects from these two possibilities, unfortunately choosing the riskier H. The phase 3 student (dashed path) recognizes multiple (BAT2) evidence as more risky than single (BAT1) evidence and therefore selects the safer cave J.

This figure is a composite of the graphic output for the two students. The testbed only executes a single student at a time. It does not generate balloons nor place the "WHY" messages on the Warren itself.

---

facts which provide justification, (3) *historical* knowledge regarding the past uses of a rule, i.e. the examples from which it may have been induced, its failures and successes, and (4) *organizational* knowledge which groups rules concerned with a common concept, e.g. all the rules concerned with a given Wumpus phase. Supplying these kinds of knowledge in turn allows a deeper analysis of the genesis of the basic skills, since the evolutionary route may involve (1) the gradual debugging of their planning structure, (2) the refinement of a declarative foundation with new rules created by deduction from this foundation, (3) the induction of rules from examples,[15] and (4) the acquisition of rules in groups. [Gol78a] discusses these extensions.

In a more developed evolutionary epistemology, all of these extensions must be considered. The payoff for this labor is that each new evolutionary relationship provides further tutoring leverage.

---

15. Self's Concept Learning Program [Sel77] embodies a set of heuristics for inducing a rule from examples. These heuristics might be used as guidelines to relate examples to a rule by "induction" links. The tutor could employ such links to ascertain when tutoring leverage might be gained by suggesting to the student that an induction is possible. (Self's Concept Teaching Program acts only by constructing examples, it never offers explicit advice about the possibility of an induction.)

7. References

[Bro73] Brown, J.S., R. Burton and F. Zdybel, "A Model-Driven Question-Answering System for Mixed-Initiative Computer-Assisted Instruction", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-3, No. 3, May 1973, pp. 248-257.

[Bro77a] Brown, J.S., R. Burton and K. Larkin, "Representing and Using Procedural Bugs for Educational Purposes" Proceedings of 1977 Annual Conference, Association for Computing Machinery, Seattle, Oct. 1977, pp. 247-255.

[Bro77b] Brown, J.S., and R. Burton, Diagnostic Models for Procedural Bugs in Basic Mathematical Skills", ICAI No. 10, Bolt, Beranek and Newman, August 1977,

[Bro77c] Brown, J.S., R. Burton, C. Hausmann, I. Goldstein, B. Huggins and M. Miller, Aspects of a Theory for Automated Student Modelling, BBN Report No. 3549, ICAI Report No. 4, Bolt, Beranek and Newman, Inc., May 1977.

[Bur76] Burton, R. & J. S. Brown, "A Tutoring and Student Modelling Paradigm for Gaming Environments", in R. Coleman and P. Lorton, Jr. (Eds.), Computer Science and Education, ACM SIGCSE Bulletin, Vol. 8, No. 1, Feb. 1976, pp. 236-246.

[Bur77] Burton, R. & J.S. Brown, Semantic Grammar: A Technique for Constructing Natural Language Interfaces to Instructional Systems, BBN Report No. 3587, ICAI Report No. 5, May 1977.

[Car70] Carbonell, J., "AI in CAI: An Artificial-Intelligence Approach to Computer-Assisted Instruction", IEEE Transactions on Man-Machine Systems, Vol. MMS-11, No. 4, December 1970.

[Carr77a] Carr, B., Wusor II: A Computer Aided Instruction Program With Student Modelling Capabilities, MIT AI Memo 417 (LOGO Memo 45), May 1977.

[Carr77b] Carr, B. & I. Goldstein, Overlays: A Theory of Modelling for Computer Aided Instruction, MIT AI Memo 406 (LOGO Memo 40), February 1977.

[Col75] Collins, A., E. Warnock and J. Passafiume, "Analysis and Synthesis of Tutorial Dialogues", in G. Bower (Ed.), The Psychology of Learning and Motivation, Vol. 9, New York: Academic Press, 1975.

[Dek76] De Kleer, Johan, Local methods for Localizing Faults in Electronic Circuits, MIT AI Memo 394, Nov., 1976.

[Gol75] Goldstein, I., "Summary of MYCROFT: A System for Understanding Simple Picture Programs", Artificial Intelligence Journal, Vol. 6, No. 3, Fall 1975.

[Gol77a] Goldstein, I. and B. Carr, "The Computer as Coach: An Athletic Paradigm for Intellectual Education", Proceedings of 1977 Annual Conference, Association for Computing Machinery, Seattle, October 1977, pp. 227-233.

[Gol77b] Goldstein, I. and E. Grimson, Annotated Production Systems: A Model for Skill Acquisition, MIT AI Memo 407 (LOGO Memo 44), February 1977.

[Gol78a] Goldstein, I.P., The Genetic Epistemology of Rule Systems, MIT AI Memo 449, January 1978.

[Gol78b] Goldstein, I.P. and J.S. Brown, The Computer as Cognitive Tool, MIT AI Memo, in preparation.

[Kof75] Koffman, E. and S. Blount, "Artificial Intelligence and Automatic Programming in CAI", Artificial Intelligence, Vol. 6, 1975, pp. 215-234.

[Moor73] Moore, J. and A. Newell, "How Can MERLIN Understand?", in L. Gregg (Ed.), Knowledge and Cognition, Potomac, MD: Lawrence Erlbaum Associates, 1973.

[Sel74] Self, J., "Student Models in Computer-Aided Instruction", International Journal of Man-Machine Studies, Vol. 6, 1974, pp. 261-276.

[Sel77] Self, J., "Concept Teaching". Artificial Intelligence Journal, Vol. 9, No. 2, Oct. 1977, pp. 197-221.

[Sle75] Sleeman, D., "A Problem-Solving Monitor for a Deductive Reasoning Task", International Journal of Man-Machine Studies, Vol. 7, 1975, pp. 183-211.

[Sle77] Sleeman, D., "A System Which Allows Students to Explore Algorithms", Proceedings of the Fifth International Joint Conference on Artificial Intelligence, August 1977, pp. 780-786.

[Smi75] Smith, R.L., H. Graves, L.H. Blaine, & V.G. Marinov, "Computer-Assisted Axiomatic Mathematics: Informal Rigor", in O. Lecareme & R. Lewis (Eds.), Computers in Education Part I: IFIP, Amsterdam: North Holland, 1975.

[Sta76] Stansfield, J., B. Carr and I. Goldstein, Wumpus Advisor I: A First Implementation of a Program that Tutors Logical and Probabilistic Reasoning Skills, MIT AI Laboratory Memo No. 381, Sept. 1976.

[Ste77] Stevens, A. and A. Collins, "The Goal Structure of a Socratic Tutor", Proceedings of 1977 Annual Conference, Association for Computing Machinery, Seattle, October 1977, pp. 256-263.

[Sus75] Sussman, G., A Computational Model of Skill Acquisition, New York: American Elsevier, 1975.

[Wes77] Wescourt, K., M. Beard and L. Gould, "Knowledge-Based Adaptive Curriculum Sequencing for CAI: Application of a Network Representation", Proceedings of 1977 Annual Conference, Association for Computing Machinery, October 1977, pp. 234-240.

[Yob75] Yob, G., "Hunt the Wumpus", Creative Computing, September/October, 1975, pp. 51-54.

LOW-LEVEL VISION, CONSISTENCY, AND CONTINUOUS RELAXATION

Steven W. Zucker
Computer Vision & Graphics Laboratory
Department of Electrical Engineering
McGill University
Montreal, Quebec, Canada

ABSTRACT

The low-level vision problem has two components: the development of representations for the information content in images and the development of algorithms for computing descriptions in terms of those representations. This paper concentrates on the algorithms and their use of consistency to reduce the effects of ambiguity and noise. In particular, when the vision problem can be decomposed into networks of local problems, relaxation labeling processes, with their explicit use of consistency criteria, naturally arise. Examples of relaxation labeling processes for refining low-level descriptions at both single and multiple (hierarchical) levels of abstraction are presented, together with theoretical results relating relaxation and consistency to histogram peak selection.

1. INTRODUCTION

Vision can be seen, from an information processing point of view, as the process of abstracting useful scene descriptions out of raw intensity arrays. The specific content of such descriptions, as well as the representational formalisms within which it can be posed, are, to a certain extent, problem or goal specific. That there should be many different representations, some of which are (essentially) stable, follows from the complexity of the visual process. Or, to put this another way, the fantastic number of final descriptions potentially recoverable from every image implies an organization of visual processing into modular, interacting stages. Then, each possibility eliminated from an earlier stage prunes an entire subtree of possibilities from later ones.

In this paper we shall concentrate on the low-level (or early) stages of visual processing. The two central problems facing designers of systems for accomplishing this are (i) the development of representations that make the useful information explicit, i.e., deciding precisely what needs to be represented, and (ii) the development of algorithms for computing descriptions in terms of these representations. Our primary emphasis will be on algorithms, although we shall make some general remarks about the form of these representations.

The more general motivation behind this paper is to study the computational aspects that are common to many of the different algorithms now available for visual processing. The common thread through these algorithms, which arises in many different forms, is the use of consistency to counteract the effects of ambiguity and noise. In the next section we review some common low-level features and measurements aimed at their detection. This introduces a decomposition of the global feature-detection problem into many local ones, which provides the beginnings of a structure on which consistency criteria can be posed and sets the path toward a discussion of cooperative algorithms designed to use these consistency criteria. Surprisingly, when consistency is present initially, the complex cooperative algorithms are shown to reduce formally to simple peak-selection processes. Finally, the distribution of the consistency computation across hierarchical systems is described.

2. LOW-LEVEL FEATURE DETECTION

The most common low-level features are directly related to patterns of intensity values. For example, an edge feature signals the border between two areas of differing intensity, while a line feature signals a thin pictorial area of one intensity flanked on both sides by constant areas of different intensities. The importance attributed to features such as these rests, in part, on our ability to interpret line drawings as readily as gray-level images. More generally, however, they point toward representations designed to capture all of the relevant information that is encoded in the intensity values.

The first stage in computing the presence of intensity features has traditionally been to perform a measurement over the image. Because of the local nature of many of these features, and because of the computational expense required in evaluating global measurements, these measurements have mainly been local. Their explicit form is a mapping from a small domain of intensity values into a number, and Rosenfeld and Kak (1976) discuss many of the designs that have been attempted.

The problem with these local measurements arises in the interpretation of their response: a strong response may be signaling the presence of the indicated feature; however, it may also be arising from a noise configuration. On the other hand, a weak response may derive from the proper configuration. Single-step descision procedures that correctly determine the presence of image features on the basis of one local measuremnt have

been notoriously difficult, if not impossible, to design.

The principle cause of this difficulty is the non-unique structure of the measurement operator (a specific response may arise from many different intensity configurations) coupled with the presence of noise in the image. The situation is equivalent to that of controlling a system in the presence of noise which, control theory has shown, requires feedback (Zucker, 1977; Astrom, 1970). One possibility for feedback is from higher-level goals or expectations (Shirai, 1975; Freuder, 1976); however, these systems, with their inseparable mixture of domain-specific and general-purpose knowledge, are not extensible to other, less-specific, situations.

A second, less restricting form of feedback can be obtained from the structure of the neighborhood surrounding the feature operator. Two such neighborhoods are immediately evident: the surrounding intensity values and the neighboring feature-detector responses. Local detectors of various sizes can be used to capture the surrounding intensity structure, with decision rules based on all of their responses. Marr (1976), for example, has developed elaborate rules for parsing detector responses into various LINE, EDGE, EXTENDED-EDGE, and SPOT assertions, but his system must still refine these assertions by subsequent processing.

Because information is available outside of the immediately local neighborhood, another class of techniques, based on cluster analysis, have been proposed (Hanson and Riseman, 1978; Schacter, Davis and Rosenfeld, 1977). In these techniques, information about spatial relationships is sacrificed in the hope that the distribution of the quantities of responses will be sufficient for selecting regions homogeneous in the relevant feature. The problems with these techniques are well known, especially in terms of their one-dimensional counterpart, histogram peak selection. They include the difficulties inherent in specifying threshold selection criteria as well as determining which feature should be histogrammed. While they do work in some specialized circumstances (e.g., Prewitt, 1970) only large computational expenditures have yielded systems that perform reasonably in more general ones (Ohlander, 1975). As we show in Section 5, however, certain circumstances in which they should work can be specified precisely, although a cooperative algorithm is required to do this.

Pyramidal data structures (Klinger and Dyer, 1976; Tanimoto and Pavlidis, 1975; Hanson and Riseman, 1978; and Levine and Leemet, 1976) have been suggested as a data structure in which feedback between neighboring feature-detector responses can be accomplished efficiently. However, in a sense pyramids define the allowable computations in terms of a specific architecture, rather then determining the relevant computations and then fitting an efficient architecture to them. Thus while they do work well for some problems, more general interactions are necessary for the proper feedback of information and the iterated cycle of comparison and adjustment that this implies. More general iterative techniques for structuring this interaction are described in the next section.

3. RELAXATION LABELING PROCESSES

The low-level vision problem, as we have been posing it, is the computation of a description of the information contained in the intensity array that explicitly represents its useful content. This description can be formed in terms of assertions attached to various pictorial positions, and it is these assertions that will serve as input to later stages of processing. The discussion in the previous section reviewed the difficulties involved in attempting to do this solely on the basis of individual feature detector responses, but it also began to point toward the observation which motivated the techniques that we shall now describe. The relevant observation is that the context in which every feature detector is situated can be used to disambiguate its response. It leads to networks of local processes, called relaxation labeling processes, that cooperate and compete with one another until locally consistent descriptions are obtained.

Relaxation labeling processes (RLP's) attempt to reduce ambiguities in the set of labels attached to nodes in a graph (Zucker, 1976). For the low-level vision problem, the nodes in this graph could indicate pictorial positions, with the arcs connecting neighboring positions. The labels would indicate the structural assertions, with the necessary feedback provided through compatibility relations.

These compatibilities allow each label set to be adjusted in a direction that makes it more consistent with all of its neighboring label sets. The adjustment is accomplished by manipulating certainty factors attached to each label. These certainty factors are important for low-level vision applications because of the continuous nature of much of the knowledge being represented and because they provide an ordering over the possible labels for each node.

The specification of an RLP requires a specification of the graph structure, the set of possible labels, the compatibility functions, and the initial labeling for each node. Denoting the set of labels for node i by Λ_i, and letting $p_i(\lambda)$ indicate the certainty (or, loosely, the probability) that label λ is correct for node i, it is updated according to the rule:

$$p_i^{K+1}(\lambda) = \frac{p_i^K(\lambda)\ [1+q_i^K(\lambda)]}{\sum\limits_{\lambda\epsilon\Lambda_i} p_i^K(\lambda)\ [1+q_i^K(\lambda)]} \qquad (1)$$

where

$$q_i^K(\lambda) = \sum_{j\epsilon Neigh(i)} c_{ij} \sum_{\lambda'\epsilon\Lambda_j} r_{ij}(\lambda,\lambda')\ p_j^K(\lambda')$$

and

$$\sum_j c_{ij} = 1 \ .$$

The c_{ij} terms weight the total influence that a particular neighbor j can have on i; for uniform influence, $c_{ij} = 1/M$, when node i has M neighbors, $j = 1,2,\ldots,M$. The $r_{ij}(\lambda,\lambda')$ terms indicate the compatibility that label λ' on node j has with label λ on i. They take values in the range [-1,1], where -1 indicates perfect incompatibility and +1 indicates perfect compatibility. Since they need not be symmetrical, it is often convenient to think of them as functions of the conditional probabilities $p_{ij}(\lambda|\lambda')$. Rule (1) is evaluated in parallel over each label on each node, and then iterated until a fixed point is achieved (for a study of these fixed points, see Zucker, Krishnamurthy, and Haar, 1978). In this way the local context is used to iteratively approach each local labeling decision.

4. LABELING ORIENTED LINE SEGMENTS

Our first application of RLP's returns to the kind of problem discussed in Section 2: interpreting the response of local feature detectors. The specific problem that we shall consider is that of asserting the presence (or absence) of unit LINE segments (Zucker, Hummel, and Rosenfeld, 1977). The raw feature detector responses are supplemented by a model for good continuation of lines, and it is this model that structures the local consistency-based feedback.

The label set, i.e., the set of possible assertions, is

$$\Lambda_i = \begin{cases} \text{LINE (OR 1)} \\ \text{LINE (OR 2)} \\ \vdots \\ \text{LINE (OR 8)} \\ \text{NO-LINE} \end{cases}$$

Every point in the image corresponds to a node to be labeled, and each node is connected to its eight nearest neighbors. Since we will eventually want to group these LINE segments into long lines and curves, orientation is represented explicitly in eight quantized steps. Also, the local line detectors are evaluated at eight orientations.

Each of the nine labels is associated with every node, and their initial certainty factors are obtained by scaling the feature detector responses into [0,1]. These initial certainty values represent one source of information for the RLP. Establishing them effectively translates the intensity-encoded information in the image into the initial state for an orientation-based RLP. The rules for updating certainties on the basis of orientation are derived from the second information source, the compatibility functions, which take the form (for good continuation between LINE orientations):

$$r_{ij}(\lambda,\lambda') = \cos\,[\alpha_\lambda-\beta]\,\cos\,(\omega[\alpha_{\lambda'}-\beta]) \qquad (2)$$

where α_λ is the angle (i.e., orientation) of label λ, $\alpha_{\lambda'}$ is the angle of λ', and β is the angle an imaginary vector from i to j makes with the standard reference. (ω will be discussed shortly). The compatibility between LINE and NO-LINE labels also varies sinusoidally:

$$r_{ij}(\lambda,\lambda') = -\cos\,(2[\alpha_\lambda-\beta])$$

where α_λ and β are defined as above. The compatibility between two NO-LINE labels is 1.0.

The ω term in (2) is necessary for a more subtle reason. In general, the label set, graph structure, updating rule, and compatibility functions may define an RLP with a bias toward particular labelings. While this may reflect the relevant semantics in some domains, it is inappropriate for a general-purpose line process. Zucker and Mohammed (1978) have shown that if:

$$\sum_j c_{ij} \sum_{\lambda'} r_{ij}(\lambda,\lambda') = \text{CONST},$$

then a homogeneous RLP, such as the one we are describing here, will not contain a bias for certain labels. The ω terms in (2) are adjusted to satisfy this condition.

Zucker et al. (1977) contains examples of an RLP very similar to this one successfully labeling the lines and curves in several satellite images. To show the empirical limitations of such a process, Fig. 1 contains an example of this RLP operating on a very noisy synthetic image. Note that the diagonal line in this image is destroyed, and that the short horizontal line is joined to the vertical one. A more powerful hierarchical system, which handles this image properly, will be described in Section 6.

5. RELAXATION AND HISTOGRAM PEAK SELECTION

If we assume that the compatibilities are related by a linear function of the underlying conditional probabilities, i.e., if

$$p_{ij}(\lambda|\lambda') = \frac{[r_{ij}(\lambda,\lambda')+1] + c_1}{c_2}, \; c_1 \text{ and } c_2 \text{ constants}, \qquad (3)$$

or if we define our RLP directly in terms of conditional probabilities, then we can obtain an updating rule analagous to (1) by substituting (3) into (1):

$$p_i^{K+1}(\lambda) = \frac{p_i^K(\lambda)\,\{\sum_j c_{ij} \sum_{\lambda'} p_{ij}(\lambda|\lambda')\, p_j^K(\lambda')\}}{\sum_\lambda p_i^K(\lambda)\,\{\sum_j c_{ij} \sum_{\lambda'} p_{ij}(\lambda|\lambda')\, p_j^K(\lambda')\}} \qquad (4)$$

A simple analysis of this new rule reveals the connection between relaxation and histogram peak selection. In particular, if we interpret $p_j^0(\lambda')$ as the probability that label λ' is correct for node j given the local measurements on j, then the inner term

$$\sum_{\lambda'} p_{ij}(\lambda|\lambda')\, p_j^0(\lambda') = p_i(\lambda|E_j)$$

is a formally correct estimate of the probability that λ is correct for i (assuming that there are only two nodes of interest in the process, i and j). Then, if this estimate $p_i(\lambda|E_j)$ were perfect,

i.e., if $p_i(\lambda|E_j) = p_i(\lambda)$, rule (4) would reduce to:

$$p_i^{K+1}(\lambda) = \frac{[p_i^K(\lambda)]^2}{\sum_\lambda [p_i^K(\lambda)]^2} . \quad (5)$$

This rule clearly terminates with

$$\lim_{K\to\infty} p_i^K(\lambda) = \begin{cases} 1 & \max_{\lambda\varepsilon\Lambda_i} (p_i^0(\lambda)) \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

In other words, relaxation formally reduces to selecting the label whose initial probability was maximum, i.e., to histogram-peak selection.

The circumstances under which relaxation is equivalent to histogram peak selection can be generalized from those above. The essential requirement is that the neighborhood contributions be in the same order (i.e., consistent with) the current label orderings. More formally, if for each node $i = 1,2,\ldots,I$, an ordering over the labels $\lambda_\alpha \varepsilon \Lambda_i$ exists such that

$$p_i^K(\lambda_{\alpha_1}) > p_i^K(\lambda_{\alpha_2}) > \ldots > p_i^K(\lambda_{\alpha_N})$$

then we require (Zucker and Mohammed, 1978)

$$p_i^K(\lambda_{\alpha_1}|E) > p_i^K(\lambda_{\alpha_2}|E) > \ldots > p_i^K(\lambda_{\alpha_N}|E)$$

for the algorithm to behave as in (6). E indicates a conditioning on the total neighborhood evidence, i.e., the updating term:

$$p_i^K(\lambda|E) = \sum_j c_{ij} \sum_{\lambda'} p_{ij}(\lambda|\lambda') \; p_j^K(\lambda') .$$

It is clear from the results in the previous section (Figure 1) that relaxation is not just performing maxima selection, otherwise a number of initially strong, but incorrect, labels would have been established. Rather, the above analysis suggests that there are two essential components to the relaxation computation. The first is an adjustment toward local consistency, while the second selects the most consistent labeling. When consistency is present initially, then relaxation reduces to the second computation.

6. HIERARCHICAL RELAXATION SYSTEMS

The decomposition of the feature detection problem in Section 4 was a spatial one. That is, the establishment of LINE assertions was accomplished by a network of local processes which, when taken together, were attempting to establish a spatially-consistent pattern. There is a second necessary kind of consistency in vision systems, however, that derives from a different representational decomposition. This decomposition organizes representations into levels of abstraction, as we discussed in the introduction, with the consistency requirement now holding between descriptions at neighboring levels (providing they project onto the same set of spatial positions). For example, the high-level description of a chair must be consistent with the underlying line description, which, in turn, must be consistent with the underlying intensity distribution. Consistency of the first kind, i.e., spatial, will be referred to as being horizontal, while consistency of the second kind will be vertical (Zucker, 1978).

The same relaxation structure that we have been using for achieving horizontal consistency can also be used for achieving vertical consistency. The only differences are in the structure of the graph, which now spans levels, and in the compatibility functions. Letting λ denote a label attached to node i at one level of abstraction, and L denote a different label attached to node I at a neighboring level of abstraction, then the interaction between them will be governed by compatibility functions of the form $r_{iI}(\lambda,L)$. Furthermore, if there is a reasonably complete description at the level of I, then a second horizontal process (in addition to the one at the level of i) could be established there as well. With all of these processes running concurrently, information could propagate both horizontally and vertically to simultaneously disambiguate the descriptions at both levels (Figure 2). In other words, the local neighborhood over which consistency is evaluated has grown from a circle, which was the case for a single horizontal process, to a sphere.

As an example of a multi-level relaxation system, we shall build upon the line labeling RLP introduced in Section 4. That system was a horizontal RLP for labeling unit LINE assertions, and the next step is to group these unit LINEs together to form long lines and curves. The traditional method for accomplishing this is with a sequential tracking algorithm (e.g., Horn, 1974); however, such sequential methods are notoriously sensitive to noise. Rather, we shall formulate the grouping as a second RLP that attempts to join pairs of neighboring LINEs together. More precisely, if we define a link between every pair of neighboring pictorial positions that have moderately certain LINEs associated with them, then the RLP will attempt to label these links as either CONNECTED or NOT-CONNECTED (or, equivalently, as LINK or NO-LINK). Thus we have a hierarchical RLP system with two horizontal processes (one at the LINE level and one at the LINK level) and two vertical processes (one in which LINE labels influence LINK labels, and one in which LINK labels influence LINE labels). The system of descriptions is hierarchical because there is a contraction over position: each LINK label spans two LINE labels.

There are three additional groups of compatibility functions necessary for this hierarchical system: one for the horizontal link process and two for the vertical processes. Furthermore, in order to separate the information sources used at each level, the horizontal link process was designed to use the intensity information contained in the underlying pixels. (Recall that the horizontal LINE process used orientation information). More specifically, if $LINK_{ij}$ spans pixels i and j, and $LINK_{jk}$ spans pixels j and k, then the compatibility between them is defined to be (in

conditional probability terms):

$$p(LINK_{ij}|LINK_{jk}) = [1 - |Int(i) - Int(k)|] \times 0.4 + 0.3 \,, \quad (7)$$

where Int(i) and Int(k) are the normalized intensities at points i and k. Such normalization is necessary for the above expression not to be a function of the dynamic range of intensities for different systems, and consists simply in scaling all intensities into [0,1], with the mean intensity set to 0.5. The restriction of (7) to the range [.3,.7] is necessary so that intensity differences do not drive LINK labels to certainty in one step. Also, note that two LINKs are neighbors in the horizontal process if they span the same pixel and the angle of continuation between them is greater then or equal to 90°.

The first vertical RLP allows information about current LINK labels to influence the underlying LINE labels. This downward information flow is based upon orientation consistency, i.e.,

$$r_{iI}(LINE|LINK) = \cos(\omega\,[\alpha_{LINE} - \alpha_{LINK}]), \quad (8)$$

where ω is again set to meet the criterion in Section 4. To scale these compatibilities into the same form as (7), we set

$$p_{iI}(\lambda|L) = \frac{r_{iI}(\lambda,L) + 1}{\sum_{\lambda} [r_{iI}(\lambda,L)+1]} \,. \quad (9)$$

The second vertical RLP uses current LINE labels to influence LINK labels, with each link seeing only the two underlying LINE nodes as its neighbors. The compatibility function contains two terms combined as a product, the first relating orientations as in (8), and the second relating intensity information according to:

$$p_{Ii}(L|\lambda)_{INT} = [1 - |Int(i) - Int(k)|] \times 0.5 \,. \quad (10)$$

With this new form, when the intensity difference is small the intensity contribution approaches 0.5, leaving the consistency determination to the orientation component.

In order to take advantage of all of the information sources simultaneously, without possibly introducing oscillatory behaviour between them, it is necessary that all processes are run concurrently (Zucker, 1978). To accomplish this we used a modified form of the updating rule (1):

$$p_i^{K+1}(\lambda) = \frac{p_i^K(\lambda) \prod_j [\sum_{\lambda'} p_{ij}(\lambda|\lambda')\, p_j^K(\lambda')]}{\sum_{\lambda} p_i^K(\lambda) \prod_j [\sum_{\lambda'} p_{ij}(\lambda|\lambda')\, p_j^K(\lambda')]} \quad (11)$$

This new form is obtained from the approximations:

$$1 + a \sum_j s_j \simeq \prod_j (1 + a s_j) \simeq \prod_j (1 + s_j)^a$$

where

$$s_j = \sum_{\lambda'} r_{ij}(\lambda,\lambda')\, p_j^K(\lambda')$$

and

$a = c_{ij} = \frac{1}{J}$. (J is a variable denoting the degree of node i).

While there are many advantages to this form of the updating rule, such as an invariance to linear errors in heuristically chosen compatibilities (Zucker and Mohammed, 1978), its principle advantage here is that all node and label dependencies are grouped into a single term, $p_{ij}(\lambda|\lambda')$, rather than into separate terms for node and label dependencies, as in (1). Thus the different processes can now be combined as a simple product

$$p_i^{K+1}(\lambda) = \frac{p_i^K(\lambda) \cdot Q_H(\lambda) \cdot Q_V(\lambda)}{\sum_{\lambda} p_i^K(\lambda) \cdot Q_H(\lambda) \cdot Q_V(\lambda)} \quad (12)$$

in which the horizontal and vertical contributions are:

$$Q_H(\lambda) = \{\prod_{j \in HN(i)} \sum_{\lambda'} p_{ij}(\lambda|\lambda')\, p_j^K(\lambda')\}$$

$$Q_V(\lambda) = \prod_{I \in VN(i)} \sum_{L'} p_{iI}(\lambda|L')\, p_I^K(L')\} \,.$$

The result of running this hierarchical system on the noisy image in Fig. 1 is shown in Fig. 3. Note that now the image is labeled properly. Thus the hierarchical system, using intensity and orientation information, is more robust in the presence of extreme noise than the pure horizontal process.

7. MODELS FOR COMPATIBILITY FUNCTIONS

The emphasis in this paper has been largely on the structure of algorithms for computing visual descriptions, rather than on the content of visual representations. Nevertheless, it was necessary to specify a particular vision problem, that of line and curve labeling, in order to define processes completely enough for concrete study. This specification defined the symbolic structure of the RLPs, i.e., the label sets, the neighborhood relations, and the compatibility functions. In this section we consider a more general form for compatibility functions, those that are functions of the image, as well as some of the potential foundations from which compatibilities may be derivable that would require them.

The label sets for RLPs define the space of representations over which they are meaningful, such as the space of lines and curves. The internal structure of this space is reflected in (or defined by) the compatibility functions. Thus the curves that we considered had two structural components: one that followed a good continuation of orientation and one that followed a good continuation of intensity. Each of these components was complete in itself, and both were mutually consistent through the vertical compatibility functions. Each defined a stable RLP that extended the other through cooperation and competition. However, they were constant for all images, thus requiring that the image-specific data enter the process through the initial certainty factors.

A more general situation is to allow the specification of the internal structure of the representational space to be variable, perhaps depending on the image or a different knowledge source, rather than a constant. Then the compatibility functions would also be variable and the precise notion of consistency would be data (or

alternate knowledge-source) dependent.

To study one way in which image data could enter the compatibility functions, we developed an RLP for labeling INTERIOR, EDGE, and NOISE points in noisy images (Zucker and Leclerc, 1978; for an EDGE labeling process more like the LINE process that we discussed, see Hanson & Riseman, 1978). The design philosophy behind the process is that neighboring INTERIOR points should not have large intensity differences between them, that EDGEs should follow intensity differences, and that NOISE points should correspond to isolated intensity differences. Thus the compatibility functions are image dependent, and they were implemented as simple functions of neighboring intensity differences. For example, an INTERIOR label would support a neighboring INTERIOR label if their intensity difference were small (with respect to the average intensity), but would detract support if the difference were large. An example of this process is shown in Figure 4. Note that, since sufficient image information is in the compatibilities, the initial certainty factors can all be set to uniform values. Thus both the a priori and the a posteriori data enter in the same manner.

There are certainly many other knowledge sources necessary for the specification of representational domains that are richer than those considered here. These sources could fit together in a cooperative environment much like the one described in Section 6, perhaps augmented with adjustable compatibilities. Other sources that have been shown to admit this kind of representation include subjective contours (Ullman, 1976), stereo fusion (Marr and Poggio, 1977), and surface orientation (Woodham, 1977; Horn, 1974; Mackworth, 1973). Furthermore, Barrow and Tenenbaum (1978) have conjectured that many intrinsic scene characteristics, such as surface reflectance, orientation, distance, and illumination could be computed in a similar fashion. These studies suggest the exciting possibility that detailed models for the structure of the physical world could be developed which could then be used to derive the compatibility functions exactly. Such compatibilities would certainly have elaborate interdependency structures.

8. CONCLUSIONS

While the processes of low-level vision have been studied a great deal, these studies have tended to be disparate and problem-dependent. There are universal themes, however, and these themes include the use of various forms of problem decomposition to deal with complexity and the use of consistency to deal with ambiguity and noise. Taken together, these themes imply that low-level vision systems could be modeled as networks of local processes whose intercommunications are governed by local consistency relations.

Relaxation labeling processes embody this kind of structure, and several of their applications in low-level vision were described in this paper. Two of these indicated different representational decompositions, one across spatial positions at a single level of abstraction, and the other across levels of abstraction but over a single projected spatial position. Together they describe a type of canonical structure for representations in which the abstract flow of information, both horizontally and vertically, can be studied.

When a representational system admits such an organization, or when it forms a hierarchy, then the decomposition suggests that local neighborhoods should be viewed as spheres, encompassing both neighboring spatial positions and neighboring descriptions at adjacent levels of abstraction. Then consistency can be achieved both horizontally and vertically with information flowing to supplement partial results everywhere.

The formal analysis that we did revealed that relaxation, in addition to achieving local consistency, was also computing local maxima. This suggests an important difference between continuous relaxation processes, such as the one considered here, and discrete ones: continuous relaxation orders consistent hypotheses, while discrete processes do not. In general, however, understanding the precise computation that a distributed system such as relaxation is performing is extremely difficult, and much remains to be done. This increased computational understanding, coupled with a better understanding of the representation of visual information, should lead to the development of more functional vision systems.

ACKNOWLEDGEMENTS

This research was supported by NRC Grant A4470. I would like to thank Y. Leclerc and J. Mohammed for their help in developing the relaxation processes shown, and J. Petley for her help in preparing this paper.

REFERENCES

1. Astrom, K., An Introduction to Stochastic Control Theory, Academic Press, New York, 1970.

2. Barrow, H., and Tenenbaum, J.M., Recovering intrinsic scene characteristics from images, in E. Riseman and A. Hanson (eds.), Computer Vision Systems, Academic Press, New York, 1978.

3. Freuder, E., A computer system for visual recognition using active knowledge, TR-345, Artificial Intelligence Lab., M.I.T., 1976.

4. Hanson, A., and Riseman, E., Segmentation of natural scenes, in E. Riseman and A. Hanson (eds.), Computer Vision Systems, Academic Press, New York, 1978.

5. Horn, B.K.P., The Binford-Horn line finder, A.I. Memo 285, Artificial Intelligence Lab., M.I.T., 1973.

6. Horn, B.K.P., Understanding image intensities, Artificial Intelligence, Vol. 8, 1977, in press.

7. Klinger, A., and Dyer, C., Experiments on picture representation using regular decomposition, Computer Graphics and Image Processing, 5, 68-105, 1976.

8. Levine, M.D., and Leemet, J., A method for non-purposive picture segmentation, Proc. Third Int. Joint Conf. Pattern Recognition, 1976, 494-498.

9. Mackworth, A.K., Interpreting pictures of polyhedral scenes, Artificial Intelligence, 4, 1973, 121-137.

10. Marr, D., Early processing of visual information, Philosophical Trans. Royal Society, London (Series B), 275, 1976, 483-524.

11. Marr, D., and Poggio, T., Cooperative computation of stereo disparity, Science, 194, 1977, 283-287.

12. Ohlander, R., Analysis of natural scenes, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, 1975.

13. Prewitt, J.M.S., Object enhancement and extraction, in B. Lipkin and A. Rosenfeld (eds.), Picture Processing and Psychopictorics, Academic Press, New York, 1970.

14. Rosenfeld, A., Hummel, R., and Zucker, S.W., Scene labeling by relaxation operations, IEEE Trans. Systems, Man, and Cybernetics, SMC-6, 1976, 420-433.

15. Rosenfeld, A., and Kak, A., Digital Picture Processing, Academic Press, New York, 1976.

16. Schacter, B., Davis, L., and Rosenfeld, A., Some experiments in image segmentation by clustering of local feature values, TR-510, Computer Science Center, University of Maryland, 1977.

17. Shirai, Y., A context-sensitive line finder for recognition of polyhedra, Artificial Intelligence, 4, 1973, 95-119.

18. Tanimoto, S., and Pavlidis, T., A hierarchical data structure for picture processing, Computer Graphics and Image Processing, 2, 1975, 104-119.

19. Ullman, S., Filling-in the gaps: The shape of subjective contours and a model for their generation, Biological Cybernetics, 1976.

20. Woodham, R., A cooperative algorithm for determining surface orientation from a single view, Proc. Fifth International Joint Conference on Artificial Intelligence, 1977, 635-641.

21. Zucker, S.W., Relaxation labeling and the reduction of local ambiguities, Proc. Third Int. Joint Conf. on Pattern Recognition, 1976; also in C.H. Chen (ed.), Pattern Recognition and Artificial Intelligence, Academic Press, New York, 1976.

22. Zucker, S.W., Production systems with feedback, in F. Hayes-Roth and D. Waterman (eds.), Pattern-Directed Inference Systems, Academic Press, New York, 1977, in press.

23. Zucker, S.W., Vertical and horizontal processes in low-level vision, in E. Riseman and A. Hanson, (eds.), Computer Vision Systems, Academic Press, New York, 1978.

24. Zucker, S.W., Hummel, R., and Rosenfeld, A., An application of relaxation labeling to line and curve enhancement, IEEE Trans. Computers, C-26, 1977, 393-403 and 922-929.

25. Zucker, S.W., Krishnamurthy, E.V., and Haar, R.L., Relaxation processes for scene labeling: Convergence, speed, and stability, IEEE Trans. Systems, Man, and Cybernetics, SMC-8, 1978, 41-48.

26. Zucker, S.W., and Leclerc, Y., Intensity clustering by relaxation, IEEE Computer Society Workshop on PR and AI, Princeton University, 1978.

27. Zucker, S.W., and Mohammed, J.L., Analysis of probabilistic relaxation labeling processes, IEEE Conf. on Pattern Recognition and Image Processing, Chicago, 1978; also Technical Report 78-3R, Dept. of Electrical Engineering, McGill University.

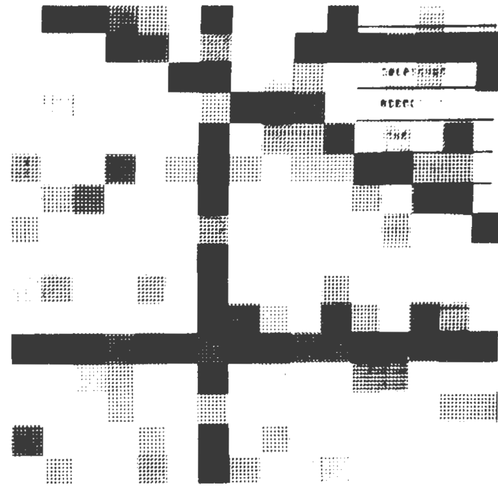
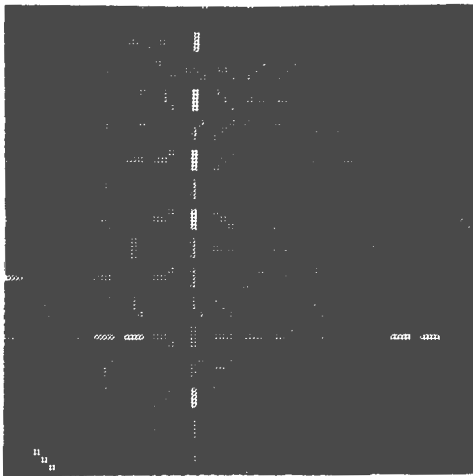
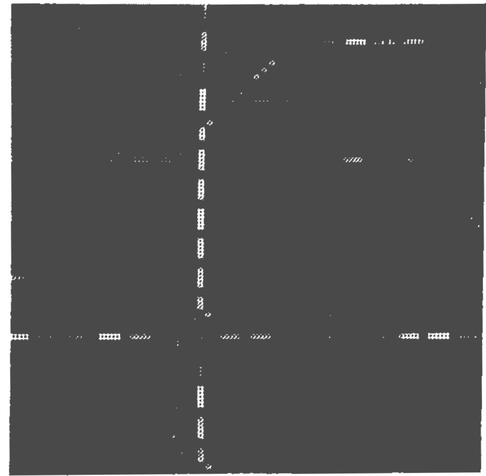


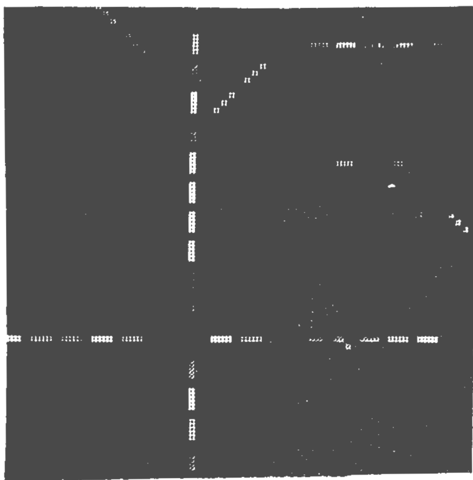
Fig. 1a: A (16x16) image containing three crossing lines and one short line, plus additive gaussian noise.



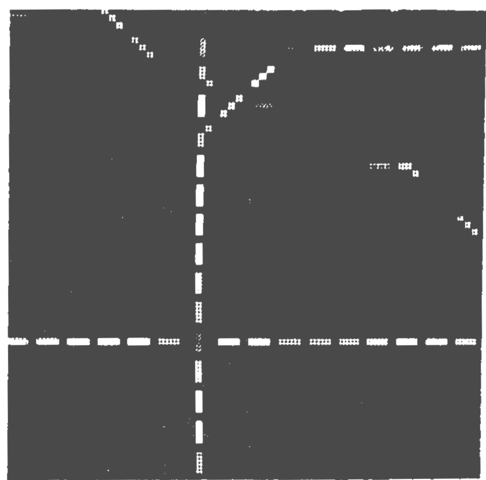
ITERATION 1



ITERATION 5



ITERATION 10



ITERATION 20

Fig. 1b: Horizontal LINE RLP operating on the image in Fig. 1a. The maximum LINE assertion is displayed at each position with an intensity proportional to its certainty.

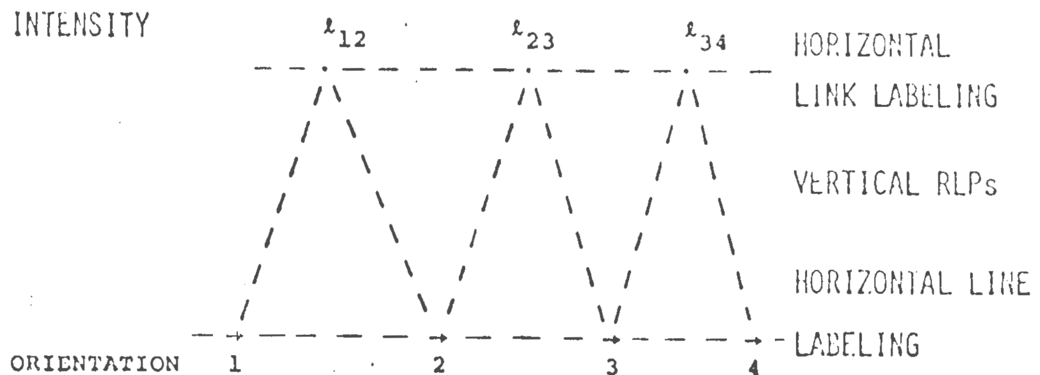
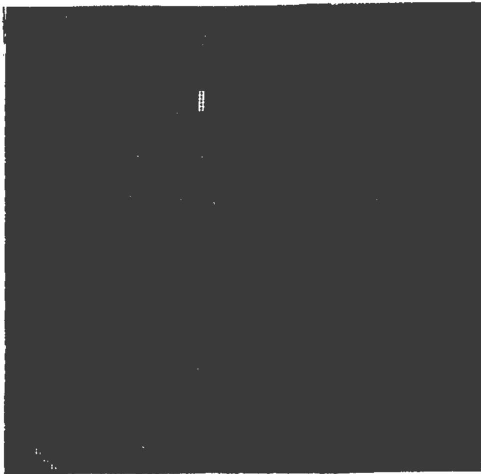
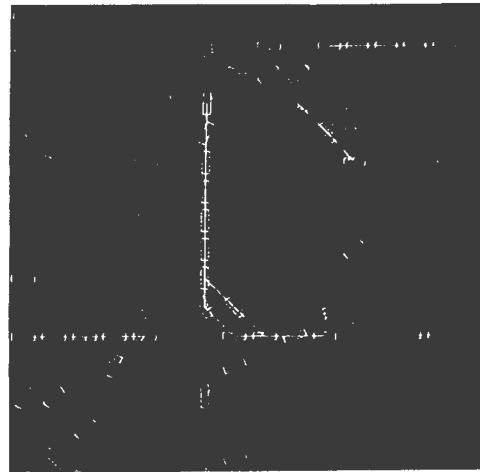


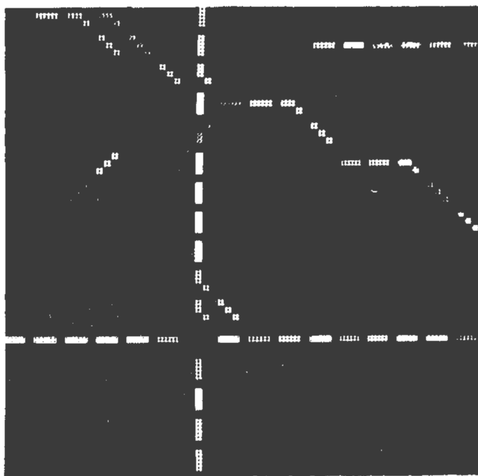
Fig.2: Information flows in the hierarchical system for line labeling and grouping.



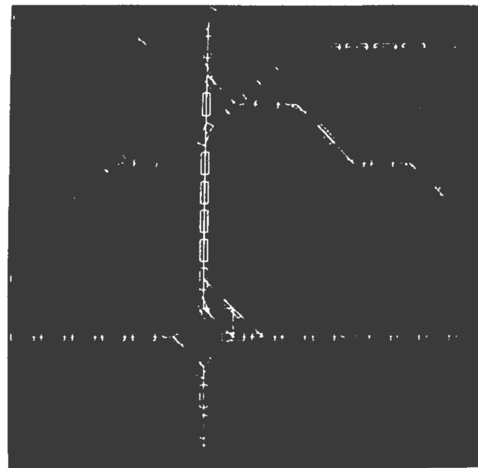
ITERATION 1, LINE LEVEL



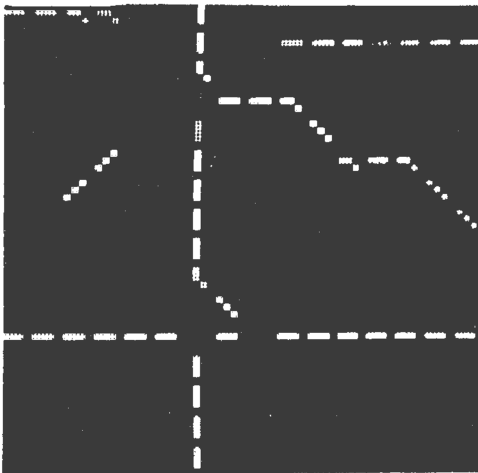
ITERATION 1, LINK LEVEL



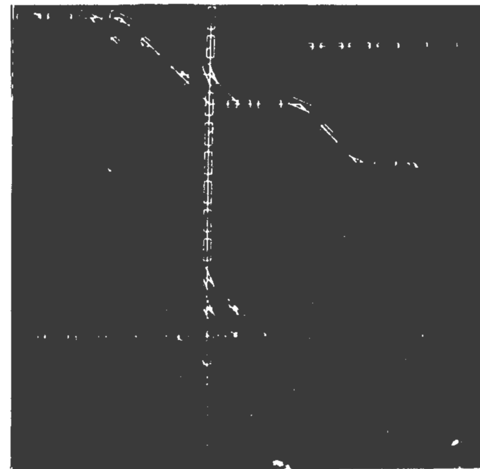
ITERATION 5, LINE LEVEL



ITERATION 5, LINK LEVEL

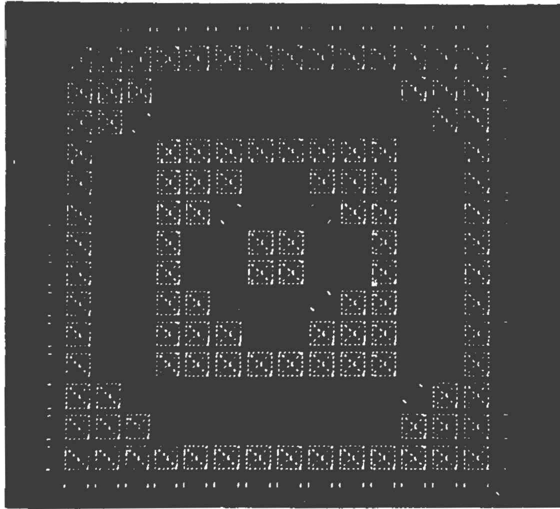


ITERATION 20, LINE LEVEL

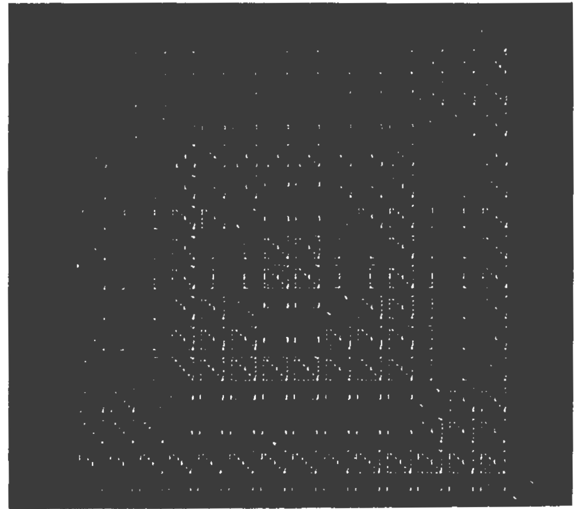


ITERATION 20, LINK LEVEL

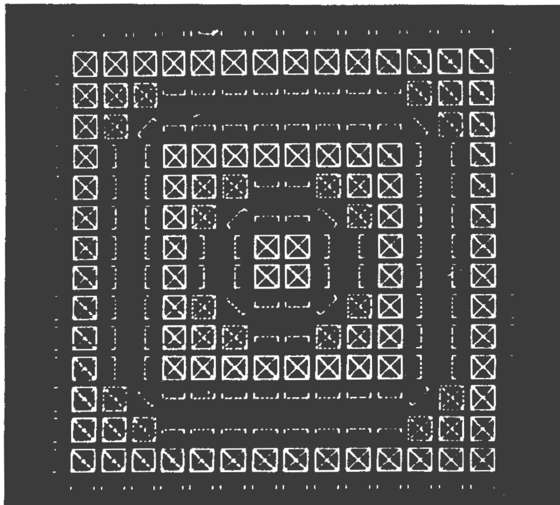
Fig. 3: Hierarchical system operating on the image in Fig. 1a. The LINK process is displayed together with the underlying LINE labels. Note that the uncertainty in the LINE labels has been removed by the LINK process.



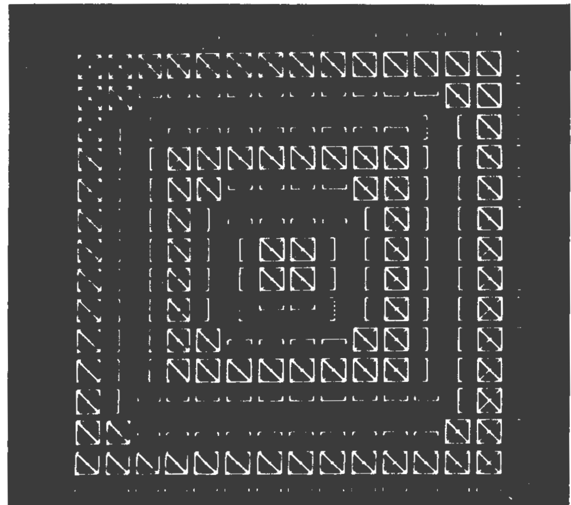
ITERATION 1



ITERATION 2



ITERATION 5



ITERATION 20

Fig. 4: An RLP for labeling EDGE, INTERIOR, and NOISE points. The INTERIOR labels are displayed as boxes, and the EDGE labels are displayed with the appropriate orientation. The original image consisted of three nested squares.

PHOTOMETERIC STEREO: A REFLECTANCE MAP TECHNIQUE FOR DETERMINING OBJECT RELIEF FROM IMAGE INTENSITY

Robert J. Woodham
Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Introduction

Work on image understanding has led to a need to model the imaging process. One aspect of this concerns the geometry of image projection. Less well understood is the radiometry of image formation. Relating the intensity values recorded in an image to object relief requires a model of the way surfaces reflect light.

A reflectance map is a convenient way to incorporate a fixed scene illumination, object photometry and imaging geometry into a single model that allows image intensity to be related directly to surface orientation. This relationship is not functional since surface orientation has two degrees of freedom and image intensity provides only one measurement. Local surface topography can not, in general, be determined by the intensity value recorded at a single image point. In order to determine object relief, additional information must be provided.

This observation has led to a novel technique called photometric stereo in which surface orientation is determined from two or more images. Traditional stereo techniques determine range by relating images of an object viewed from two directions. If the correspondence between picture elements is known, then distance to the object can be calculated by triangulation. Unfortunately, it is difficult to determine this correspondence. The idea of photometric stereo is to vary the direction of the incident illumination between successive views while holding the viewing direction constant. This provides enough information to determine surface orientation at each picture element. Since the imaging geometry does not change, the correspondence between picture elements is known a priori. This stereo technique is photometric because it uses the intensity values recorded at a single picture element, in successive views, rather than the relative positions of features.

The Reflectance Map

The fraction of light reflected by a surface in a given direction depends upon the optical properties of the object material, the surface microstructure and the spatial and spectral distribution and state of polarization of the incident light. A key photometric observation is:

No matter how complex the distribution of incident illumination, for most surfaces, the fraction of the incident light reflected in a particular direction depends only on the surface orientation.

The reflectance characteristics of an object material can be represented as a function $\phi(i,e,g)$ of the three angles i, e and g defined in figure 1. These are called, respectively, the *incident*, *emergent* and *phase* angles. The angles i, e and g are defined relative to the object surface. $\phi(i,e,g)$ determines the fraction of the incident light reflected per unit surface area, per unit solid angle, in the direction of the viewer.

If the equation of a surface is given explicitly as:

$$z = f(x,y)$$

then a surface normal is given by the vector:

$$[\partial f(x,y)/\partial x, \partial f(x,y)/\partial y, -1].$$

If parameters p and q are defined by:

$$p = \partial f(x,y)/\partial x$$
$$q = \partial f(x,y)/\partial y$$

then the surface normal can be written as $[p,q,-1]$. The quantity (p,q) is called the *gradient*, and *gradient space* is the two-dimensional space of all such points (p,q). Gradient space is a convenient way to represent surface orientation. It has been used in scene analysis [Mackworth 73]. In image analysis, it is used to relate the geometry of image projection to the radiometry of image formation [Horn 77].

Image forming systems perform a perspective transformation [figure 2(a)]. If the size of the objects in view is small compared to the viewing distance, then the perspective projection can be approximated as an orthographic projection [figure 2(b)]. Consider an image forming system that performs an orthographic projection. To standardize the imaging geometry, it is convenient to align the viewing direction with the negative z-axis. Assume appropriate scaling of the image plane so that object point (x,y,z) maps onto image point (u,v) where:

$$u = x$$
$$v = y.$$

An important simplification inherent in the assumption of an orthographic projection is that the viewing direction, and hence the phase angle g, is constant for all object points. Thus, for a standard light source and viewer geometry, the fraction of incident light reflected depends only on gradient coordinates p and q.

Further, suppose each object point receives the same incident illumination. Then, the amount of the incident light reflected in a particular direction depends only on the surface orientation. The assumption that the size of the objects is small compared to the viewing distance allows one to relate the amount of light reflected per unit solid angle in the direction of the viewer directly to image intensity. Thus, for the given imaging geometry, for a given distribution of incident illumination and a given object material, the image intensity corresponding to a surface point with gradient (p,q) is unique.

The *reflectance map* $R(p,q)$ determines image intensity as a function of p and q. A reflectance map captures the surface photometry of an object material for a particular light source, object surface and viewer geometry. It explicitly incorporates both the geometry and radiometry of image formation into a single model.

If the viewing direction and the direction of incident illumination are known, then expressions for $\cos(i)$, $\cos(e)$ and $\cos(g)$ can be derived in terms of gradient space coordinates p and q. Suppose vector $[p_s, q_s, -1]$ defines the direction of incident illumination. Then:

$$\cos(i) = \frac{1 + pp_s + qq_s}{\sqrt{1 + p_s^2 + q_s^2}\sqrt{1 + p^2 + q^2}}$$

$$\cos(e) = \frac{1}{\sqrt{1 + p^2 + q^2}}$$

$$\cos(g) = \frac{1}{\sqrt{1 + p_s^2 + q_s^2}}$$

These expressions allow one to transform an arbitrary surface photometric function $\phi(i,e,g)$ into a reflectance map function $R(p,q)$.

Reflectance maps can be determined empirically, derived from phenomenological models of surface reflectivity or derived from analytic models of surface microstructure. One simple, idealized model of surface reflectance is given by:

$$\phi_a(i,e,g) = \rho \cos(i)$$

This reflectance function corresponds to the phenomenological model of a "Lambertian" reflector which appears equally bright from all viewing directions. Here, ρ is an "albedo" factor and the cosine of the incident angle accounts for the foreshortening of the surface as seen from the source. The corresponding reflectance map is given by:

$$R_a(p,q) = \frac{\rho\,(1 + pp_s + qq_s)}{\sqrt{1 + p_s^2 + q_s^2}\sqrt{1 + p^2 + q^2}}$$

A second reflectance function, similar to that of materials in the maria of the moon and rocky planets, is given by:

$$\phi_b(i,e,g) = \rho \cos(i)/\cos(e)$$

This reflectance function corresponds to a surface which, as a perfect diffuser, reflects equal amounts of light in all directions. The cosine of the emergent angle accounts for the foreshortening of the surface as seen from the viewer. The corresponding reflectance map is given by:

$$R_b(p,q) = \frac{\rho\,(1 + pp_s + qq_s)}{\sqrt{1 + p_s^2 + q_s^2}}$$

Reflectance maps are independent of the shape of the objects being viewed. To emphasize that a reflectance map is not an image, it is convenient to present $R(p,q)$ as a series of "iso-brightness" contours in gradient space. Figure 3 and figure 4 illustrate the two simple reflectance maps $R_a(p,q)$ and $R_b(p,q)$ defined above.

Reflectance Map Techniques

Using the reflectance map, the basic equation describing the image-forming process can be written as:

$$I(x,y) = R(p,q) \tag{1}$$

This equation has been used in image analysis to explore the relationship between image intensity and object relief. Determining object relief from image intensity is difficult because (1) is underdetermined. It is one equation in the two unknowns p and q. In order to determine object relief, additional assumptions must be invoked.

Reflectance map techniques help make these assumptions explicit. For certain materials, such as the material of the maria of the moon, special properties of the reflectance map simplify the solution [Horn 75] [Horn 77]. Other methods to determine object relief from image intensity embody assumptions about surface curvature [Horn 77] [Woodham 77]. "Simplified" surfaces have been proposed for use in computer aided design [Huffman 75]. When properties of surface curvature are known a priori, these can be exploited in image analysis [Woodham 78]. This is useful, for example, in industrial inspection since there are often constraints on surface curvature imposed by the drafting techniques available for part design or the fabrication processes available for part manufacture. One purpose of these studies is to deepen our understanding of what can and can not be computed directly from image intensity.

Other reflectance map techniques use (1) directly to generate shaded images of surfaces. This has obvious utility in graphic applications including hill-shading for automated cartography [Horn 76] and video input for a flight simulator [Strat 78]. Synthesized imagery can be registered to real imagery to align images with surface models. This technique has been used to achieve precise alignment of LANDSAT imagery with digital terrain models [Horn & Bachman 77].

Photometric Stereo

Photometric stereo is a novel reflectance map technique which uses two or more images to solve (1) directly. The idea of photometric stereo is to vary the direction of incident illumination between successive views while holding the viewing direction constant. Suppose two images $I_1(x,y)$ and $I_2(x,y)$ are obtained by varying the direction of incident illumination. Since there has been no change in the imaging geometry, each picture element (x,y) in the two images corresponds to the same object point and hence to the same gradient (p,q). The effect of varying the direction of incident illumination is to change the reflectance map $R(p,q)$ that characterizes the imaging situation.

Let the reflectance maps corresponding to $I_1(x,y)$ and $I_2(x,y)$ be $R_1(p,q)$ and $R_2(p,q)$ respectively. The two views are characterized by two independent equations:

$$I_1(x,y) = R_1(p,q) \tag{2}$$

$$I_2(x,y) = R_2(p,q) \tag{3}$$

Two reflectance maps $R_1(p,q)$ and $R_2(p,q)$ are required. But, if the phase angle g is the same in both views (i.e., the illumination is simply rotated about the view vector), then the two reflectance maps are rotations of each other.

For reflectance characterized by $R_b(p,q)$ above, (2) and (3) are linear equations so that two views are sufficient to uniquely determine surface orientation at each image point (provided the directions of incident illumination are not collinear). Here, (2) and (3) suggest that a 90° angle between the directions of incident illumination would be optimal for photometric stereo.

In general, equations (2) and (3) are nonlinear so that more than one solution is possible. One idea would be to obtain a third image

$$I_3(x,y) = R_3(p,q) \tag{4}$$

to overdetermine the solution.

For reflectance characterized by $R_a(p,q)$ above, three views are sufficient to uniquely determine surface orientation at each image point [Horn 78]. Let $I = [I_1,I_2,I_3]'$ be the column vector of intensity values recorded at a point (x,y) in each of the three views (' denotes vector transpose). Further, let

$$n_1 = [n_{11},n_{12},n_{13}]$$
$$n_2 = [n_{21},n_{22},n_{23}]$$
$$n_3 = [n_{31},n_{32},n_{33}]$$

be unit vectors which point in the direction of the three positions of the incident illumination. Construct the matrix **N** where:

$$N = \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix}$$

Let $n = [n_1,n_2,n_3]'$ be the column vector corresponding to a unit surface normal at (x,y). Then,

$$I = \rho\, N\, n$$

so that,

$$\rho\, n = N^{-1}\, I$$

(provided the inverse N^{-1} exists). This inverse exists if and only if the three vectors n_1, n_2 and n_3 do not lie in a plane. In this case,

$$\rho = |N^{-1}\, I| \quad \text{and}$$

$$n = (1/\rho)N^{-1}\, I \tag{5}$$

Unfortunately, since the sun's path across the sky is planar, this simple solution does not apply to outdoor images taken at different times during the same day.

Equation (5) suggests that three mutually orthogonal directions of incident illumination would be optimal for Lambertian reflectance. In any stereo technique, however, there is some trade-off to acknowledge. In photometric stereo, choosing a larger phase angle g leads to more accurate solutions. At the same time, a larger phase angle causes a larger portion of gradient space to lie in the shadow region of one or more of the sources. A practical compromise is achieved by using four light sources and a relatively large phase angle. Solutions are accurate and most of gradient space lies in regions illuminated by at least three of the sources. Three image intensity measurements overdetermine the set of equations and establish a unique solution.

The images required for photometric stereo can be obtained by explicitly moving a single light source, by using multiple light sources calibrated with respect to each other or by rotating the object surface and imaging hardware together to simulate the effect of moving a single light source. The equivalent of photometric stereo can also be achieved in a single view by using multiple illuminations which can be separated by color.

Photometric stereo is fast. It has been developed as a practical scheme for environments in which the nature and position of the incident illumination is known or can be controlled. Initial computation is required to determine the reflectance map for a particular experimental situation. Once calibrated, however, photometric stereo can be reduced to simple table lookup and/or search operations.

Applications of Photometric Stereo

Photometric stereo can be used in two ways. For a given image point (x,y), equations (2) and (3) can be used to determine the corresponding gradient (p,q). Used in this way, photometric stereo is a general technique for determining surface orientation from image intensity. Figure 5 illustrates the reflectance map contours obtained from synthesized images of a sphere using a three source configuration.

For a given gradient (p,q), equations (2) and (3) can also be used to determine corresponding image points (x,y). Used in this way, photometric stereo is a general technique for determining points in an image whose corresponding object points have a particular surface orientation. Figure 6 illustrates the image intensity contours obtained from synthesized images of a sphere using a three source configuration.

This latter use of photometric stereo is appropriate for the so called industrial bin-of-parts problem. The location in an image of "key" object points is often sufficient to determine the position and orientation of a known object tossed onto a table or conveyor belt.

A particularly useful special case concerns object points whose surface normal directly faces the viewer. Such points form a unique class of image points whose intensity value is invariant under rotation of the incident illumination about the view vector (i.e., a change in direction of illumination which preserves the phase angle g). Thus, it is possible to locate such object points without explicitly determining the reflectance map R(p,q). Whatever the nature of the function R(p,q), the value of R(0,0) is not changed by rotation about the gradient origin. Figure 7 repeats the example given in figure 6 but for the case p = 0, q = 0.

Acknowledgements

This work 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 Project's Agency of the Department of Defence under Office of Naval Research contract N00014-75-C-0643.

References

Horn, B. K. P. (1975), "Obtaining Shape from Shading Information", in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, pp 115-155, 1975.

Horn, B. K. P. (1976), "Automatic Hill-Shading Using the Reflectance Map", (unpublished), 1976.

Horn, B. K. P. (1977), "Understanding Image Intensities", in *Artificial Intelligence*, Vol 8, pp 201-231, 1977.

Horn, B. K. P. & Brachman, B. L. (1977), "Using Synthetic Images to Register Real Images with Surface Models", AI Memo 437, M.I.T. AI Laboratory, August 1977.

Horn, B. K. P. (1978), "Three Source Photometry", (personal communication), 1978.

Huffman, D. A. (1975), "Curvature and Creases: A Primer on Paper", in *Proc. of Conf. on Computer Graphics, Pattern Recognition and Data Structures*, pp 360-370, 1975.

Mackworth, A. K. (1973), "Interpreting Pictures of Polyhedral Scenes", in *Artificial Intelligence*, Vol 4, pp 121-137, 1973.

Strat, T. M. (1978), "Shaded Perspective Images of Terrain", AI Memo 463, M.I.T. AI Laboratory, March 78.

Woodham, R. J. (1977), "A Cooperative Algorithm for Determining Surface Orientation from a Single View", in *Proceedings of IJCAI-77*, pp 635-641, August 1977.

Woodham. R. J. (1978), "Reflectance Map Techniques for Analyzing Surface Defects in Metal Castings", TR-457, M.I.T. AI Laboratory, (in press).

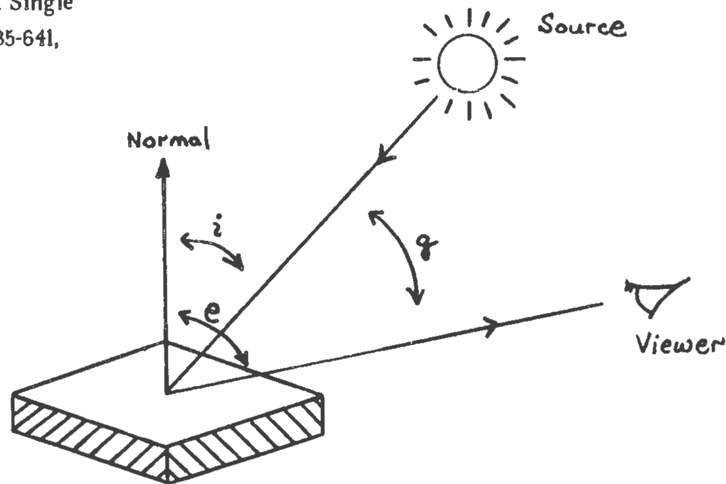


Figure 1 Defining the three photometric angles i, e and g. The incident angle i is the angle between the incident ray and the surface normal. The view angle e is the angle between the emergent ray and the surface normal. The phase angle g is the angle between the incident and emergent rays.

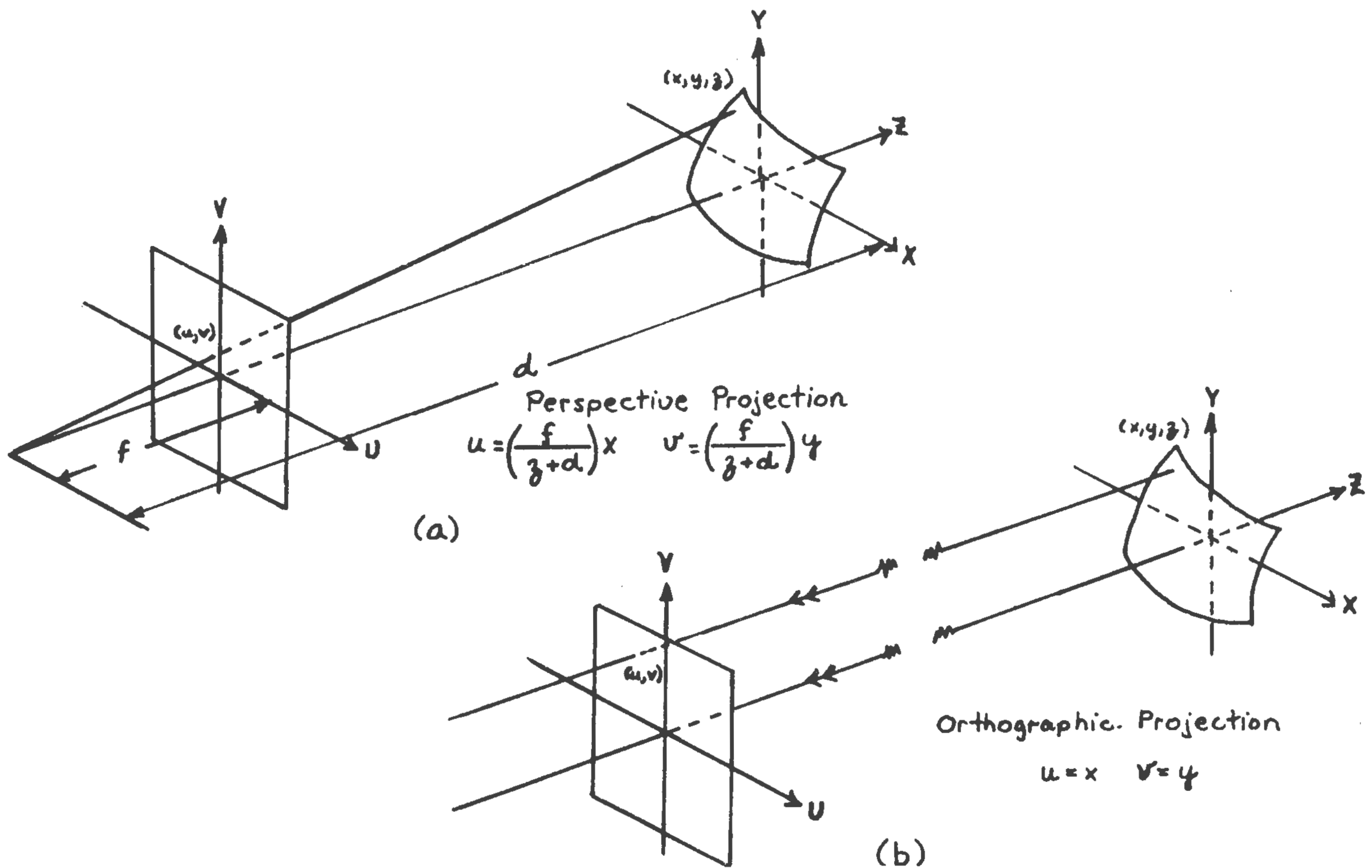


Figure 2 Characterizing image projections. Figure 2(a) illustrates the well-known perspective projection. [Note: to avoid image inversion, it is convenient to assume that the image plane lies in front of the lens rather than behind it.] For objects that are small relative to the viewing distance, the image projection can be modeled as the orthographic projection illustrated in figure 2(b). In an orthographic projection, the focal length f is infinite so that all rays from object to image are parallel.

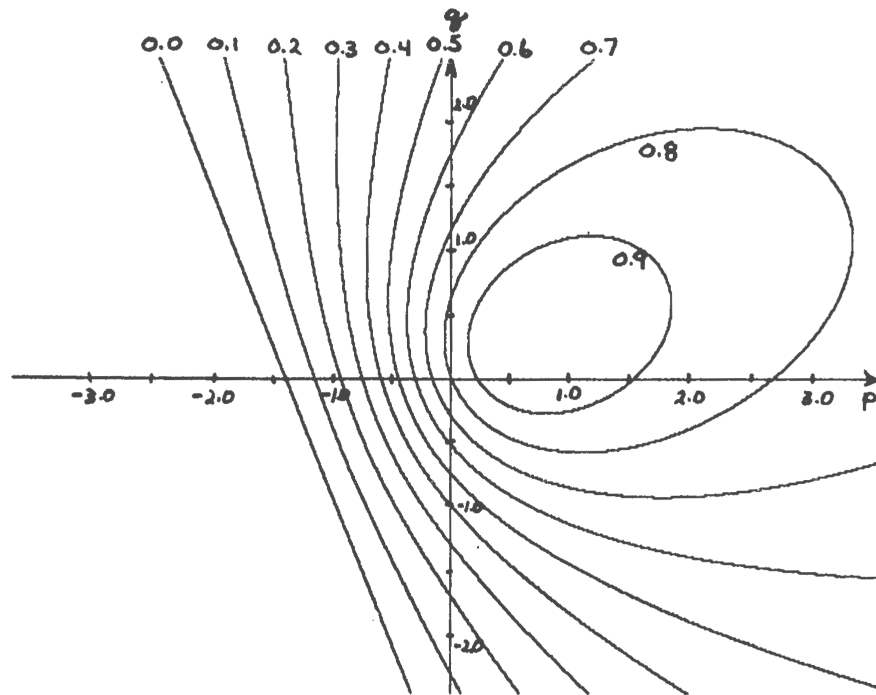


Figure 3 The reflectance map $R_a(p,q)$ for a Lambertian surface illuminated from gradient point $p_s = 0.7$ and $q_s = 0.3$ (with $\rho = 1.0$). The reflectance map is plotted as a series of contours (spaced 0.1 units apart).

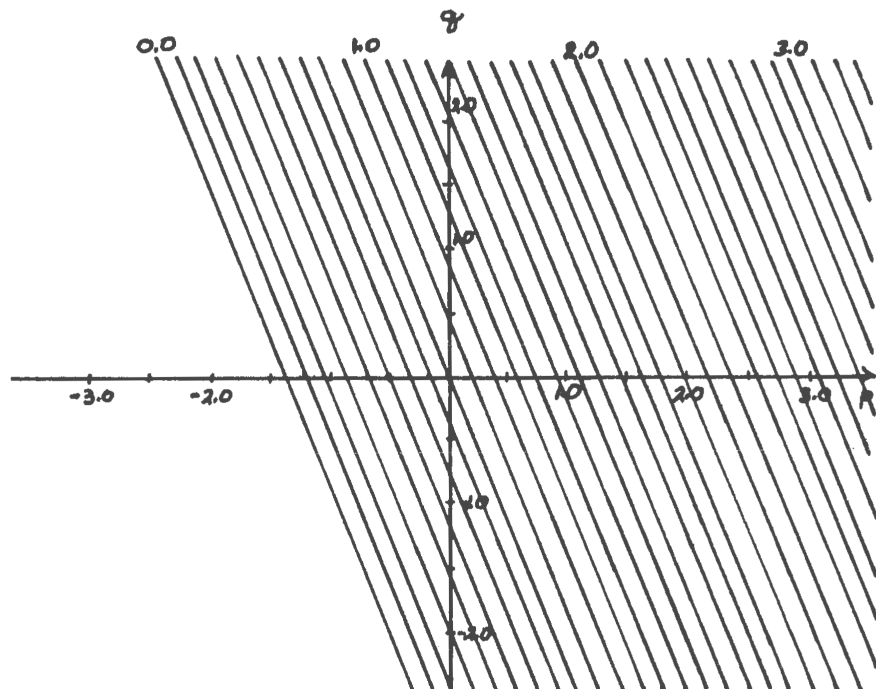


Figure 4 The reflectance map $R_b(p,q)$ for a perfect diffusing surface illuminated from gradient point $p_s = 0.7$ and $q_s = 0.3$ (with $\rho = 1.0$). The reflectance map is plotted as a series of contours (spaced 0.1 units apart).

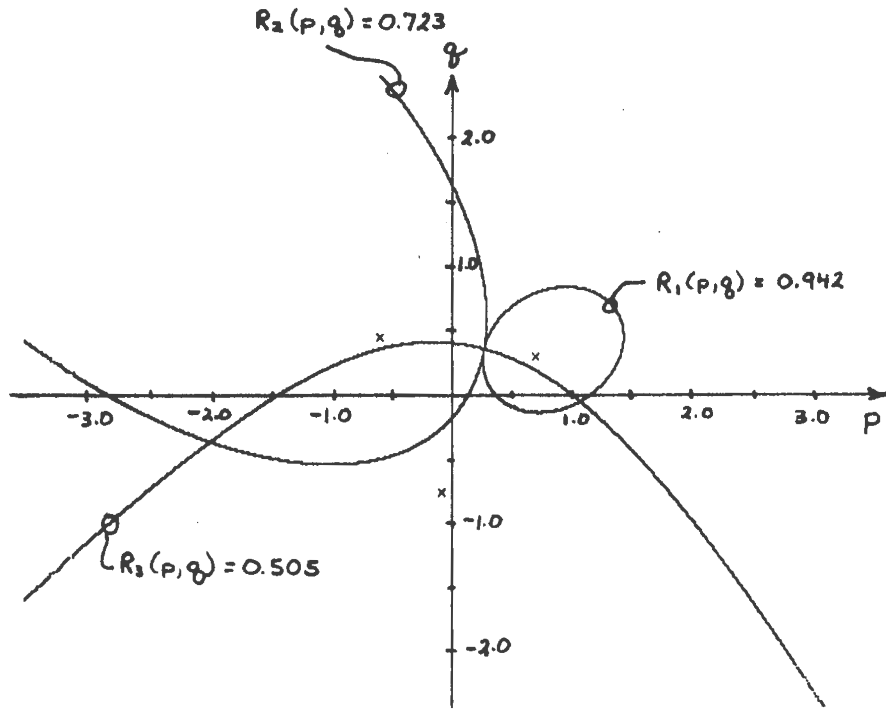


Figure 5 Determining the surface orientation (p,q) at a given image point (x,y). Three (superimposed) reflectance map contours are intersected where each contour corresponds to the intensity value at (x,y) obtained from three separate images (taken under the same imaging geometry but with different light source position). $I_1(x,y) = 0.942$, $I_2(x,y) = 0.723$ and $I_3(x,y) = 0.505$.

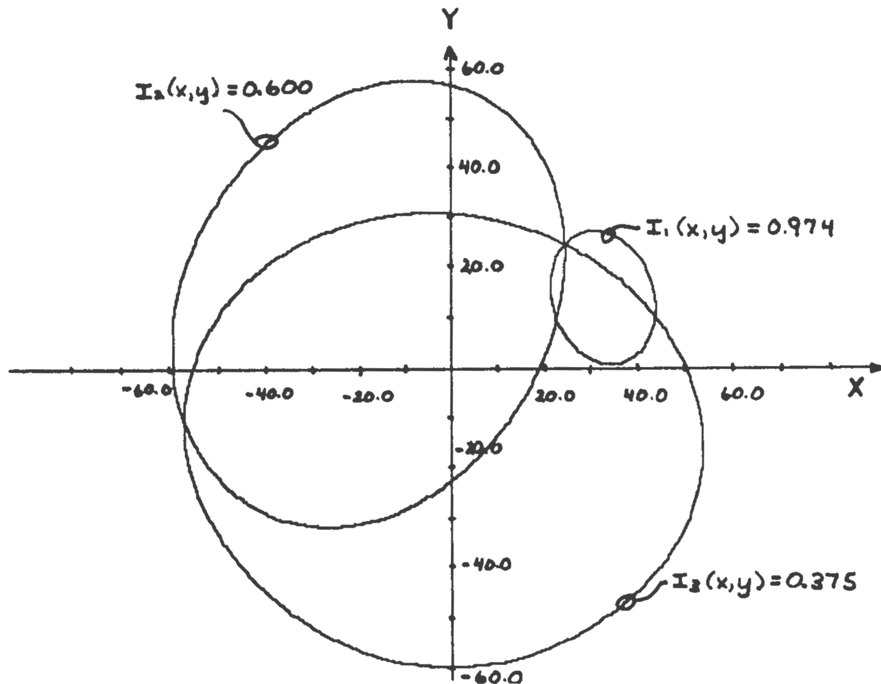


Figure 6 Determining image points (x,y) whose surface orientation is a given gradient (p,q). Three (superimposed) image intensity contours are intersected where each contour corresponds to the value at (p,q) obtained from three separate reflectance maps. (Each reflectance map characterizes the same imaging geometry but corresponds to a different light source position.) $R_1(p,q) = 0.974$, $R_2(p,q) = 0.600$ and $R_3(p,q) = 0.375$.

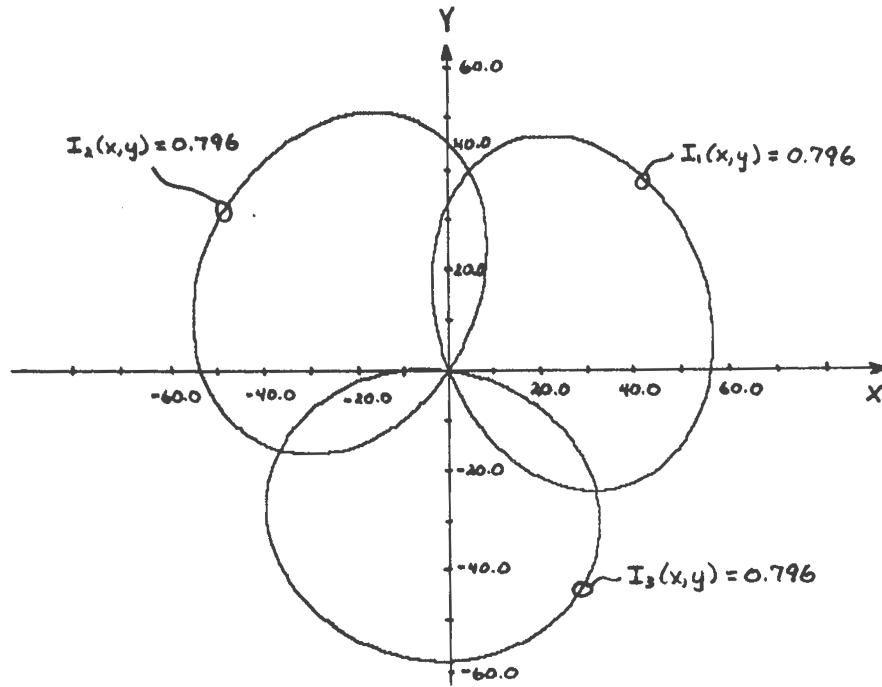


Figure 7 Determining image points whose surface normal directly faces the viewer. Three (superimposed) image intensity contours are intersected where each contour corresponds to the value at (0,0) obtained from three separate reflectance maps. (Each reflectance map characterizes the same imaging geometry but corresponds to a different light source position.) Note that the reflectance map value at (0,0) does not change with light source position (provided the phase angle g is held constant).

Image Segmentation and Interpretation Using A Knowledge Data Base

Samir I. Shaheen & Martin D. Levine

Department of Electrical Engineering
McGill University
Montreal, Quebec.

ABSTRACT

This paper overviews and discusses a model representation and control structure for a high level component of a computer vision system. The high level stage is characterized by the necessity to solve subproblems containing large search spaces, diverse and large sources of knowledge, and requiring non-deterministic (possibly in error) decisions made at various levels in the analysis. The high level stage adopts the concept of competition and cooperation as a basic paradigm for the system vision strategy. The system uses a relational database together with a relational algebraic sublanguage as an accessing mechanism to both the long and short term memories. Current knowledge about the particular picture under study is stored in the Short Term Memory which is designed as a communication channel between the different sources of knowledge of the system. The Long Term Memory contains all the relevent information (syntactic, semantic and pragmatic) about the class of scenes under analysis.

1. INTRODUCTION

This paper introduces a particular approach to the problem of computer vision, that is, the problem of segmentation and interpretation of two dimensional pictures which are projections of three-dimensional scenes. We have adopted a methodology in the design that stresses the development of modular processes, each of which is meant to deal with a major vision subproblem. These subtasks or modules should be able to communicate with each other in order to implement feedback and to cooperate in achieving an overall goal. As shown in figure 1, the system consists of many independent sources of knowledge which cooperate through a common data structure (short term memory) to achieve their own individual processing goals.

At the input, the color picture is segmented into regions containing pixels whose primary features (such as color and texture) are similar using a pyramidal region growing method described in (Levine 78). This method is based on a shared nearest neighbor clustering technique (Jarvis et al. 73) modified by a connectivity requirement and applied through a pyramidal data structure (Tanimoto et al. 75). It is relatively insensitive to absolute thresholding, and all pixels at every level of the pyramid are examined in parallel and not in a prespecified order. This low level processing stage of the analysis can be looked upon as a means of reducing the number of individual entities to be analyzed at the higher levels, thereby facilitating data handling and storage. Typically, it may result in a compression from the 128 x 128 pixels in the input (red, green and blue intensity planes) to the order of one to two hundred regions. The latter are then the input to the higher processing stages.

Beginning with these regions, Section 2 deals with the short term memory which embodies the current knowledge about the picture under consideration. Section 3 briefly discusses the relational algebraic sublanguage which is employed in conjunction with the relational database. This type of data structure is used to implement both the Short Term and Long Term Memories, the latter discussed in Section 4. Finally, Section 5 describes the high level processor which is concerned with the interpretation strategy for achieving the system goals.

2. SHORT TERM MEMORY (STM)

Due to the diversity of the various sources of knowledge needed to analyze complex natural scenes, as well as their dynamic interaction, the vision system must be modular in nature. This implies that each source of knowledge should be designed as a module that has access to the necessary information for achieving its processing tasks, as well as the results (often partial) from other

sources. Each module should be able to transmit its own conclusions regarding the scene under consideration to the other parts of the system. This allows for the possible parallel operation of the different sources as well as the ability to develop the system in stages. Because of these requirements, and motivated by the HEARSAY speech understanding system (Reddy et al., 73a, 73b, Lesser et al. 74), the Short Term Memory (STM) in figure 1 was introduced as a buffer or communication channel between the different sources of knowledge. This memory facilitates the desired interaction by allowing each source to transmit (broadcast) its own results, and to have access to both the information about the scene under study and all the deductions made by the other sources of knowledge. To achieve this, each new source needs only be aware of the overall STM structure and the communication language to and from the STM. Because of the modularity property, it is not affected by the previously installed sources or the removal or reformulation of some of them. We note that each source of knowledge may define a specific data structure (in our case relations) to provide and communicate specific information (e.g. the occlusion analyzer).

The STM is designed and implemented as a relational database (Shaheen 78) that maintains information about the scene such as the current segmented regions and their associated feature descriptions (see Relation 1). Structural relationships, as embodied by topological constraints among the regions, are also included (see Relations 2 and 3). In addition, associated with each region is a list of possible interpretations as deduced by the different modules of the system (see Relation 4). These lists of interpretations are ordered to reflect the current state of knowledge, as well as the focus of attention of the analysis.

The region features and the structural relationships prevailing among the regions are generated by the feature analyzer shown in figure 1. This module has access not only to the basic image planes (red, green, blue and texture description), but also to the pyramidal data structure generated by the low level processor (Nazif 78). Moreover, it is capable of generating new regions by merging other regions and then updating the appropriate STM relations.

The order of the regions in the STM is important as it controls the focus of attention of the high level processor. The latter scans the STM and examines the first region it finds satisfying specific conditions described in Section 5.2. The hypotheses associated with this region are tested and their corresponding confidences are updated to reflect the current overall knowledge about the scene. The STM regions are then reordered, thereby directing the system's attention to a different part (region) of the scene under study.

3. RELATIONAL ALGEBRAIC SUBLANGUAGE

A relational algebraic sublanguage has been defined and implemented to act as an accessing mechanism to be used by the different modules of the system (Shaheen 78). Such operations as JOIN, PROJECT and RESTRICT, together with UNION and INTERSECTION, are available to be used by the sytem modules or by the human operator during both the learning and experimentation stages. A detailed introduction to the relational database and the algebraic sublanguage can be found in (Date 74).

The usage of the relational algebraic sublanguage facilitates and guarantees the independence and modularity of the system modules. It also relieves the individual modules of the task of database administration and the problems associated with a particular physical storage device for the knowledge base. Each module can define temporary or permanent relations without interfering with the other modules. This is because all system relations are in the third normal form (Date 74), thereby providing an adequate level of data integrity. This data structure simplifies the task of the designers of the different modules as it pertains to the available computer resources. The only knowledge required by them are the particular relation formats and the relational algebraic sublanguage.

4. LONG TERM MEMORY (LTM)

The Long Term Memory is also a relational database that contains all of the relevent information (syntactic , semantic and pragmatic) about the class of scenes under analysis. Initially we have concentrated on pictures which can be analyzed on the basis of two dimensional models and do not require the third dimension which provides the depth information. A method of using monocular depth cues to compute the necessary three dimensional information is under study (Rosenberg 78). The data regarding the world models and their constituents are entered into the Long Term Memory by means of a learning phase . They are accumulated by computing (with the aid of the human operator) different object features over a set of representative images of the visual world under study. The attributes of each object (or part of an object) in the world model are

entered into the relation OBJECT ATTRIBUTES. By object (or part of an object) we do not only refer to entities that are generally quite well defined by their shape, such as a "door" or a "car", but also to items such as "sky" or "ground" whose boundaries need to be delineated in each image. LTM relations exist to inform the system modules, such as the hypothesis initialization module, of the distinguishable features of each object as well as the acceptable variations in these features.

Conceptual classes such as QUALIFYING, LOCATIONAL, JOINING, INTRUSION, DIVISION and CONTAINMENT as described in (Firshein 71), may be defined by a set of relations (see Relations 5 and 6). Contextual cues about the visual world under study are used by the different modules of the system to achieve their processing goals. As an example, consider the LOCALIZING conceptual class, or as it is often referred to the complexity predicate (Firshein 71), which is shown in Relation 6. It can be extremely powerful in limiting subsequent search after a particular object (or part of an object) has been recognized. Most probably it would be used in conjunction with information appearing in other relations such as, for example, constraints about spatial relationships (e.g. the fact that the "sky" may be adjacent and above the "roof" but cannot be adjacent to a "door"). These constraints are coded in a specified format and stored in different LTM relations.

The above LTM relations are employed by the system in directing the processing of the various sources of knowledge of the system. In general, they consist of tuples, where each tuple T is a conditional statement composed of zero or more condition elements (constraints) and zero or more action elements. The hypothesis tester uses a group of these relations to evaluate and update interpretation confidences of the region under consideration. We refer to such a condition-action group as a rule, and note that it includes information regarding syntactic, locational and relative position constraints. The latter specify and determine the compatibility of each hypothesis of a region Rx with the best available interpretation for the regions in its immediate neighborhood. For example, a relation is used to store spatial constraints which are matched against an input pattern. Here, matching implies that the system is able to verify the hypothesis that any two regions R1 and R2 with interpretations I1 and I2 are compatible (i.e. satisfies all of the constraints associated with the rules that contain I1 and I2 as adjacent objects). In case of a match (R1 and R2 are compatible), the hypothesis tester executes an action associated with the matched constraints, which tends to increase the confidence of the tested pattern. The hypothesis tester is described in more detail in Section 5.1.

Another set of relations in the LTM is used to direct the focus of attention of the system. In general, they specify an action or a set of actions to be excuted in certain situations. The situation-action pair, also termed a rule, is specified as a conditional statement in the form of :

$$S(C1...Ci...Cn1) \dashrightarrow (A1...Aj...Am1)$$

where Ci is the i-th condition element and Aj is the j-th action to be excuted provided $n1 \geq 0$ and $m1 > 0$.

The focus of attention module finds the first situation in the LTM that matches the current knowledge about the scene (STM knowledge) and excutes its associated actions. We note the importance of the order of the situation-action pairs in the LTM relations. Of course, other more intelligent search strategies for a matching situation may be introduced. Here we should also point out that default actions are excuted when no other situation is true. This is specified by having an action associated with a null situation (situation with zero conditions) as the last situation examined. The situation-action pairs are used to specify heuristic PLANS, activate other system modules or update the current knowledge about the scene. The following are some examples of these situations and their associated actions:

(i) If "door" was the best interpretation of the current region Rx under consideration with confidence Cx greater than some threshold, and region Rx contains some other region Ry, then the action of the system would be to increase the confidence of the hypothesis indicating that region Ry is a "doorknob".

(ii) If two adjacent regions have the same best interpretation (e.g. "sky") with confidence greater than some threshold, then the system's action would be to activate the merging module as well as the feature analyzer to update the appropriate relations.

Section 5.2 describes the system control structure in light of this set of rules and the effect on the focus of attention of the system.

A significant aspect of this approach is that the knowledge base (LTM) can be easily edited to change any of these relations (e.g. contextual cues, spatial constraints, condition-action and situation-action relations) thereby providing the system user w th a powerful experimentation tool for testing different strategies and approaches.

5. CONTROL STRUCTURE and INTERPRETATION STRATEGY

5.0 Initialization

The high level processing stage starts by generating a number of hypotheses for each region. The object hypothesizer is initially invoked to obtain a preliminary set of hypotheses (interpretations) for each of the regions in the STM. Using an exhaustive search, these are considered independently as candidates for matching with objects in the LTM in order to generate a set of all possible hypotheses about each of the regions. Use is made of various LTM relations that describe the distinguishable features of each object, as well as the acceptable variations in these features, to determine all the possible hypotheses about the region. Hypotheses (interpretations) for each region, together with their respective matching confidences, are entered in the STM in decreasing order with respect to their confidence. Moreover, the regions in the STM are also ordered using any feature (e.g. area, position or intensity) initially specified by the system user. This allows the user to direct the focus of attention of the system by specifying which region should be examined first by the system (e.g. the largest region or the brightest region). Starting with this region, the analysis proceeds using the hypothesis tester and focus of attention modules as will be described in Sections 5.1 and 5.2.

5.1. Hypothesis Tester

The high level processor employs the concepts of competition and cooperation as a basic paradigm for the system vision strategy (Arbib 75). The system uses the competition between the possible hypotheses for a particular region and the competition and cooperation between the immediate neighbors of the region in order to achieve a global compatible interpretation of all of the regions. Basically, it employs local information from the immediate neighborhood of a region to obtain a global understanding of the whole scene (Zucker 76, 78, Waltz 72).

Assume that region Rx is the region on the top of the STM. Let us define the following :

- Ix(k) is the k-th interpretation of regoin Rx where $1 \geq k \geq n$ (n is the number of active interpretations of region Rx);
- Cx(k) is the confidence of the k-th interpretation of Rx;
- Rj is the j-th immediately adjacent region to Rx where $1 \geq j \geq m$ (m is the number of adjacent regions);
- IRj is the current best interpretation of region Rj ;
- CRj is the confidence that region Rj is interpreted as object IRj;
- ARj represents the attributes of region Rj;
- SRj represents the structural relationships associated with region Rj;
- J(Rj,Rx) represents the adjacency of region Rj to region Rx, defined as the ratio of the common boundary of Rj and Rx divided by the total perimeter of Rx ($0 \geq J(Rj,Rx) \geq 1$).

As described in Section 4, the rules are of the condition-action type. First, the system considers the best interpretation of each of the neighboring regions of Rx (IRj), their confidences (CRj), the adjacent regions' attributes (ARj), and the structural relationships with respect to region Rx and to the other regions (SRj). Using these data, together with the attributes of region Rx, the hypothesis tester (figure 1) examines the compatibility of the interpretation of region Rx as object Ix(k) with its immediate adjacent regions' current best interpretaion IRj. This is accomplished by applying all the constraints associated with interpretations Ix(k) and IRj as adjacent objects. If all of these constraints are satisfied, the action of the system is to increase the confidence Cx(k) of region Rx as being object Ix(k), otherwise Cx(k) is decreased, as will be shown later. This operation is achieved by the hypothesis tester which tests the validity of of the different hypotheses about region Rx and updates them to reflect the influence of its immediate neighbors. The interpretation list of region Rx is then reordered on the basis of decreasing confidence. An interpretation of region Rx may be deactivated (not considered in subsequent examinations of the region) if it is found to be incompatible with the neighboring regions (e.g. its confidence is less than some threshold). At a certain point in the analysis, each region will be left with one active hypothesis which is the best possible compatible region with its adjacent

regions. If more than one interpretation exists, the system employs the interpretation with the highest confidence.

Analytically, the system examines each interpretation Ix(k) of region Rx and all the other scene knowledge as described above to calculate $\Delta Cx(k)$ as follows:

$$\Delta Cx(k) = \sum_{j=1}^{J=n} \delta_{jx} \cdot J(Rj, Rx) \cdot C_{Rj}$$

where δ_{jx} is set to 1 if the hypothesis tester is able to verify the compatability of regions Rx and Rj interpretations as objects Ix(k) and IRj using the constraints in the LTM relations and the current knowledge about the scene under study; otherwise δ_{jx} is set to -1.

The new unnormalized confidences of the different interpretations of region Rx are defined as:

$$\overset{*}{Cx}(k) = Cx(k) \cdot \left(1 + \frac{\Delta Cx(k)}{\sum_{k=1}^{k=n} \left|\Delta Cx(k)\right|}\right)$$

These are then normalized as follows:

$$Cx(k) = \frac{\overset{*}{Cx}(k)}{\sum_{k=1}^{k=n} \left|\overset{*}{Cx}(k)\right|}$$

If any of the interpreation confidences is found to be less than some threshold (i.e. Ix(k) is voted to be incompatible with its neighbors), interpretation Ix(k) of region Rx is deactivated. That is, it is not considered in the subsequent examinations of region Rx.

After the hypothesis tester updates the interpretation confidences of region Rx, the focus of attention module examines the current knowledge about the scene (i.e the STM knowledge with the now updated information about region RX) to direct the next processing stages and the course of action to be taken to achieve the overall processing goal of the system.

5.2. System Focus of Attention

The analysis described in Section 5.1 is always followed by the activation of the focus of attention module. As described in Section 4. this module operates within a control framework called recognize-act cycle. First it scans the set of situations specified in the LTM in attempt to find the situation that has all of its conditions satisfied with respect to the current knowledge in the STM. It then excutes all of the actions associated with the satisfied situation. In most cases the conditions are expressed in terms of the best new interpretation of region Rx (top region on the STM), its attributes, structural relationships and global knowledge about the scene. The actions may be one or more of the following:

i- Reordering of the regions in the STM to direct the system focus of attention to a different part of the image. This is achieved by moving the regions to be examined (i.e. the regions in that specific part) to the top of the STM.

ii- Updating some other region interpretaion using the LOCALIZING conceptual class (Relations 6) after a particular object has been recognized with some degree of confidence.

iii- Executing the safest merge, that is, finding if any of the adjacent regions has the same object interpretation with a confidence greater than some threshold and then merging these regions with region Rx. Merging regions will also activate the feature analyzer to update region boundaries and to calculate the new attributes of the region as well as its structural relationships to other regions already in the STM. This new information is then inserted into the appropriate STM relations.

iv- Implementating a PLAN which is stored as a procedure or matching another set of situation-action rules. This allows for a hierarchical structuring of the control.

v- Feeding back to the low level processing stages.

In case the STM has not been reordered by any of the actions defined above (i.e. action i was not triggered), the high level processor moves region Rx to the bottom of the STM. In addition, it places all the immediately adjacent regions of Rx at the top of the STM in an increasing order of confidence, thus allowing the least confident immediate neighbor of region Rx to be the first region to be examined next using the more confident interpretations of its surrounding regions.

To ensure the propagation of the interpretations of the region Rx and its associated confidences before the system re-examines Rx, a fixed length queue is introduced to maintain (remember) the

last Nq (the queue length) regions examined by the system. Hence region Rx will be inserted in the queue and the first element (region) of the queue will be deleted. The system then proceeds to consider the next region on the top of the STM, Rx'. This region is compared with the elements of the queue. If Rx' already exists in the queue, the next region in the STM is taken. The process is repeated until a region Rx' is selected from the STM which does not exist in the queue. This technique controls the frequency of examination of each region in the STM. It also allows each region's updated interpretations to affect the other regions' interpretations before they get updated. In other words, the effect of the identification of region Rx as object Y with a certain confidence will affect directly its adjacent neighbors Rj and we should allow this effect to propagate to the regions not in the immediate neighborhood of Rx before re-examining Rx. This also prevents incorrect local interpretations in part of the image from directing the system towards an incorrect global interpretation.

The length of the queue, Nq, will determine the size of the region neighborhood that will be affected by the local information before being influenced by the global understanding. As the length of the queue approaches the number of regions in the STM, the system will go through an analysis of most of the regions before returning to a particular region. On the other hand, if the queue length is zero, the system may (in certain situations) continue examining a set of regions in a small part of the image using only local information. The queue length is a focus of attention parameter determined by the user depending on the vision strategy being pursued.

This module also monitors the changes with time in the confidences of the various regions. Define ΔP to be :

$$\Delta P^{t+1} = \sum_{k=1}^{k=n} \left| C^{t+1}(k) - C^{t}(k) \right|$$

where

$Cx^{t+1}(k)$ is the updated k-th interpretation confidence of region Rx (region on top of STM).

$Cx^{t}(k)$ is the old interpretation confidence for the same interpretation of region Rx.

If ΔP is not equal to zero, the process continues as described above. But if ΔP is equal to zero during the last Nt region examinations (Nt is the number of regions that has more than one interpretaion), the process is stopped and each region is labelled with its current best interpretation.

The above process will continue until each region has a unique interpretation compatible with the global scene understanding. However, it can also be stopped at any stage, yielding the best set of interpretations for the different regions in the image at the time.

6. CONCLUSION

In summary, the previous sections have described the high level components of the computer vision system. Due to the diversity of the various sources of knowledge needed to analyze complex natural scenes, as well as their dynamic interaction, the modularity of the design has been emphasized. This also allows the incremental development of such a complex system. The Short Term Memory, which is implemented as a relational database, was introduced to work as a buffer between the different sources of knowledge. A relational algebraic sublanguage has been defined to serve as a commmunication language and accessing mechanism between the system modules.

The high level processor is designed as a knowledge driven system with two types of rules, condition-action and situation-action. These conditions or sets of conditions and their associated actions are stored in the relational database as part of the world model. They form the contextual cues, constraints, and PLANS used by the system in the segmentation and interpretation processes. The rules may be easily changed using the relational database editor which in turn, gives the user a very powerful tool for experimentation.

An important aspect of the operation of the high level processor is that of the focus of attention. This concept is employed to direct the analysis to various parts of the picture according to well defined criteria and parameters. In this way, it is possible to test different vision strategies.

The computer vision system described above embodies the paradigm of competition and cooperation. We have tested it with the problem of image segmentation and interpretation of color outdoor pictures. The preliminary experimental results have been encouraging and an extensive analysis of the performance of the system is presently underway.

ACKNOWLEDGEMENTS

This work was partially supported by the National Research Council of Canada under grant No. A4156 and the Department of Education, Province of Quebec. S. Shaheen was supported by a fellowship awarded by the Faculty of Graduate Studies and Research at McGill University. The authors would like to thank David Kashtan, Juhan Leemet and David Ting for their contributions to the research described in this paper.

REFERENCES

Arbib, M.A., Two Papers on Schemes and Frames, Coins Tech. Report 75C-9, Computer and Information Science, University of Massachusetts at Amherst, Oct. 1975.

Date, C.J., An Introduction to Database Systems, Addison- Wesley Publishing CO. Inc., Reading, Massachusetts, 1975.

Erman, L.D., Lesser, V.R., A Multi-level Organization for Problem Solving Using Many Diverse, Cooperating Sources of Knowledge, Proc. 4th IJCAI, Tbilisi, Georgia, Sept. 1975.

Firschein, O., Fischler, M.A., A Study in Descriptive Representation of Pictorial Data . 2IJCAI, London, England, Sept. 1-3,1971.

Jarvis, R. A., Patrick, E. A., Clustering Using a Similarity Measure Based on Shared Near Neighbour, IEEE Trans. on Computers, Vol.C-22, no.11,Nov. 1973.

Lesser, V.R., Fennell,R.D.,and Reddy, D.R., Organization of the HEARSAY II Speech Understanding System, Proc. IEEE Symp. Speech Recognition, Pittsburg, Pa.,1974. Reprinted in IEEE Trans. on Acoustics, Speech, and Signal Processing, ASSP-23,no. 1, Feb. 1975.

Levine, M. D., A Knowledge - Based Computer Vision System, in Computer Vision System, E. M. Riseman & A. R. Hanson (eds.), Academic Press, N.Y., 1978.

Levine, M. D., Region Analysis in Structure Computer Vision : An Approach to Machine Perception Based on Psychological Principles and Image Hierarchies. S. Tanimoto & A. Klinger (eds.), 1978.

Nazif,A., A Survey of Color. Boundary Information and Texture as Feature for Low Level Image Processing, Tech. Report No. 78-7R,Department of Electrical Engineering, McGill University. Montreal, Canada, 1978.

Reddy, D. R., Erman, L.D.,and Neely, R.B. (1973a), A Model and System for Machine Perception of Speech, IEEE Trans. on Audio and Electroacoustics, AU-21, no.3,1973.

Reddy, D.R., Erman, L.D., Fennell, R.D.,and Neelv. R.B. (1973b), The HEARSAY Speech Understanding System : An Example of the Recognition Process, Proc. 3IJCAI, Stanford, California, August 1973.

Rosenberg, D., Monocular Depth Perception for a Computer Vision System, M.Eng. Thesis, Dept. Electrical Engineering, McGill University, in preparation.

Shaheen, S.I., An Implementation of a Relational Database and an Algebraic Sublanguage for a Computer Vision System, Tech. Report No. 78-12R, Department of Electrical Engineering, McGill University, Montreal, Canada, 1978.

Tanimoto, S., Pavlidis, T. A., A Hierarchical Data Structure for Picture Processing, Computer Graphics and Image Processing, Vol. 4. 1975.

Waltz, D.G., Generating Semantic Descriptions from Drawings of Scenes with Shadows, Report TR-271, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, August, 1972.

Zucker, S.W., Relaxation Labelling and the Reduction of Local Ambiguities, Proc. 3rd IJCPR, San Diego, 1976.

Zucker, S.W., Local Structure Consistency and Continuous Relaxation, Presented at the NATO Advanced Study Institute on Digital Image Processing and Analysis, Bonas, France, June 1978.

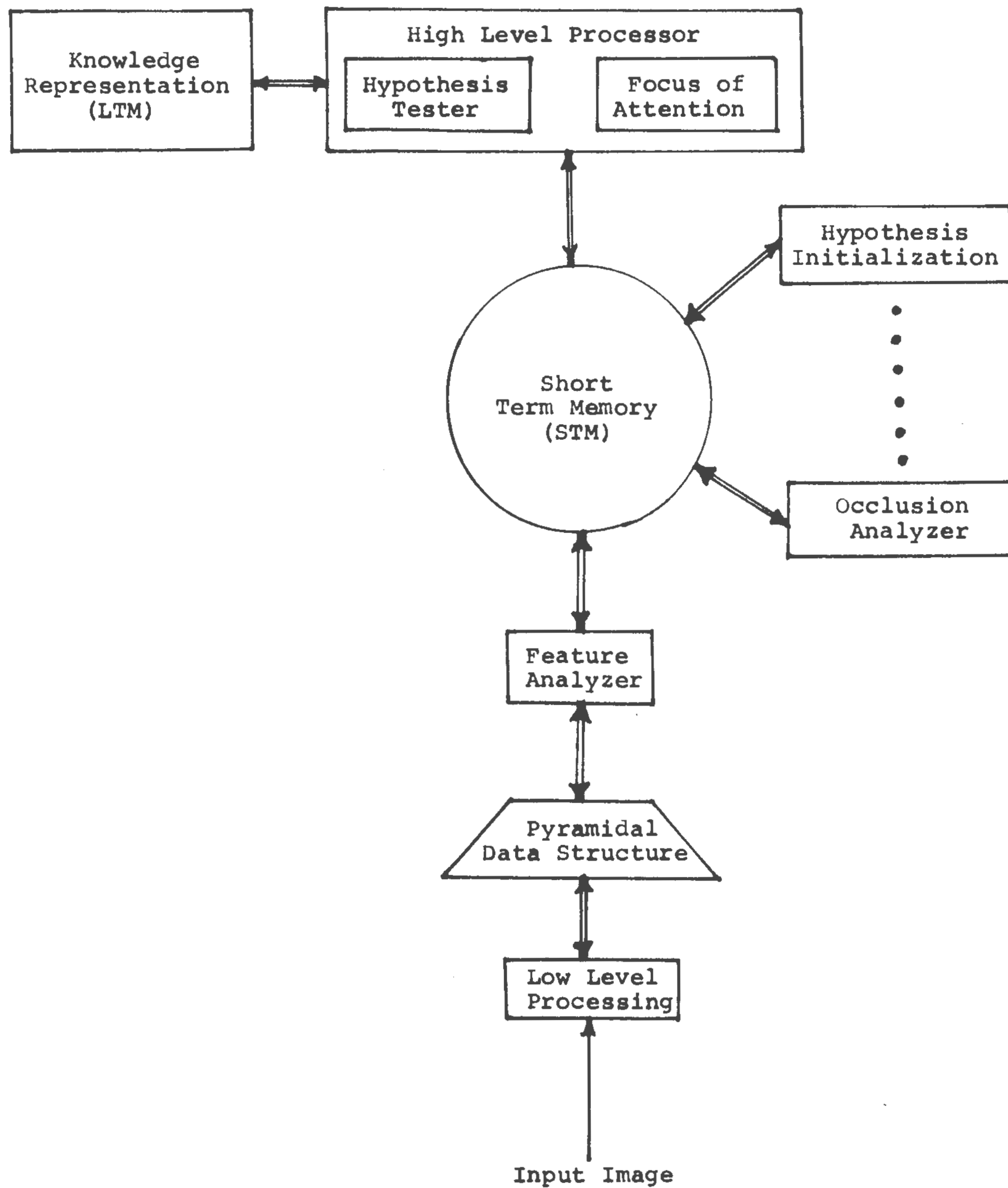
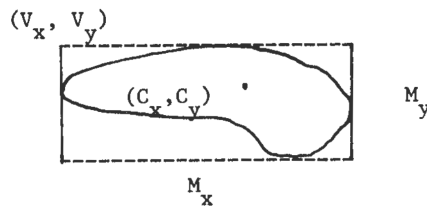


Figure 1. A Knowledge-Based Computer Vision System

Region No.	F_1	F_2	F_3	C_x	C_y	V_x	V_y	M_x	M_y	Area	Perimeter	...
1	2	100	110	15	19	1	1	45	42	1077	209	...
.	.	.	.	.	.	.	.	.	.	.	.	...
.	.	.	.	.	.	.	.	.	.	.	.	...
.	.	.	.	.	.	.	.	.	.	.	.	...
i	F_{1_i}	F_{2_i}	F_{3_i}	C_{x_i}	C_{y_i}	V_{x_i}	V_{y_i}	M_{x_i}	M_{y_i}	A_i	P_i	...
.	.	.	.	.	.	.	.	.	.	.	.	...

Relation 1: Region Attributes

(F_1, F_2, and F_3 are the average attributes used in the description of region color (red, green and blue, or I, Y and Q, or I, H and S).



Region X	Region Y
1	11
1	12
5	12
.	.
.	.
.	.
i	j
.	.
.	.
.	.

Relation 2: X ABOVE Y

Region X	Region Y
1	2
1	4
2	3
.	.
.	.
.	.
i	j
.	.
.	.
.	.

Relation 3: X LEFT-OF Y

Region No.	No. of Active Interpretations	I(1)	C(1)	I(2)	C(2)	I(3)	C(3)	...
1	3	sky	0.65	roof	0.27	cloud	0.08	...
2	2	sky	0.83	roof	0.17	...	...	...
5	1	roof	0.95	...	...	...	...	...
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
i	n	$I_i(1)$	$C_i(1)$	$I_i(2)$	$C_i(2)$	$I_i(3)$	$C_i(3)$	...
.	.	.	.	.	.	.	.	.

Relation 4: Region Hypotheses and Their Confidences

Object X	Object Y
wall	door
wall	window
door	doorknob
sky	clouds
.	.
.	.
.	.

Relation 5: X CONTAINS Y (CONTAINMENT)

Object X	Object Y	Minimum In View	Maximum In View
wall	house	2	2
door	house	1	any
window	house	1	any
doorknob	door	1	1
.	.	.	.
.	.	.	.
.	.	.	.

Relation 6: Complexity Predicate (QUALIFYING: PART-OF)

THE EXTRACTION OF PICTORIAL FEATURES

A.H. DIXON
THE DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF WESTERN ONTARIO
LONDON, CANADA

Yoshiaki Shirai demonstrated the feasibility of the following paradigm for scene analysis: that the construction of an interpretation for a scene should guide the location of search and the type of operations performed on the grey level data [6]. This paper combines Shirai's paradigm with some special procedures for processing digitized images from a television camera into a system which demonstrates a general methodology for the recognition of pictorial features. A side benefit of this approach is that it encourages economical use of storage, fast execution, and distributed processing.

The extraction of meaningful information from the television camera image can be characterized by four major tasks:

1. initial simplification of picture content through image partitioning,
2. extraction of visual cues,
3. aggregation of visual cues into putative pictorial features,
4. verification and refinement of dominant hypothesized pictorial features.

While computations involve every point in the image, they compute only local properties and therefore only a small portion of the image is stored at any one time. An important feature of the system should be its ability to verify hypotheses about the existence of particular features by re-examining sample grey levels from the camera. The result of the initial computations is a list of some of the features of the scene. These are used by a "higher" level system to propose ways of searching for further features with a "second look". New features are integrated with previously discovered features in succeeding "looks" until no new features are found.

This procedure is not intended to find all the pictorial features of any given type. On the contrary, only those features for which strong evidence has been obtained by two different detection procedures, namely aggregation and verification, are hypothesized. In keeping with the overall scene analysis paradigm, a function of the construction of an interpretation is to predict the location of weaker pictorial features, using both the present state of the interpretation and knowledge about the scene domain.

Conceptually, each task can be viewed as a separate computational procedure requiring as input the results of the previous task and generating the inputs for the next task. The interpretation of scene can then guide the feature extraction processes by selection of an appropriate task supplemented by the necessary inputs which may have been already generated or subsequently modified. Moreover, if each process is an independent module, then alternate methods for performing a task can be selected.

To demonstrate the methodology with an implementation, a simple and familiar family of scene domains has been used; namely those domains where the primary pictorial features are associated with straight edges. While this assumption does not affect the overall paradigm, it does suggest the importance of finding ways of characterizing other pictorial features.

IMAGE PARTITIONING

The role of the "first look" at the scene should be to determine, as quickly as possible, a sufficient set of visual cues to enable subsequent procedures to generate a partial description. This description would then guide more intensive analysis of certain parts of the image where closer scrutiny was required. This closer scrutiny could be obtained through use of available hardware such as a zoom lens on the camera, or by the selection of alternate procedures for examining the grey level data in a particular area.

The coarse analysis usually involves partitioning of the image into windows of uniform size and then performing some computation on each window. An adaptation of a line parameterization technique previously described by O'Gorman and Clowes [3] and Duda and Hart [2] has been developed to generate visual cues through image partitioning. This procedure allows for the fast aggregation in a matrix of all the local evidence for contrast boundaries. Every point in the image is associated with some local neighbourhood of pixels upon which a computation is performed. One such computation approximates a directional

derivative at the associated point, if the grey level matrix is viewed as a surface, or function of two variables. Under this interpretation an unique line can be associated with each point. Each entry (ρ,θ) on the matrix corresponds to the collection of points associated with a line oriented at an angle θ from the horizontal, and at a distance ρ from some fixed point in the image.

Rather than apply this technique to the entire image, it is applied to each window in a uniform partition of the image. Local evidence at a particular point (x,y) is determined by applying a set of convolution masks to a region centred at (x,y). Each convolution mask covers a rectangular neighbourhood of points. These points are partitioned into two sets and the magnitude of the difference between the mean grey levels of each set is computed. The mask yielding the largest magnitude determines the orientation θ of the local evidence for an edge. The distance, ρ, is given by eq.(1),

$$\rho = x \cos\theta + y \sin\theta \qquad (1)$$

and the magnitude is added to entry (ρ,θ) of the matrix. After the set of masks has been applied to each point of the window, the largest entry and its position in the matrix determines the location and orientation of the dominant visual cue for an edge passing through the window.

It is evident from this algorithm that the visual cues correspond to measures of evidence for straight edges in the scene. It does not include mechanisms for generating evidence of curved edges. On the contrary, curved edges are viewed as a sequence of short straight edge segments and the existence of such edges would generate corresponding visual cues.

The result of this algorithm is the generation of one visual cue for each window in the uniform partition of the image. The selection of only the dominant visual cue for each window avoids the issue of deciding whether an entry in the rho-theta matrix is significant at the expense of finding fewer edges. Since the goal is the extraction of edges sufficient to guide further analysis of the scene, we need only find enough edges to perform this task successfully.

CLUSTERING OF VISUAL CUES

The complexity of an algorithm for grouping together independent visual cues depends on the scene domain. Being able to exploit knowledge about the scene is even more important when there are few visual cues such as those produced by the image partition method above. In fact, it seems evident that there are not enough visual cues to be able to propose putative edges with any confidence unless some simplifications are inherent in the scene. It is here that the restriction on the scene domain is exploited.

The visual cues of two windows which are 8-adjacent are placed in the same cluster if their respective orientations and locations are consistent with belonging to a common straight edge. One such constraint is that the orientations be nearly the same.

The role played by the location or "rho value" of a visual cue in determining whether two windows should be grouped together depends on the relative positions of the adjacent windows as well as the common orientation. As an example, two horizontally adjacent windows would not normally be associated with a common edge if the orientation of the respective visual cues were vertical. However, if an edge in the original scene were coincidentally near the boundary between two windows, it would provide contributions to the rho-theta matrices of both windows. This is because the application of a convolution mask to a point near the boundary of a window will include points outside that window. For this reason the relative locations of visual cues within adjacent windows are interpreted in the context of the nature of the adjacency (horizontal, vertical, or diagonal).

The perpendicular distance between two parallel lines corresponding to the visual cues is computed and compared to the amount that a convolution mask overlaps the boundary of a window. Two adjacent windows are said to belòng to the same cluster provided the respective lines associated with their dominant visual cues are parallel and sufficiently close together.

HYPOTHESIZING AN EDGE

A cluster of visual cues may be interpreted as supporting evidence for a straight edge passing through the windows associated with the cluster. A point which lies on the edge segment contained within each window is computed from the orientation and location of the visual cue. To the set of points a line is fitted which provides a first representation of the edge. This "hypothesized edge" defines a trajectory along which an appropriate tracking algorithm can examine the original camera image to verify the existence and refine the position and orientation of the edge. In particular the endpoints of the edge are located. This provides information which was not previously available from the rho-theta procedure, being only roughly defined by the two windows furthest apart in a cluster.

The rho-theta procedure has been used by Clowes to construct line drawings of scenes in a puppet world. His algorithm saved the location of all contrast boundary points contributing to each entry in the rho-theta matrix. The hypothesis of an edge in the scene then required that sorting and merging algorithms be applied to sets of points associated with adjacent entries of the matrix. Additional procedures were employed to decide whether matrix entries defined one or more edges.

No re-ordering of a set of points is required in this implementation, since a tracker is used instead. The tracking algorithm proposed selects the three most significant visual cues from a cluster. As mentioned previously, a putative edge has already been predicted by a least squares fit to all the visual cues associated with the cluster. A convolution mask oriented parallel to the

predicted edge is applied along a path perpendicular to the edge and through points associated with the three strongest visual cues. From this, three points are found which purportedly lie on the predicted edge. The tracker now verifies the existence of an edge through these three points by proceeding in both directions from the middle point through the other two until no discontinuities are detected for a consecutive number of attempts. The essence of this method is similar to that proposed by Lerman and Woodham [4] and provides an accurate means of determining endpoints since tracking begins in an area where the edge is most strongly defined.

IMPLEMENTATION

The methodology suggests a natural division of the scene analysis process into two systems. One system deals with the sampling and manipulation of grey level data and the compression of that data into meaningful units which have been called visual cues. The second system integrates these units into an interpretation and directs further sampling and manipulation of the grey level data. Because of the small volume of information exchanged between the two systems, it was appropriate to implement the system on two computer facilities, one dedicated to image processing (an INTERDATA 7/32 minicomputer), and one responsible for maintaining an internal representation of knowledge about the scene domain (a DECsystem10 timesharing facility).

The present implementation of the image sampling system consists of a collection of subprograms for performing the four major tasks described at the beginning of this paper. A supervisory program within the image sampling system is responsible for dispatching requests from the scene interpretation system. A request to perform a particular task or sequence of tasks is made by sending the name of the routine together with an appropriate set of parameters.

The present objective of the scene interpretation system is to construct a complete line drawing for the scene which does not conflict with evidence obtained from image sampling. It is therefore possible for several alternate line drawings to be constructed when the scene consists of edges which are difficult to detect directly. This is because the constraints for accepting the hypothesis of the existence of an edge have been considerably relaxed. The image sampling system can "hallucinate" edges in places selected by the scene interpretation system provided there exists no evidence in either the image or in the knowledge base about the scene domain which would contradict such an assumption.

The initial starting point for the construction of a line drawing by the scene interpretation system is a collection of line segments presented to it by the image sampling system. A partial line drawing is constructed by merging appropriate collinear line segments and by connecting line segments which appear to have a common vertex. The criterion for defining a common vertex is simply that the endpoints of two line segments be within a given distance from one another, relative to the actual lengths of the line segments involved.

The partial line drawing is used in conjunction with knowledge about the nature of edges in the scene domain to formulate hypotheses about the location of missing edges. The present system is modelled after Shirai [6] and the first goal is to find the outside boundary of the aggregate of objects forming the scene so that concave and convex vertices can be distinguished. Using Shirai's algorithm edges are proposed and a task request is made to the image sampling system for verification. The results of verification can be "confirmed", "rejected", or "undetermined", the last category corresponding to the situation where the edge was not found but no conflicting evidence was found either. An example of evidence for "rejection" might be a contrast boundary oriented differently from the predicted edge.

The scene interpretation system must decide whether or not to assume the existence of an "undetermined" edge. The design of heuristics for making such decisions is an area of future research. Presently only two simple strategies have been explored. When "undetermined" edges are "rejected" the system performs as Shirai's algorithm would. This means that unless all edges are strongly defined, a complete line drawing cannot be constructed. When "undetermined" edges are interpreted as "confirmed", complete line drawings are almost always constructed although not always "correctly".

The source of the difficulty is the order in which edges are proposed by Shirai's strategy. To avoid the possibility of accepting a large number of "undetermined" lines, the results of any circular search must be "confirmed". This does not preclude the possibility of confirming the existence of two "undetermined" edges obtained as extensions of a concave vertex. At present, an "undetermined" edge is assumed to be confirmed if the average contrast is reasonably close to an adjustable threshold. The average contrast is one of several values returned by the image sampling system and used by the scene interpreting system to determine the results of verification. Although not implemented, a more satisfactory approach might be to initially accept as "confirmed" the existence of both extensions of a concave vertex and then subsequently to "clean up" the completed line drawing by using knowledge about the scene domain to reject unlikely lines.

SUMMARY

The partial line drawing generated can be completed using procedures similar to those proposed by Shirai [6]. Of particular interest is the fact that the constraints for accepting the existence of an edge can be considerably relaxed. In effect, the system can "hallucinate" edges in selected places providing there exists no evidence in the image or in the knowledge base about the scene domain which would contradict such assumptions. By using domain

dependent knowledge to interpret the partial line drawing, predictions can be made about the existence of other edges. Such predictions can then be verified by subsequent "looks" at the scene.

REFERENCES

1. R.M. Burstall, J.S. Collins, R.J. Popplestone, Programming in POP-2, Edinburgh University Press, 1971.

2. Richard O. Duda, Peter E. Hart, Pattern Classification and Scene Analysis, John Wiley, 1973.

3. P. O'Gorman, M.B. Clowes, Finding Picture Edges Through Collinearity, Laboratory of Experimental Psychology, University of Sussex, 1973.

4. P. Lerman, Robert Woodham, Tracking, New Progress in Artificial Intelligence, Patrick Winston (ed.), MIT AD/A-002 272, June 1974.

5. Donald A. Norman, David E. Rumelhart, Explorations in Cognition, Freeman, San Francisco, 1975.

6. Yoshiaki Shirai, Analyzing Intensity Arrays using Knowledge about Scenes, The Psychology of Computer Vision, Patrick Henry Winston (ed.), McGraw-Hill, New York, 1975, pp.93-113.

ASPECTS OF A THEORY OF SIMPLIFICATION, DEBUGGING, AND COACHING

Gerhard Fischer
Massachusetts Institute of Technology

and

John Seely Brown
and
Richard R. Burton
Bolt Beranek and Newman, Inc.

ABSTRACT

Today, millions of people are learning to ski in just a few days instead of the months it took to learn twenty years ago. In this paper, we analyze the new methods of teaching skiing in terms of a computational paradigm for learning called increasingly complex microworlds (ICM). Examining the factors that underly the dramatic enhancement of the learning of skiing led us to focus on the processes of simplification, debugging, and coaching. We study these three processes in detail, showing how the structure of each is affected by the basic skills required to perform a task, the equipment involved in its execution, and the environment in which the skill is executed. Throughout, we draw parallels between the process of learning to ski and learning computer programming and problem-solving.

Our goal is to achieve insight into the complex issues of skill acquisition and design of learning environments -- especially computer-based ones -- through the analysis of the intuitively understandable domain of ski instruction.

1. INTRODUCTION

The most effective use of computers for education is to support active learning environments in domains that previously had to be learned statically. While some work, though not nearly enough, has gone into developing particular environments, much less has gone into clarifying the general issues that affect the acquisition of a skill in a complex environment.(1) Our own work has led us to believe that a thorough analysis of skill acquisition is necessary to augment our intuitive understanding of the subtleties involved in designing the next generation of learning environments.

In this paper, we examine the learning of an extremely complex skill, skiing, through the language of computational learning environments. We have two goals. One is to explicate the remarkable advances in the methods of _teaching_ skiing, which have greatly reduced the time required to learn to ski. The other is to analyze the features of the highly successful skiing learning environment in an attempt to articulate the fine grain structure of a theory of learning environments and to identify principles to guide the design of computer based learning environments.

The paradigm on which we shall base our examination of the teaching of skiing is called "increasingly complex microworlds" (ICM). In this paradigm, the student is taken through a sequence of environments (microworlds) in which his tasks become increasingly complex. In the analysis of skiing, the aspects of the ICM paradigm we will stress are simplification, debugging, and coaching. Throughout the discussion, we will also point out how the learning experience (as viewed from the ICM paradigm) has been implemented in skiing by three fundamental components of the learning experience: the basic skills required to perform a task, the equipment involved in its execution, and the environment in which the skill is executed. The analysis of skiing

(1) Although one would expect research in the fields of task analysis and behavioral objectives to be relevant, it has not been. This is in part due to the lack of a precise computational theory of teaching and learning, and in part to the lack of appropriate languages for discussing the deep structure knowledge representation of a domain.

raises a host of general questions that should be asked when designing learning environments based on the ICM paradigm. For example, which kinds of simplification can stand in isolation, and which require explicit coaching to prevent the induction by the student of false models that later must be unlearned? Throughout our analysis, we shall draw parallels to skiing from the domain of learning environments that teach computer programming and problem-solving.

2. Why Skiing?

Skiing is an extremely complex skill, to learn and to perform. It is representative of an important class of real-time control skills (or data driven skills), where error correction is essential in order to cope with deviations and sudden changes in the expected environment. However, highly successful methods have been developed to teach skiing. This is not true for most other complex skills. These methods suggest criteria necessary to design successful learning environments for other complex skills. In addition, skiing provides an intuitively understandable domain, with which many people have personal experience.(2) Even nonskiers can relate the examples used in learning to ski to other physical skills, such as bike-riding.

2.1 Skiing as a Success Model

Skiing is an instance of a success model (Papert 1976); it is an example of the successful acquisition of a complex skill. In skiing, the conditions of learning are more important than the total time or mere quantity of exposure. This implies that the teaching of skiing has evolved into a highly successful instructional process. The two main uses of a success model are:

1. to identify the features that make it successful
2. to abstract these features and try to transfer them to less successful learning situations.

We do not have a complete theory to explain why the learning process in skiing was so dramatically enhanced during the last twenty years, but we are convinced that the following features were of great importance:

- Redefinition of teaching goals
- Improved equipment
- Access to new environments
- Better teaching methodologies and conceptualizations.

(2) Our knowledge and insights about skiing are drawn primarily from one of the authors (Fischer) who has worked as a part-time ski instructor for many years.

We are aware that other factors influence the learning process besides the ones we investigate in the following sections. All ski areas have many expert skiers around, so that learning can take place according to the medieval craftsman model. This enhances the ability of the less experienced skier through interaction with the more experienced one.

The person learning to ski is highly motivated. Skiing is fun. It provides a wide variety of experiences; every run is different from the previous run. Skiing is good exercise. It provides a nice change in the life style of many people. In addition, societal pressures contribute to the motivation to learn to ski. Being a skier is fashionable. We will ignore the problems of motivation in this discussion and will assume that the learner is motivated. Although motivation is clearly an important consideration in the design of learning environments, we shall not address it in this paper.

We must also note a few of the negative aspects of skiing: it is expensive, it is time-consuming, and it can be dangerous. For these reasons, the task of identifying the aspects of skiing that make it a success model becomes even more interesting.

2.3 The ICM (Increasingly Complex Microworlds) Paradigm Applied to Skiing

The acquisition of a complex skill is difficult when the starting state and the final state are too far apart. Good learning environments, structured according to the ICM paradigm, provide steppingstones or intermediate levels of expertise so that within each level the student can see a challenging but attainable goal. In skiing, technological advances and the methodologies built around these advances make it easy to get started. This means that practice (a task within an intermediate level) is not considered a form of torture that must be endured before the learner can enjoy excellence.

As an example of the ICM paradigm in skiing, consider a novice learning to ski. The student begins on short skis over smooth terrain. The short skis allow him to develop rhythm, and they make it easier to turn and get up from a fall. The smooth terrain limits his speed and reduces the danger. As the student gains ability within these constraints, he is given slightly longer skis and steeper, more complex slopes until he is using full length skis on uncontrolled slopes. At each step, the microworld in which he must perform is made increasingly complex.

We should point out that the ICM paradigm may be usefully applied to sports other than skiing. A large body of knowledge about skill acquisition is available in the literature of different sports. The authors of textbooks for these sports supply a great deal of knowledge about the critical components and essential steppingstones for the complex skills they describe, as well as awareness of the most common problems and special exercises to eliminate them. However, these authors often lack a conceptual framework that would allow them to generalize their knowledge or to structure it according to different criteria.

We would like to acknowledge the work by Austin (1974). He analyzed the skill of juggling in terms of a computational metaphor and used the resulting analysis to develop novel methods of teaching juggling. In our work, we seek to analyze the process of learning to ski within the framework of the ICM paradigm, with the goal of expanding the paradigm.

3. Aspects of a Theory of Simplification

One of the major design decisions within the ICM paradigm is choosing or generating appropriate microworlds. The primary means of generating alternative microworlds is through simplification. This section describes a taxonomy of knowledge, methods, and heuristics that could serve as a basis for evolving a theory of simplification in the learning process.

Simplifications are possible in each of the three major components of the learning process: the skill required to perform a simplified version of a task, the equipment involved in executing the task, and the environment in which the task is executed. Often it is not just one of the components, but their synergistic interaction, that leads to powerful learning microworlds.

3.1 The Basic Skills

The designer of a learning environment can select some beginning microworlds for developing particular subskills in isolation. Some of the basic physical skills of skiing can be taught without skiing. Students can thus develop these subskills without having to deal with the interactions and side effects of the whole aggregate of subskills. Examples would be: learning a certain rhythm, strengthening certain muscles, and improving the mobility of certain parts of the body. At a more advanced level, a trick skier may practice his somersaults into a pool or on a trampoline.

Great care must be taken to choose a microworld in which the simplified skill is isomorphic in its most important components to the final form of the skill (see Section 3.5). In juggling, the skill of ball-handling can be practiced with one or two balls. This develops the necessary subskills of tossing and catching, as well as hand-eye coordination. However, the easiest form of three-ball juggling, called cascade juggling, can't be simplified to an isomorphic two ball juggling (see Austin, 1974).

3.2 The Equipment

The best known example of a simplification of equipment in skiing is the graduated length method. In this method, a beginner skier is started on short skis. As the student becomes proficient, his skis are gradually lengthened to (whatever may be considered) full length skis. Short skis are used as transitional objects in the learning process. They make it easier to get started and make early success more likely. At the next level, the shorter skis are not needed anymore. An interesting perspective on the hand-held electronic calculator may be to view it as a transitional object in learning mathematics. Similarly, the computer may serve as a transitional object in learning how to build cognitive models.

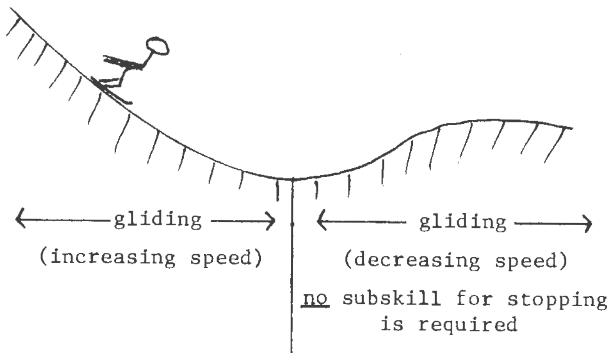
It is interesting to ask why it took so long for someone to think of using short skis in the learning process. For one thing, skiing itself changed. Twenty years ago, people wanted to ski fast in straight lines for which longer skis are better. Nowadays the final state of expert skiing involves making many turns (which is facilitated by short skis). For another thing, teaching by the graduated length method requires a different instructional organization. To be economically feasible, the new method needs large ski schools where students can rent short skis instead of buying them, so they can be returned after they are no longer needed. The economic consideration that has hindered exploration of transitional objects in learning will not be as important in computer-based learning environments, because the transitional objects are symbolic structures.

Short skis are not the only technological improvement in the equipment used in skiing. Safety bindings reduce the fear and eliminate the catastrophic consequences of wrong behavior, therefore, supporting an active approach to mastering new challenges. (In an interactive computer system, the "UNDO" command supports a similar type of exploration because it reduces the risk involved in making errors.) Ski tows and gondolas provide access to new environments in the form of moderately steep and wide glaciers with snow conditions suited

to the early phases of the learning process. In addition, they increase considerably the time that people can actually spend skiing. A parallel improvement in computer programming is the development of time-sharing systems and languages that reduce the amount of time a student spends waiting for his program to be run.

3.3 The Environment

Skiing (as we have pointed out before) is an aggregate of subskills. A major aid in learning any complex collection of skills is the opportunity to practice the subskills independently. We must design or find microworlds structured to allow a learner to exercise particular skills. For the beginner in skiing, gliding and stopping are two essential subskills that have to be learned. But stopping cannot be practiced without gliding, and gliding is dangerous unless you know how to stop (in Simon's words (1969), the system is only nearly decomposable). The problem can be solved by choosing the right environment:



This example leads us to state: The decomposability of a skill is a function of the structure of the environment as well as of the skill itself.

Modern ski areas have made another important contribution to the simplification of the environment. They provide the novice with constant snow conditions. A beginner can first learn to maneuver well on packed slopes without having to worry about the variabilities of ice or deep powder. In learning to play tennis, the ball shooting machine provides a similar form of simplification. Having a supply of nearly constant balls removes some of the variables from the process of learning a stroke.

The wide variety of slopes in a large ski area has another important impact on learning. It allows the coach to choose a microworld dynamically according to the needs of the learner; this eliminates the need to force every learner through the same sequence of microworlds.

3.4 Simplification's Dependency on Top-level Goals

Technological improvements have eliminated certain prerequisites for skiing, that is, they have simplified skiing by removing inessential parts. It is not necessary any more to spend a whole day of hard physical exercise in order to gain a thousand meters of elevation to ski one nice run. The goal of skiing is gliding downhill successfully, not getting stronger muscles and a better physical condition by climbing uphill for several hours. If climbing were one of our top level goals, the use of gondolas and chair lifts would hardly be an appropriate simplification towards the acquisition of these skills. Clarifying the top level goals may imply a different standard of measurement for the hierarchical ordering of the subskills and a corresponding change in the sequence of microworlds.

The importance of clarifying top-level goals can also be seen in programming. As computing becomes cheaper, concerns about machine efficiency will be replaced by concerns about cognitive efficiency, how to facilitate the understanding and writing of programs. This change in perspective requires new conceptualizations and methodologies, which will lead to a new set of simplifications for the acquisition of the skills of programming and problem solving (Fischer 1977).

3.5 Useful Versus Possible Simplification

The range of possible simplifications is much larger than the range of useful simplifications. The designer of a learning environment must look carefully at what each microworld does for the overall goal. Several possible uses for a microworld come to mind. A microworld:

- Makes it easier to begin learning a skill by creating the right entry points
- Accelerates the acquisition of a skill
- Provides intermediate goals/challenges that are (and seem to be) attainable
- Provides practice of the important subskills in isolation, allowing the common bugs to occur one at a time instead of in bunches

A complicating factor in choosing microworlds is that non-monotone relationships often exist between simplifications of the microworld and the corresponding simplifications of the task. Using a moderately steep hill to practice is a useful simplification for the following reasons:

- o Is easier to control speed.
- o The student doesn't have to make big turns and can stay closer to the fall-line.
- o The student doesn't have to lean away from the hill with his upper body (which appears to be counterintuitive for many people and increases their fear).

The interesting fact is that this is not a monotone relationship: If the hill is too flat, it may be impossible to attain enough speed to turn. Another example of this sort is that skiing is difficult on a slope with big mogels, but, in making turns, small mogels can be very helpful.

3.6 The Danger of Oversimplification

Skiing is representative of an important class of real-time, data-driven control skills. This means that a sudden, unexpected change in the environment requires high-order error correcting and debugging skills to cope with the deviations. If the microworlds are too friendly (which may serve well in getting started) they may suppress the development of these higher-order skills. The skier must learn to cope with icy spots and rocks that lie hidden under soft snow.

Developed ski areas themselves constitute a simplification, because they close avalanche areas and keep the skier away from cravasses, they pack down slopes, they rescue people if they get hurt, etc. This implies that people skiing only in these areas never acquire the planning and debugging knowledge they need to move around in more hostile environments. One danger of working with simplifications is that they may lead to unjustified extrapolations. One task of a good coach is to reduce the level of protectiveness gradually (not all ski areas eliminate the need for stopping) and lead people to the right new challenges. There is another danger: Learning to perfect the performance in one environment, such as packed slopes, may reduce the willingness of a skier to practice in powder, because the difference in his performance between the two environments may be too great.

Both of these dangers can be seen in efforts to teach computer programming that start with BASIC. The linear nature of a program in BASIC and the small size of solutions to typical introductory problems often lead students to develop debugging strategies that will not generalize to large programs. One such strategy is to step through a program one statement at a time. Some students also resist leaving friendly (albeit limited) BASIC environments, in which they can adequately solve small problems, for the complexities of data declarations, functional decomposition, and advanced control structure statements. Note that these extrapolations are not ones intended by those who have designed the learning environment. They arise from simplifications made to create the microworlds in BASIC. Understanding the inappropriate generalizations that can develop in each microworld is one of the tasks facing a learning environment designer.

4. DEBUGGING

4.1 The Importance of Debugging to the ICM Approach

As a student moves from one microworld to one at next level of complexity, he may need to modify his knowledge in several ways:

- o New subskills may be introduced that must be mastered (skiing over mogels).
- o Changes in the environment may require new interactions between skills (gliding and stopping).
- o Some skills that were idiosyncratic to a microworld may have to be unlearned.

While a designer should strive for simplifications that reduce the chances for incorrect generalizations, this is not always possible nor necessarily desirable. In skiing, an instructor has the problem of how to deal with the poles. Even though they are quite important for the advanced skier, the only major skill a beginner need learn is to carry them so that he won't hurt himself. While practice without poles would prevent formation of any inappropriate skills, empirical evidence suggests that eliminating the use of poles is not a useful simplification. Even if they are used incorrectly, the poles still support balance and mobility, and it is apparently easier to unlearn an incorrect use of poles than to incorporate the poles into a learned skill without using them from the beginning. The goal of a sequence of microworlds is not to remove all chances for misconceptions, but instead to increase the possibility that the student will learn to recognize and correct his own mistakes.

4.2 Nonconstructive Versus Constructive Bugs - Implications for a Piagetian Environment

An important characteristic of a Piagetian environment (Papert, 1978) is the notion of a constructive bug: the learner gets enough feedback to recognize a bug, to determine its underlying causes, and on this basis, to learn procedures to correct the bug. This notion is sharply contrasted with the notion of a "nonconstructive" bug, where a student may recognize he is wrong but not have the necessary information to understand why.

The critical design criterion for selecting the right microworld may well be finding an intermediate microworld that transforms nonconstructive bugs into constructive ones. From the domain of skiing, examples of environmental support for such a transformation follow:

- o If the skier leans too much to the hill with his upper body, a change to a steeper hill will indicate this to him, because he will start sliding down the hill.
- o If he holds his knees too stiffly, trying to stay on the ground while skiing over a bumpy slope will point out his inflexibility.
- o If he doesn't ski enough on the edges of his skis or if he makes turns too sharply, a slope with soft snow, where he can observe his tracks, will indicate where each of these conditions are occurring.

In all of these cases, the microworld is chosen to allow the student's previous experience to be used to debug the new task.

A good coach knows a large number of specific exercises (micro-microworlds) designed to transform nonconstructive bugs into constructive ones. These exercises are goal-directed toward certain bugs. His expertise must include the ability to distinguish the underlying causes (which may be hidden and indirect) from the surface manifestations of the bugs. To mention just one example: lifting up the end of the inside ski in a turn provides the skier with the feedback that most of his weight is on the outside ski (where it should be). Exercises of this sort (which provide the basis for self-checking methods) are of vital interest and are essential in teaching and learning a physical skill (for examples, see Carlo, 1974 and DVSL, 1977) whereas in the cognitive sciences, research in self-checking methods is still in its infancy (see Brown and Burton, 1978).

Another way to turn nonconstructive bugs into constructive ones is through the appropriate use of technology. The most obvious example is the use of a video camera, which helps the student to compare what he was doing to what he <u>thought</u> he was doing.

5. Coaching

Acquiring a complex skill, even when supported by a good learning environment and appropriate technology, does not eliminate the need for a good coach. The introduction of simplifications increases the importance of a coach. He must be able:

- o To make sure that within each microworld the right subskills are acquired, instead of ones that would later have to be unlearned.
- o To design the right exercises, provide the right technology, and select the right microworlds to turn nonconstructive bugs into constructive ones.
- o To perform a task in the student's way in order to maximize the student's chances of recognizing his bugs.
- o To mimic and exaggerate the behavior of the students.
- o to explicate his knowledge in terms the student can understand and execute.

The following example may be used to illustrate the need for executable advice. Many books are written from the instructor's point of view. The student often receives advice (in the book or on the ski slope) that he cannot execute. An example of such advice is, "Put your weight forward," given to skiers who don't know where their weight is. The instructor tells the student the "what" without telling him the "how" and without providing him with knowledge or procedures to translate the "what" into the "how".

Let us give another example of the distinction between executable and observable advice. When skiing in powder snow, the advice, "Your ski tips should look out of the snow", is <u>observable</u> by the student. That is, the student can see whether his ski tips stick out of the snow or whether they are buried below the surface. But the advice is not directly executable. The corresponding executable advice would be "Lean backward," (or "Put your weight backward", if he knows how to shift his weight. This advice is not directly observable. The interesting dependency relationship is that the "what" can be used to control the "how." The change in language from "how" to "what" as a process becomes understood, characterizes the movement from machine to higher-level programming languages.

Let us mention briefly a few other important aspects of coaching. The coach must:

- o Draw the borderline between free and guided exploration (free exploration in a dangerous environment could end up with the student in a cravasse or an avalanche)
- o Decide when to move on to avoid simplified versions of the skill that cause bad habits
- o Be aware that coaching is more important at the beginning of the acquisition phase then later on because a conceptual model must be created, entry points must be provided, and self-checking methods must be learned (to overcome the problem that it is hard to give yourself advice).

6. Aspects of a Theory

There is no doubt that a theory of simplification, debugging, and coaching would provide us with better insight into the complex issues of skill acquisition and design of learning environments. We hope that our observations, examples, and conclusions are a first step toward this end. We believe that a theory of this kind will not be reducible to one or two general laws; that is, we won't be able to characterize such a theory with a few theorems. We expect that the difficulties encountered in constructing a crisp theory in the domain of learning environments will be similar to those encountered, for example, in developing a theory of semantic complexity, (Simon, 1969).

Acknowledgements

This research was supported in part by the Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute for Behavioral and Social Sciences, and Navy Personnel Research and Development Center under Contract No. MDA903-76-C-0108.

We would like to thank Seymour Papert for a number of interesting discussions. We would also like to thank Kathy M. Larkin for her comments on earlier drafts of this paper.

REFERENCES

Austin, Howard. A Computational View of the Skill of Juggling. Massachusetts Institute of Technology, AI Memo No. 330, December 1974.

Carlo. The Juggling Book. Vintage Books, 1974.

Brown, J.S. & Burton, R.R. Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. Cognitive Science, Volume 2, Pages 155-191.

DVSL (Deutscher Verband fuer das Skilehowesen). Skilehoplan, Volume 1 - Volume 7, BLV Verlagsgesellschaft, Munique

Fischer, G. Das Loesen Komplexer Problemanf-gaben durh naive Beuutzer mit Hilfe des interaktiven Programmierens, FG CUU, Darmstadt

Papert, S. Some Poetic and Social Criteria for Education Design. Massachusetts Institute of Technology, AI Memo No. 373, June 1976.

Papert, S. Computer-Based Micro-Worlds as Incubators for Powerful Ideas. Massachusetts Institute of Technology, AI Laboratory, March 1978.

Simon, Herbert. The Sciences of the Artificial. Massachusetts Institute of

KNOWLEDGE STRUCTURING: An Overview

Mark S. Fox

Computer Science Department[1]
Carnegie-Mellon University
Pittsburgh, Pennsylvania
17 April 1978

Abstract

Knowledge structuring is the process by which a set of partial, unrelated concepts are combined (related) to form a unified structure. The structuring process creates a hierarchy of concepts and relations that cover the original set. Examples of knowledge structures described here are classification structures (e.g., phylogenetic classification of mammals), and implication structures (e.g., evidence supporting the hypothesis "strategic threat"). The concept of an "Exo-criterion" is introduced to guide the structuring process. A Knowledge Structuring System currently under development is described, including three knowledge acquisition methods: Introspection, Question-asking, and Experimentation.

1. Introduction

This research investigates the transformation of information into knowledge. Information is technically defined as unrelated units (e.g., facts, concepts, objects, etc.) -- unrelated in the sense that no conceptual links (e.g., "is-a", "part-of", "owns", etc.,) exist among them. Knowledge is the result of structuring information. Structuring can be viewed as the construction of relations and concepts, usually hierarchical, that combine information into coherent structures. For example, the following is a set of information units: {Chipmunk, Rhino, Gazelle, Lion}. An example of structuring this information is the phylogenetic hierarchy normally employed in zoological texts. The root concept of this hierarchical structure is Mammal. This is not the only possible structuring. There exist an infinite number of possible structurings dependent on the reason for structuring. A phylogenetic structuring describes evolutionary (ancestry) relations via a classificatory hierarchy. Another, totally different, hierarchy could be composed based on survivability methods. Figure 1 depicts such a hierarchy. The same information as found in the phylogenetic structure is operated upon but different and possibly, new relations and concepts are used.

The basic principle is that structure generation can be adequately modelled as being guided by some *criterion*. A criterion is a point of view or a proposed application of the information. The structuring process is not limited to the information-knowledge transformation, but can be applied to the restructuring of knowledge depending on the criterion.

Little research has been done in this direction. In most systems, the criterion is implicitly defined. Learning programs (Samuel, 1963; Winston, 1970; Michalski, 1974; Hayes-Roth, 1976; Fox & Hayes-Roth, 1976; Fox & Reddy, 1977; Buchanan & Mitchell, 1977) are tailored to their domain. Functions that construct and rate descriptions reflect the specific learning problem. The learning of different descriptions, based on different perspectives of the same exemplars requires a major change in the algorithm[1]. Other systems may be general but lack the criterion mechanism, thus floundering in their computation. To combat the combinatorial choice of inferences to draw, systems such as HEARSAY-II (Erman & Lesser, 1975; Erman, 1977) use a *Focus of Attention* (Hayes-Roth & Lesser, 1977) mechanism. AM (Lenat, 1976) uses an *Agenda* combined with local heuristics that rate the inferences. In both cases, the rating function or heuristics are built into the system. In contrast, the knowledge structuring system's criterion is a parameter which is external to the system and can be <u>easily</u> changed. We call this an *Exo-Criterion System*.

In contrast, Numerical Taxonomists (Sokal & Sneath, 1963) in the construction of classification structures have taken the view that similarity measures should be used which are based solely on numerical methods (e.g., cluster analysis) using as many object attributes as possible[2]. The only way numerical taxonomy may bias the classes constructed is to apply <u>user-defined</u> weights to the attributes. This is a crude, unmanageable approximation of an exo-criterion system[3]. The criterion is implicitly provided by the user through the weights.

The following describes further the concept of knowledge structuring and illustrates it using two diverse examples. The rest of the paper outlines the design of the knowledge structuring system (KSS) currently under development.

---

[1]This work was supported in part by the Defense Advanced Research Projects Agency under contract no. F44620-73-C-0074 and monitored by the Air Force Office of Scientific Research, and by Grant MCS77-03273 to The RAND Corporation from the National Science Foundation. In addition, the authour was partially supported by a National Research Council of Canada Postgraduate Scholarship.

[1]Michalski's system allows the user to specify different simplification functions (criteria) for each task.

[2]One of the problems common to these methods is the inability to use nominal or ordinal information in measures.

[3]Given that there is a large number of objects to classify and each has a large number of attributes, the number of weights to be defined is overwhelming. In addition, weight consistency between attributes is difficult to enforce.

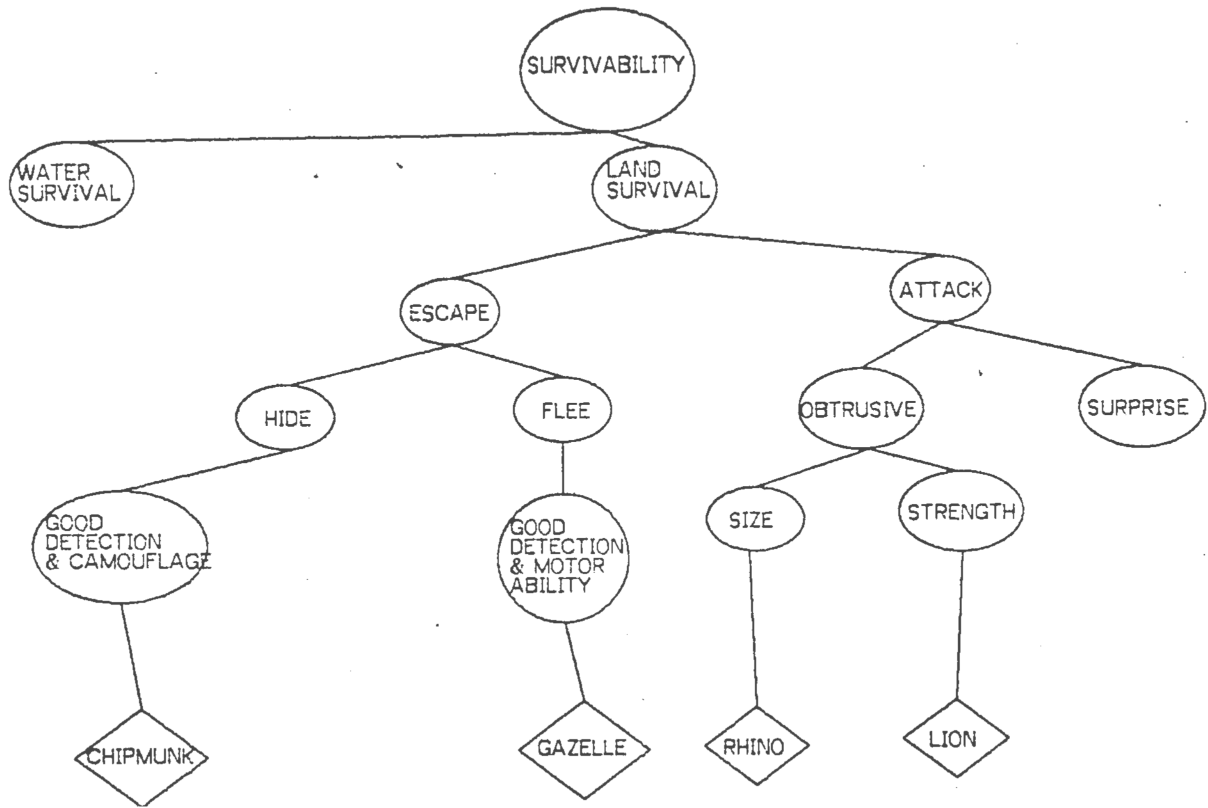


FIGURE 1: SURVIVABILITY CLASSIFICATION OF MAMMALS

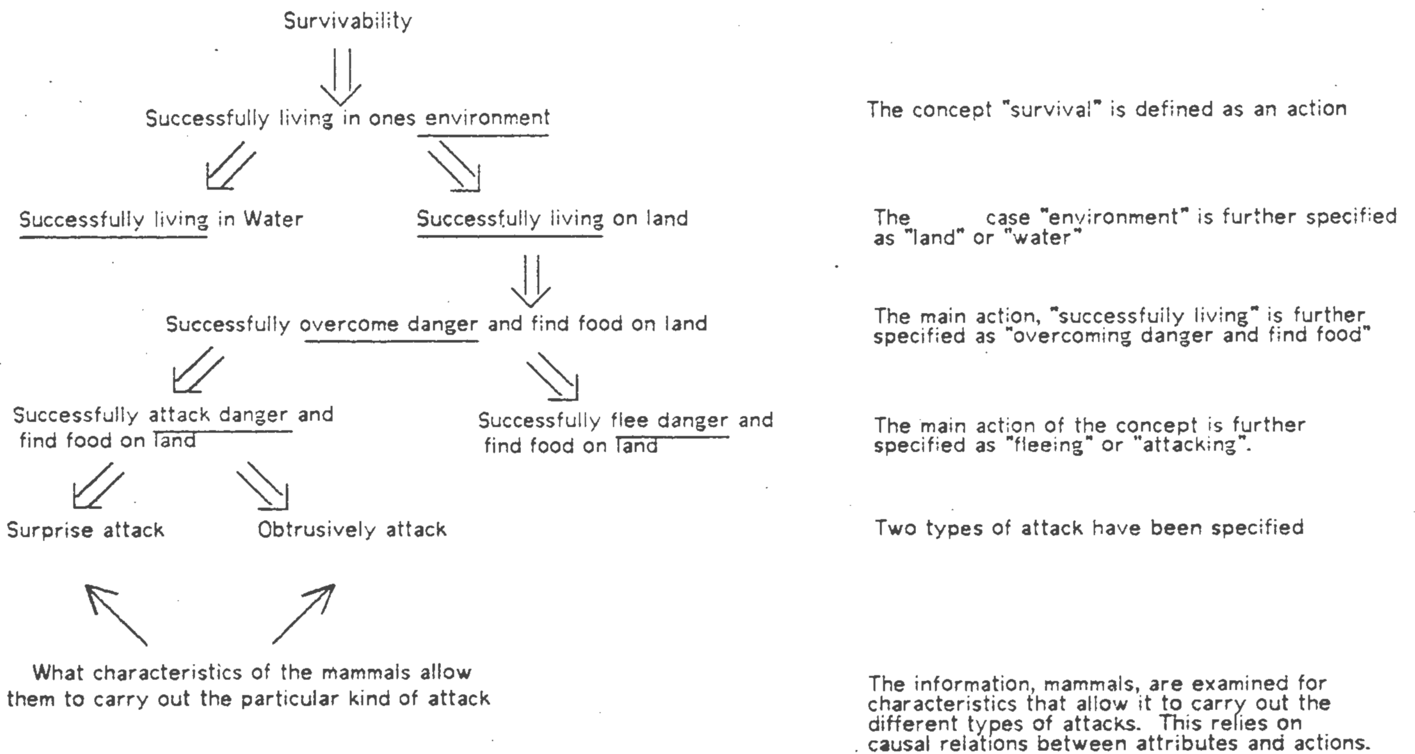


FIGURE 2: Partial Derivation of Knowledge Structure for criterion "Survivability".

2. Knowledge Structures

Up till now, the difference between information and knowledge has been stressed. (Each piece of information is a primitive unit lacking any connection to other information primitives.) In fact, information primitives are separate, independent, but internally structured; i.e., they are little pieces of what I am technically calling knowledge. That is, each information unit is a *Knowledge Structure* (KS) containing various types of relations. The facility of cognition afforded by considering a structure to be a simple information unit is exactly that of abstracting the description of a complex object. Viewing a KS as an information unit serves to reduce the complexity of the structuring task. Hence, knowledge structuring, in the sense used here, takes KSs and relates them by superimposing a structure, thereby creating a new Knowledge Structure.

KSs are constructed via complex symbolic processing of information units to derive relations that are abstract in nature. A new knowledge structure can be viewed as a level 2 KS, i.e., $KS^{[2]}$, which was created by structuring KSs of level 1, i.e., $KS^{[1]}$s. This definition is recursive. A $KS^{[3]}$ can be defined that treats $KS^{[2]}$ as information units. This is intuitively pleasing as we each form multiple hierarchies to structure our information, each hierarchy emphasizing a different relational type[1].

In the knowledge structuring task, the system is presented with information units that are KSs of varying degrees of complexity, and only partially defined. The system must view each KS as a simple information unit to be related to other units, but must have access to the internal structure of each unit (KS) when deciding upon the relationships.

If knowledge is represented in graph form, as implied above, information units can be defined as a nodes, and structure as arcs relating the information units. Any sub-graph is a knowledge structure. A node that represents and replaces a KS sub-graph is also an information unit. This new node can be related to other nodes in the graph. This is exactly what happens in the recursive structuring of knowledge described above. Every KS can be an information unit, and every information unit a KS. Part of the knowledge structuring task is to create new relations among nodes. Hence the number of possible KSs is unbounded.

Knowledge structuring can be defined as the selection of interesting KSs from an infinite space and the construction of new KSs upon those chosen, via the attachment of relations. This definition generalizes from the concept learning tasks of Winston (1970) and Hayes-Roth (1976) in that hierarchies of abstractions are formed. The goal of knowledge structuring is to take information units, which are themselves complex (multi-level), and to relate them through hierarchies of (sometimes new) relations and concepts.

How is one to decide the interestingness of a KS, new or old, or choose the relations with which to tie information units together? Again the exo-criterion is appealed to. For example, the phylogenetic classification of mammals requires that each information unit (e.g., lion) be a complex structure containing data such as physical characteristics: (colour height, weight), eating habits, habitat, etc. A choice must be made of what sub-structures within the mammals' KSs to use to catalyse the structuring process. The choice is dependent upon how the sub-concepts inferentially relate to the criterion. The relations used to structure the information units are also dependent upon the criterion. In the phylogentic example the relation is a class hierarchy.

[1]While most structures are not hierarchical, they are loose hierarchies because of this layered approach to building knowledge structures. The existence of multiple hierarchies indicates information may be structured in more than one way depending on the relations used which depends, in turn, on the criterion.

3. Examples

Figure 1 is a structuring of mammals based on attributes necessary for their survival. The root node, SURVIVABILITY, is the criterion of the hierarchy. Upon examination we discover that the single word SURVIVABILITY does not suffice as a criterion. There exist many fundamentally different structures dependent on the criterion type and SURVIVABILITY is not specific enough to reduce the ambiguity. In this case, the implicit criterion is: "to classify the mammals with respect to the attributes that lend to their survivability." Thus the structure type is a *Classification*. The structuring process as depicted in Figure 2 can be described as the further specification of the criterion into conjunctive and disjunctive sub-criteria combined with causal relations between mammal attributes and specific actions.

The structuring process relies heavily on the further specification of cases[1]. If we define SURVIVABILITY as: "SUCCESSFULLY LIVING IN ONE'S ENVIRONMENT," ENVIRONMENT can be further specified as LAND or WATER: "SUCCESSFULLY LIVE ON LAND" or "SUCCESSFULLY LIVE IN WATER". Hence two sub-classes have been specified. Specification continues until a sequence of relations is found connecting the sub-criterion (class concept) and information unit. Specifically, a chain of inferences (e.g., causal, ancestral, etc.) relating mammal attributes to a sub-criterion must be constructed. This chain may be at an arbitrary level of abstraction depending on the domain information available. For example, concepts such as "appendage" and "striking" are less specific (more abstract) than "paws" and "clawing".

The resulting hierarchy can be evaluated structurally by how well the classes partition the information, and semantically by how a class relates to the criterion.

Another example of knowledge structuring occurs in strategic analysis. A strategic analyst peruses information emanating from some country and decides whether that country's actions comprise a strategic threat (e.g., the building of nuclear bombs). Large quantities of information must be reviewed (e.g., newspaper reports). The information is interpreted with respect to its strategic importance. The interpretation process is called *synthesis*. Figure 3 depicts a hierarchy that synthesizes individually uninteresting information to support a strategic threat interpretation. The synthesis process can viewed as another form of knowledge structuring. The criterion is: "THERE EXISTS A STRATEGIC THREAT." The structuring problem is to see if the information implies the strategic threat. This type of structuring is called *Implication*. Figure 4 illustrates a partial derivation of the structuring. Again heavy use is made of case-specification. But in an implicative structuring, emphasis is placed on the

[1]Cases in a concept are the slots whose interpretation is similar to that of cases in Fillmore(1968).

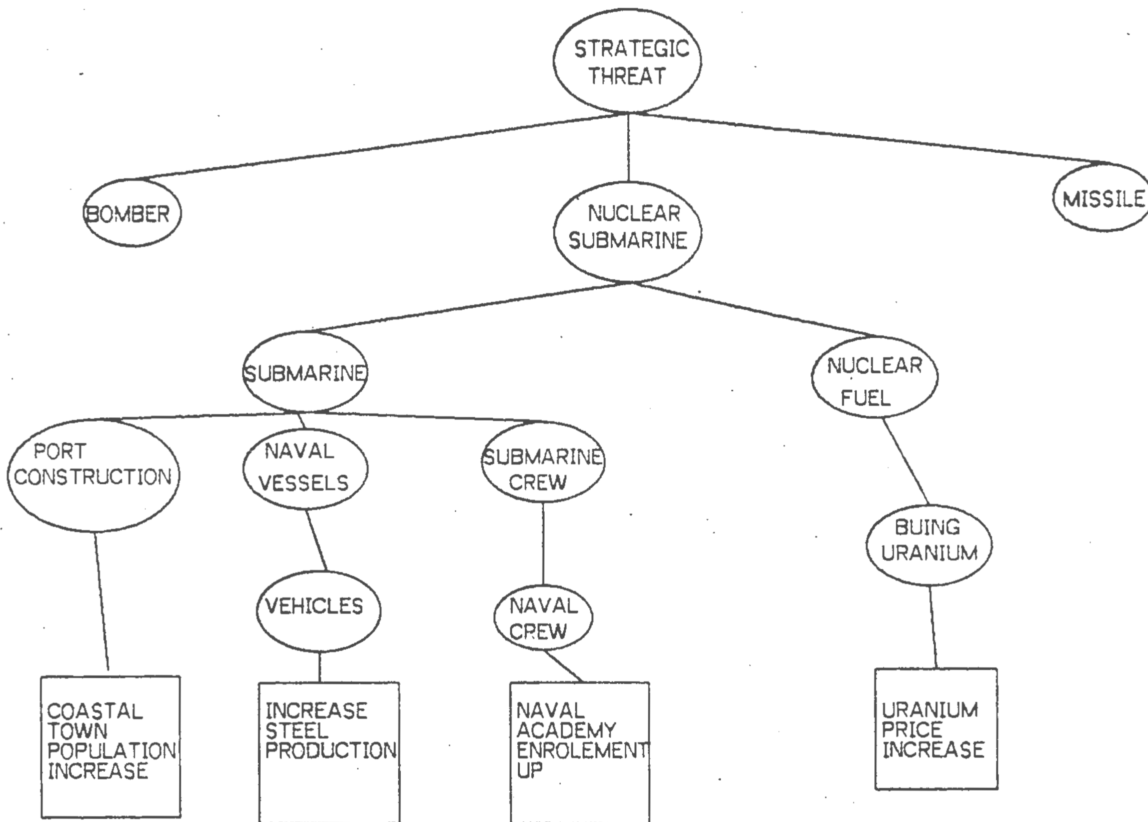


FIGURE 3: Synthesis of information to support "Strategic Threat".

Strategic Threat — The original, ill-defined criterion

There exists a strategic threat — The criterion is further specified

There exists objects capable of destroying the country — Strategic threat is further specified as an action

∃ missiles capable of destroying the country

∃ nuclear submarines capable of destroying the country

∃ bombers capable of destroying the country

The agent case, "objects", is further specified by three types. The specification is constrained by the action of the agent.

∃ a submarine ......

Nuclear fuel

On the left, the sub-criterion is further specified, on the right the pre-condition of nuclear fuel is depicted.

Build a submarine

Submarine crew

Must buy uranium

The rest is a brief outline of the causal requirements of each concept in the structure.

Naval crew

Port Construction

Materials

Uranium prices increase

Naval Academy Enrollment up

Increase steel Production

Coastal Town Population increase

FIGURE 4: Structuring of news using "Strategic Threat" criterion.

elaboration of *Causal* and *Co-relational* pre-conditions (e.g., IF [pre-condition] THEN [post-condition]). This is due to the predominance of causality-explaining nodes in the target knowledge structure.

Structure evaluation relies upon pre-condition satisfaction in causal relations, and the semantic relationship between the causal relation and the criterion.

An important characteristic of an exo-criterion knowledge structuring system is it allows the synthesization of the same information from more than one view (criterion). This allows the testing of alternate interpretations of the same information. For example, the system could be re-run with the criterion of "strengthening merchant shipping".

These examples illustrate two types of knowledge structures. The first is a *classificatory structure* which requires the information covered by a class to reflect the class description. The members of a class induced partition possess similar attributes that are interesting with respect to the criterion. The second is an *implicative structure*. Information is combined based on its causal necessity in explaining the criterion. The construction of causal chains relating information to criteria requires a problem-solving capability similar to those found in (Newell & Simon, 1963; 1972), (Fikes & Nilsson, 1972), and (Sacerdoti, 1975). Plausable causal and inferential sequences at various levels of abstraction must be constructed and analysed to relate information and criteria.

Both structure types rely upon the same mechanisms. Two inference mechanisms have proven useful. The first is *concept specification*. By further specifying abstract terms within a concept, new concepts are created representing the new sub-class (sub-criterion). This specification is constrained by the context-sensitivity of the case being specified; what role the slots play relative to the concept it is part of. The second mechanism is the elaboration of pre and post-conditions of causal and co-relations. This is a search based mechanism that examines concepts that already exist in the knowledge base.

Underlying these structures is a representation of knowledge that facilitates the representation and manipulation of arbitrary concepts. A concept must encompass the semantics of a criterion and preserve its intrinsic ambiguity. The process of structuring in the examples can be viewed as the reduction of criterion ambiguity by specification and causal analysis.

4. A User's View

The knowledge structuring system is designed to be interactive; allowing the user to define the criterion and information to be used in the structuring process. A natural language interface is not anticipated as it is beyond the scope of this research. A minimal interface will be provided. The user must express information to be added in concept form. It is anticipated that a certain portion of information requested by the system will be specific enough to reduce the complexity of this interface. The system will provide help facilities describing existing concepts, slots, rules, etc.

The system can be viewed as an investigator. Once given the criterion, and the partially described information, the system may ask the user directed questions to augment its knowledge base. The method of knowledge acquisition by questions is described in Section 8. An important assumption is that the knowledge base used in structuring lacks, initially, the large amounts of knowledge necessary to carry out the structuring. It is up to the system to acquire this knowledge using the methods available to it (See Section 8).

As the system is used, the knowledge base grows. The structuring process is not simplified by the accumulation of more knowledge. The structuring process shifts from discovering new relations among concepts (information, criteria) to deciding which relations should be emphasized and which should be ignored. The balance of knowledge acquisition and focused search will be investigated in various applications.

The processing of the system can be divided into three categories. The first, *Bottom Up Theory Hypothesization*, analyses the information units to be structured to create concepts that describe (classify, synthesize, etc.) subsets of them. The analysis can be independent and/or dependent upon the criterion. Some mechanisms used in the analysis are: similarity measures (e.g., cluster analysis, interference matching), morphological similarity, and analogical analysis between concepts.

The second category of processing, *Top Down Theory Hypothesization*, carries out the search process depicted in section 3. Given a criterion concept, new concepts are created and old ones instantiated during the search process.

The third category, *Discovery*, augments the knowledge base via three mechanisms. The first is *Introspection*. During the knowledge base search, portions of concepts are found to be empty. By comparison to other concepts (e.g., analogical, same super-type, morphological similarity, etc.) these holes can be filled. If knowledge cannot be deduced via introspective analysis, a second mechanism, *question-asking* is used. A question is asked of the user and the provided information incorporated into the concepts. The third mechanism is *Experimentation*. This mechanism experiments with concepts to discover their causal relationships. By manipulating an episode in which the concept of interest is found, relationships can be discerned. These three mechanisms are further described in section 8.

These different categories of processes elaborate a hierarchical-like structure emanating from the criterion to encompass the information. Elaboration results in the creation of new concepts and instantiation of concepts already contained in the knowledge base. These concepts are placed in a system working space. The criterion and information to be structured are also placed there. The final structuring is a subset of concepts and relations in the working space.

5. Building Knowledge Structures

As described above, many types of processing are required. The building of a knowledge structuring system requires not only the execution of these processes, but the strategies required to apply them. An implicatory structuring may require strategies and mechanisms different from those in a classificatory structuring. The approach taken here for strategy representation and execution, and general system coordination is derived from production system architectures (Newell, 1973; Lenat & McDermott, 1977) and HEARSAY-II (Erman & Lesser, 1975; Erman, 1977).

Strategy information is represented as rules similar to meta-rules (Davis & Buchanan, 1977). At any instance, only a subset of all rules in the system are active -- their conditions can be inspected. We describe all active rules as being enclosed by a window. Only rules in the window are active. The right-hand side of a rule may add or remove rules from this window (activate, de-activate respectively) (Fox, 1977). Associated with the window is a set of memories to which information may be added, modified, or removed by the rules. A window and its associated memories comprise a *system*.

Rules may create new systems. The creation of a new system requires the specification of what rules are to be placed in the new window initially, what memories to be created, and what information placed in these memories. In addition, a limited amount of resources may be allocated for processing in the new system. Consumption of the resources causes a return to a prespecified system. A goal may be specified in addition to or instead of resources; attainment of the goal also causes a return. A system is either suspended or destroyed when it creates a new active system.

In addition to rules and systems, there exist a set of data-activated knowledge sources whose invocation is determined by their pre-conditions (See (Erman & Lesser, 1975; Erman, 1977) for a description of knowledge sources and their pre-conditions in HEARSAY-II). The rules in a system's window can activate or de-activate a knowledge source's pre-condition, thus controlling what knowledge sources can be invoked. During an active system's execution, the system's rules decide what knowledge sources to activate and the amount of resources to expend; they then initiate the data-directed processing. That is, once the knowledge sources are chosen and a cost-limit determined, the knowledge sources are executed in a HEARSAY-like data-directed fashion using data in the active system's memories[1], until the resources are consumed.

The motivation behind creating multiple systems is to facilitate attention focussing. The knowledge structuring system must be able to focus dynamically upon a few concepts with a subset of knowledge sources within the structure being built while suspending but maintaining its other interests. This is achieved through the creation of an active system with only those concepts in the memories and knowledge sources active.

The knowledge sources currently under development are:

Concept-Specification:	Further specifies slots within a concept to create a new, more specific concept.
Bottom-Up:	Creates abstractions of concepts using various methods.
Scout:	Searches for a chain of relations between two concepts.
Prune:	Removes concepts from the knowledge structure that do not play any signficant part (e.g., dead end inference chains).
Class:	Measures how well a concept acts as a classification concept in the knowledge structure.
Implicate:	Measures how well a concept acts as an implication concept in the knowledge structure.
Introspection:	Acquires knowledege by utilizing inference rules within the knowledge base.
Question-Asking:	Acquires knowledge by asking the user very specific questions.
Experimentation:	Acquires knowledge by creating and executing experiments upon episodes represented in the knowledge base.

[1]A knowledge source may add data not only to active memories, but to suspended systems' memories as well.

The application of these modules requires some sort of strategy. The following is an example of a simple strategy.

1. Expend x resources on concept specification focused on the criterion.

2. Expend y resources on bottom-up concept formation focused on the information units.

3. Expend z resources Scouting for paths between criterion derived concepts and information unit derived concepts.

4. Combine disjoint sub-structures (island-drive).

5. Prune concepts in knowledge structure.

6. Rate concepts as Classification (or Implication) nodes in the knowledge structure.

7. Apply 1-6 on portions of information and criterion not yet related.

During the running of the system, rule 1 would create a new active system with the criterion placed in a memory and concept-specification being the only knowledge source activated. The knowledge source would fire on the criterion and any concepts it derives from the criterion, creating a hierarchy of criterion-based concepts. The system would return to the initiating system once its resources were consumed.

6. Knowledge Representation

A key issue involved in creating a knowledge intensive system is the knowledge representation. Section 3 gives an idea of the diversity of information to be used. Arbitrary definitions at different levels of abstraction and ambiguity must be represented. Hierarchies of categories, transformations, actions, and causal relations are needed. The following is a brief list of knowledge types to be represented:

1. object, attribute, value triples.

2. structural relations.

3. causal relations.

4. time relations.

5. multiple levels of abstraction in both data and procedure.

6. type hierarchies.

In addition, we must accomodate and assimilate new information (Moore & Newell, 1973) (Simon, 1977).

The approach taken here is the melding of diverse approaches to knowledge representation and its use. To begin with, the primary unit of representation is the *Concept*. It is similar to Schema (Bartlett, 1932), Frames (Minsky, 1975), Beings (Lenat, 1975), Concepts (Lenat, 1976), and Units (Bobrow and Winograd, 1977). It combines Fillmore's (1968) Case Form approach, which appears in Schank's (1975) Conceptual Dependency representation, with a lexical level representation. The case form approach allows the representation of concepts at the level of deep structure, which allows comparison between concepts. On the other hand, lexical information is stored, enabling lexical comparisons and manipulation of lexical concepts. Each concept is *Typed*. The type provides the slots (facets, cases) that are in the concept (e.g., is-a inheritence), and they are filled with lexical information. There is no limit to the number of types. Each type is defined by another concept.

In most systems, the slots in a concept are system primitives. Flexibility in accomodation, assimilation, matching, and manipulation is allowed when slot names are themselves concepts. This idea is similar to what is found in Merlin (Moore and Newell, 1973) and AM (Lenat, 1976).

One can manipulate concepts as declarative representations: matching and manipulating concepts, building new concepts, etc. But there is the procedural aspect to this representation. The concept "sort" could be represented declaratively by stating a precondition, a list of elements, and post-condition, a sorted list of elements, along with other information like the type of elements. A procedural definition can also be attached to the "sort" concept. This allows the sort concept to be applied (in the Lisp sense) to a list of elements with the output of the application, a sorted list. Each concept has associated with it both a declarative and procedural description, where applicable.

In addition to the regular slots found in a concept, there are meta-descriptions. Meta-descriptions occur in two forms. The first are descriptions associated with each slot, which provide information about what fills the slot. The following are the standard slot meta-descriptions:

Restrict	restrict the types of information that may fill a slot.
Default	default filler for empty slots.
To-Fill	directs system in how to fill slot if empty.
If-Filled	initiates "side-effect" processing when slot is filled.

The second type of meta-description is concerned with the relationships between the slots, and information concerning the concept as a whole. For example, information as: to why the concept was created and by what knowledge source, what slots semantically modify or restrict other slots, what slots further specify other slots in a type hierarchy, or rules for specification or application. The following are the standard concept meta-descriptions:

Capply:	the procedural definition of the concept.
Restrict:	what slot restricts the specification of another.
Modify:	what slot modifies the interpretation of another.
Preclude:	what slots further define slots higher in the type hierarchy.
Elaborate:	rules specifying what slots to elaborate in concept-specification.
History:	why and by whom the concept was created and/or modified. Used to focus search.
Class:	what class the concept is a member of.
Fspec:	what concept this concept is a further specification of.

An important feature of any knowledge representation, largely ignored by most systems to date, is the storage of information necessary to optimize search and knowledge acquisition. Initially, the search and knowledge acquisition rules will be quite general. As experience is gained structuring information, the context sensitivity of the rules will be recognized and stronger heuristics constructed. This requires the storage and analysis of empirical data. Eeach concept will store records of its use in search and knowledge acquisition. This data will be analysed by various methods, some of which are:

- Each concept is composed of one or more types. All concepts of the same type can be analysed for commonalities in the empirical data. This is stored in the concept for that type. This commonality (abstraction) analysis can recurse up through the type hierarchy.

- We frequently find certain lines of reasoning (sequence of concepts) useful in the structuring process. Using the network model method described in (Fox & Hayes-Roth, 1976) these sequences can be discovered. Hence new search rules will be constructed.

7. Search and Context Maintenance

Search, as depicted in concept-specification or scouting knowledge sources, is a key element in the system's processing. Given a criterion and information to be structured, the system must search for interesting relationships between the two. In the survival example, search (concept-specification) was along the type hierarchy of cases in the criterion concept. The decision as to what path to explore from a concept is dependent upon the system's *Global Search Context*, and the concept's *Local Search Context*. Local search context is specified by rules attached to each concept and the restriction and modification meta-descriptions among slots.

Local context is dependent upon the concept under investigation. It is independent of the reasoning that led to the consideration of that concept. For example, the survivability example refers to the word "attack". This, in itself, is a concept. Attached to it may be a local search rule that points out that the instrument of the attack may be of interest. Also, the meta-description Restrict slot specifies that the instrument restricts the type of action. Therefore the instrument should be specified before the method of attack. The search could then investigate the concept(s) associated with the instrument case.

The need for a global search context in Speech Understanding Systems has been expressed by Lesser and Erman (1977). The decision as to what portions of a concept should be attended to is dependent upon how it relates to its global context. This context is not only defined by the sequence of concepts that led to the concept under consideration (i.e., its ancestors in a hierarchy) but by concepts in other parallel branches of the criterion elaboration hierarchy[1].

Two modes of global context construction occur in the survival example. The first is a side effect of the criterion elaboration by concept specification. "SUCCESSFULLY ATTACK DANGER AND FIND FOOD ON LAND" is a new concept created in the elaboration process. It displays the main concept of the original survivability criterion combined with the specifics of the particular elaboration of the criterion. The global context is incorporated during the concept's creation. The decision as to what portion of the concept to investigate is dependent upon its "importance" in this concept. The action (verb) is of importance partially due to the emphasis placed on it by the modifier "successfully" and it pivotal position in the concept. Hence, it is of interest and is investigated further. The global context is implicitly brought to bear during the decision.

The second mode relies upon the system dynamically building a context representation independent of the concept under investigation. Continuing the survival example, the first type of global context led to considering the concept ATTACK, then the local context said to investigate the "instrument" of attacking. The realm of possible instruments is vast. It can only be pruned by knowing the original context of the concept ATTACK. That is, the attack is on land and the attackee is possibly dangerous. Other information such as whether the attacker and attackee are animate would be useful. This information was not explicitly carried along in the search. At this point in the structuring process we are looking for relations, possibly causal in nature, between attack and the information to be structured (mammals).

8. Knowledge Acquistion

It is obvious from these examples, that the creation of a knowledge structure requires a large amount of domain knowledge. For example, the structuring of mammals based on survivability requires not only knowledge of physical characteristics but what part they play in the animals survival, e.g., a keen sense of smell for detecting an enemy. This knowledge is initially separate from the information to be structured. Not only is the amount of knowledge great, but the inferences drawn are complex. Suppose we were to design a system to structure information and/or knowledge, and provide it with a knowledge base and inference generator. One could never define, a priori, the knowledge necessary for such a task. Nor could one foresee the knowledge needed in other applications of the system. The quantity, diversity, and obscurity (to the system designer) of the knowledge, requires that the system dynamically acquire the knowledge relevant to the task.

An important problem immediately arises: How does a system focus its acquisition of knowledge? The solution is the *criterion*. It is the key piece of information used in rating what knowledge to acquire. The closer the relationship between knowledge and the criterion, the higher the rating. In a sense, the criterion biases the computation of the system.

During the search process, many slots (concepts) are found to be empty. One of the mechanism for filling those slots is *introspection*. This mechanism is implemented by attaching knowledge acquisition rules to slots in a concept, describing where in memory the proper information (other concepts) may be found. In addition to rules, deduction concepts and analogy mechanisms may be employed.

A second method of acquiring knowledge is *question-asking*. Slot related rules can be embedded to ask the user directed questions. The value of a question in acquiring relevant knowledge is dependent upon its specificity. A question's specificity depends on the concept's abstraction level (e.g., level in type hierarchy) and search context. The concept "action" may specify questions at the level of object, agent, instrument, etc., while the concept "attack" may have rules (combined with expected case filler types) specific enough to ask: "what weapon was used as the instrument of attacking." Utilizing the context mechanisms described in Section 7, detailed questions can be asked.

Another mechanism for guiding the formation of questions requires the ability to solve problems at various levels of abstraction. Consider the survival example at the point of trying to relate "attack" to one of the mammals. The system's knowledge base may not contain information specific enough to relate the two. That is, the present state of the knowledge base is not domain specific. It may know nothing about mammals attacking with claws, teeth, tails, or guns. But it may have an abstract description of the concept "attack", which may be defined as "the application of force to overcome another object." The analysis of force application could lead to: "movement or manipulation of objects." Continuing this analysis, the mammal may be abstractly related to "attack". The system could then ask specific questions based on this abstract analysis to elicit domain knowledge. For example: "can the arm of the mammal be viewed as an object that can apply force?", "Is the force applied great enough to overcome the opponent's force mechanisms?" etc. By building an abstract chain of inferences, relations, etc. that connect two concepts of interest, questions can be posed to elicit the domain specific information.

The sequences of abstract relations can also be built by analogy. If the system has an example of attacking in another domain, then by traveling up the type hierarchy, the attack example can be described at an abstract level. The sequence of abstract concepts can be used in question-asking in the same way that the abstract sequence is derived by problem-solving.

Experimentation is a third method of knowledge acquisition. It appears to be a very powerful method that has yet to be seriously investigated in the Artificial

[1] The process of elaborating the criterion does not result in a true hierarchy but a structure whose underlying form is hierarchical and contains "fuzzy" nodes. That is each node has inference paths (mostly dead ends) running off in many directions.

Intelligence community[1]. This mechanism experiments with episodes and concepts to discover their causal relationships. By manipulating an episode in which the concept of interest is found, relationships can be discerned. For example, if we are investigating the survival characteristics of a gazelle we may have an episode in which the mammal exhibits its survival abilities: "A gazelle is travelling through a forest. It is being stalked by a lion. The gazelle stops, lifts its head and runs away, escaping from the danger." This episode is sufficiently vague that it does not indicate the attributes of the gazelle that facilitated its recognition of and escape from danger. By experimenting with the episode, the interesting characteristics can be unveiled. If it is thought the gazelle's hearing allowed it to detect the lion then the episode can be rerun (either internally or by asking the user) and changed by putting ear-muffs on the gazelle. The outcome of the episode (gazelle's death) will inform the system whether hearing was a key factor. The inability to build and maintain an internal model sufficient to carry out experiments is not an impediment. The question-asking facility discussed above suffices. By posing the proper questions, the system user can carry out the actual experiment and relay the results. Hence a powerful question-asking ability lays the ground-work for experimentation.

9. Structure Evaluation

Given that the criterion determines the type of structure, it is also the criterion that is used to evaluate hypothesized structures. The question is: How is the criterion integrated into the evaluation process?

There are two types of evaluation techniques. The first is *constructive*. Constructive techniques are used in the actual construction of the knowledge structure. They are dependent upon the relations used. For example, if a class relationship is used, it may be based on the similarity of attributes between information units. The decision as to what attributes are diagnostic is based on the attribute's relation to the criterion. In many mammals, the existence of hair may not be a useful classification, relative to the survival criterion.

The inferences that are used to hypothesize possible structures are subject to evaluation. In most other systems, inferences have (at most) static ratings associated with them. The combining of information requires the combining and evaluation of these multiple inferences (Shortliffe, 1975; Duda & Hart, 1976; Hayes-Roth et al., 1977). The use of dynamically defined inference ratings (by relation to the criterion) provides us with the power we need to measure knowledge structures using different criteria.

The second evaluation technique is *operative*. It is an evaluation of a structure based on its actual performance. If a knowledge structure was created to model the university enrollment forecasting process, then its performance, i.e. predictability, would be an operative evaluation. Operative techniques can be *external* or *internal* to the system. In the external case, the system must produce a knowledge structure, give it to the user, then evaluate the performance results and change the it accordingly.

We call the internal case *experimentation*. Experimentation takes any structure, hypothesis, or relation and tests it in a closed, controlled environment. Under the systems direction, the environment and/or the hypothesis being tested (e.g., a gazelle detects danger by hearing it) can be changed. By various techniques, theories can be corroborated or rejected. In the survival of mammals example, an episode where a mammal is in a dangerous situation would be run and re-run to see what attributes of the mammal are pertinent for its survival. Placing the mammal near water, on a rocky hill, in day or night, all may be important environmental changes used to ferret out necessary attributes. Experimentation actually bridges two areas of concern. It is used as an evaluation technique, and as a method of knowledge acquisition. Each test increases the system's knowledge of the mammal.

10. Summary

We have introduced the concept of *Knowledge Structuring* as the process that generates *knowledge* by combining *information* previously unrelated, or by creating alternate structures for (previously structured) knowledge. The key factor in the knowledge structuring process is the *criterion*. The number of different potential structurings is large: its selection is guided by the criterion. Generality is attained by making the criterion a system parameter, resulting in an *Exo-Criterion System*.

Two types of problems, classification and implication have been examined using the knowledge structuring paradigm. They demonstrate the need for exo-criterion knowledge structuring systems.

A knowledge structuring system requires a robust knowledge representation. In our approach, we have built upon the labours of others. Interesting additions are the integration of empirical information for learning, the melding of the declarative and procedural approach to representation, and the use of concept meta-descriptions.

Of particular interest to this research is the study of knowledge structuring mechanisms. In particular, the focusing of search in large knowledge bases by means of the criterion. We believe that search is a basic mechanism, upon which the knowledge sources are dependent.

Criterion guided knowledge acquisition is the second knowledge structuring mechanism. This work investigates acquisition via introspection, question-asking, experimentation. Two of the underlying mechanisms are concept-specification for creating sub-criteria and (abstract) problem-solving as a basis for question-asking.

Lastly, it should be emphasized that the process of knowledge structuring and the integration of an exo-criterion is a necessary step in the evolution of Artificially Intelligent systems.

[1]The Molgen project (Stefik & Martin, 1977) is an attempt at the automation of planning genetic experiments. Both are similar in that they have to plan the experiment but they differ in that a knowledge structuring system attempts to extract dependency information implicitly contained in the episode. This requires the recognition of the portions of the episode that pertain to the current knowledge acquisition goal, the construction of an experimentation goal, and then constructing an experiment.

11. References

Bartlett F.C., (1932), Remembering, Cambridge: Cambridge University Press.

Bobrow D., and T. Winograd, (1977), "KRL: Knowledge Representation Language," Cognitive Science. Vol 1, No. 1, 1977.

Buchanan B. and T. Mitchell, (1977), "Model Directed Learning of Production Rules," Fifth International Joint Conference on Artificial Intelligence, Cambridge MA.

Davis R. and B. Buchanan, (1977), "Meta-level Knowledge: Overview and Applications," Fifthe International Joint Conference on Artificial Intelligence, Cambridge MA. August 1977.

Duda R.O., P.E. Hart, and N.J. Nilsson, (1976), "Subjective Bayesian methods for rule-based inference systems," Proc. of NCC.

Erman L.D. and V. Lesser, (1975), "A Multi-level organization for Problem- Solving using many Diverse Cooperating Sources of Knowledge," Fourth International Joint Conference on Artificial Intelligence, Tiblissi, USSR.

Erman L.D., (1977), "A Functional Description of the HEARSAY-II System." Proc. 1977 IEEE Inter. Conf. on ASSP, Hartford, CT, May, 1977.

Fikes R.E., and N.J. Nilsson, (1971), "Strips: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2,pp 189-208.

Fillmore C.J., (1968), "A Case For Case," In Universals in Linguistic Theory, Bach and Harris (Eds.), Chicago: Holt, Rinehart & Winston Inc.

Fox M.S. and F. Hayes-Roth, (1976), "Approximation Techniques for the Learning of Sequential Patterns," Third Int. Joint Conf. on Pattern Recognition, Coronado Calif., Nov. 1976.

Fox M., (1977), "Cluster Rules", Internal Memo, The RAND Corporation, Santa Monica, June 1977.

Fox M.S. and R. Reddy, (1977), "Knowledge Guided Learning of Structural Descriptions," Fifth International Joint Conference of Artificial Intelligence, Cambridge, MA, August, 1977.

Hayes-Roth F., (1976), "Patterns of induction and associated knowledge acquisition algorithms," Tech. Report, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA, May 1976.

Hayes-Roth F., and V. Lesser, (1977), "Focus of Attention in the HEARSAY-II Speech Understand System," Fifth Int. Joint Conf. on Aritificial Intelligence, Cambridge, MA, Aug. 1977.

Hayes-Roth F., V. Lesser, D.J. Mostow, and L.D. Erman, (1977), "Policies for Rating Hypotheses, Halting, and Selecting a Solution in HEARSAY-II," In Summary of the CMU Five-year ARPA effort in Speech Understanding Research, Tech. Rep., Computer Science Department Carnegie-Mellon University, Sept. 1977.

Lenat D., (1975), "BEINGS," Fourth International Joint Conference on Artificial Intelligence, Tiblisi, USSR.

Lenat, D., (1976), "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," (Ph.D. Thesis) Computer Science Dept., Stanford University.

Lesser V.R., and L.D. Erman, (1972), "A Retrospective View of the HEARSAY-II Architecture, Fifth International Conference on Artificial Intelligence, Cambridge MA.

Michalski R.S., (1974), "Learning by inductive inference," NATO Advanced Study Institute on Computer Oriented Learning Processes, Bonas, France.

Minsky M., (1975), "A Framework for Representing Knowledge", In The Psychology of Computer Vision", P. Winston (Ed.), New York: McGraw-Hill.

Moore J., and A. Newell, (1973), "How can Merlin Understand?," In Knowledge and Cognition, L. Gregg (Ed.), Maryland: Lawrence Erlbaum Ass.

Newell A., and H.A. Simon, (1963), "GPS: A Program that Simulates Human Thought", in Computers and Thought, E.A. Feigenbaum and J. Feldman (Eds.), New York: McGraw Hill.

Newell A., and H.A. Simon, (1972), Human Problem Solving, Englewood Cliffs, N.J.: Prentice Hall.

Newell A., (1973), "Production Systems: Models of Control Structures," In Visual Information Processing, W.C. Chase (Ed.), New York: Academic Press.

Sacerdoti E.D., (1975), A Structure for Plans as Behaviour, (Ph.D. Thesis), Computer Science Dept., Stanford University.

Samuel A., (1963), "Some Studies in Machine Learning using the Game of Checkers," In Computers and Thought, E. Feigenbaum and J. Feldman (Eds.), New York: McGraw-Hill Book Co.

Schank R., (1976), "The Structure of Episodic Memory," In Representation and Understanding, D. Bobrow and A.Collins (Eds.), New York: Academic Press.

Shortliffe E.H., (1975), Computer-Based Medical Consultations: MYCIN, New York: American Elsevier.

Simon H.A., (1977), Artificial Intelligence Systems that Understand, Fifth International Joint Conference on Artificial Intelligence, Cambridge MA.

Sokal R., and P. Sneath, (1962), Principles of Numerical Taxonomy, San Francisco: W.H. Freeman.

Stefik M. and N. Martin, (1977), "A Review of Knowledge Based Problem Solving as a Basis for a Genetics Experiment Designing System", Tech. Rep., Stanford University.

Winston P., (1970), "Learning Structural Descriptions from Examples," Tech. Report AI TR-231, MIT AI Lab., Cambridge, Mass..

RE-REPRESENTING TEXTBOOK PHYSICS PROBLEMS[1]

John McDermott
Computer Science Department
Carnegie-Mellon University

and

Jill H. Larkin
Physics Department
and Group in Science and Mathematics Education
University of California, Berkeley

Abstract. Only limited attention has been given to developing AI systems that can represent for themselves the problems that they are asked to solve. PH-632, the system discussed in this paper, is a production system that models the behavior of an expert physicist in solving textbook physics problems; as with other skilled textbook physics problem solvers, a significant amount of its effort is directed toward problem representation. The system is driven by a pictorial representation of a problem; it extensively reworks the representation before generating the equations that enable it to solve the problem.

I. INTRODUCTION

Clearly the way in which a problem is represented is a significant factor in the ease with which it can be solved [Newell and Simon, 1972; Simon, 1975]; and for the most part, when an AI system is designed, the designers expend considerable effort looking for an optimal problem representation scheme. In this paper, we describe a system, PH-632, that can represent problems for itself in a way that facilitates its subsequent work on those problems. The system models the performance of a skilled physicist on simple mechanics problems. We will not be concerned here with the psychological validity of our model (see Larkin [1977a; 1977b]); rather, we will assume that our model captures, at least grossly, the basic problem solving strategy of the expert and will be concerned primarily with the utility of the problem representations implied by the strategy.

A skilled physicist solving a simple textbook problem ordinarily makes use a sequence of four representations: Given an initial statement of the problem in English, the expert re-represents the problem as a picture; the picture contains just those objects mentioned in the problem statement and displays their relationships to one another. Then the expert develops a third representation consisting exclusively of "physics entities" (eg, systems, states, forces, energies). Finally, the expert represents the problem as a set of equations whose solution will yield the answer to the problem. While it is obvious that the first and last of these representations are necessary in order for a problem to be solved, the utility of the two intermediate representations may be less clear. In this paper we will focus on two questions: (1) In what ways do the intermediate representations facilitate problem solving? (2) What knowledge is required to construct and then make effective use the second of these intermediate representations?

Several individuals are engaged in research that parallels our own. Hayes and Simon's UNDERSTAND system [1974] models the behavior of humans in the task of problem representation, and Novak [1976; 1977] and de Kleer [1977] have each developed a physics problem solving system that constructs multiple problem representations. Novak's system, ISSAC, which inputs English text stating a problem and then solves the problem, is primarily a study in natural language understanding. Most of his effort was spent in showing how a statement of a physics problem can be mapped into a "pictorial" encoding composed of instantiated "canonical object frames"; by abstracting from the irrelevant details in the problem statement, his system is able to put the problem in a form that makes solving relatively straightforward. In de Kleer's system, NEWTON, knowledge is represented in a way that enables the system to attack problems of varying difficulty with different problem solving strategies. The system first attempts to solve a problem using "qualitative" knowledge; if the problem cannot be solved using only qualitative knowledge, the system uses its "quantitative" knowledge. PH-632 is given a pictorial representation similar to that generated by Novak's system; it constructs a qualitative representation -- a second intermediate representation -- that greatly simplifies its subsequent quantitative analysis.

1 This work was supported in part by the Defense Advanced Research Projects Agency (F44620-73-C-0074) and is monitored by the Air Force Office of Scientific Research.

In the next section, we consider in more detail the different problem representations used by a skilled physicist and provide some justification for the two intermediate representations. In the third section, we describe how knowledge is represented in our system and indicate how the system uses its knowledge to construct and use the qualitative representation. In the fourth section we examine in some detail the behavior of our system on a sample problem. Finally, in the fifth section, we discuss two of the weaknesses of our system as currently implemented: its inability to make effective use of its knowledge when it encounters problems of an unfamiliar type and its inability to refine its knowledge on the basis of its experience.

II. THE PROBLEM REPRESENTATION STRATEGY

As we indicated in the previous section, an expert physicist typically makes use of four different representations as he solves a textbook physics problem. To illustrate these representations we will use a problem taken from Halliday and Resnick [1966, pp. 126-127]. The first representation is the one given in the book:

> Block B in Figure 9 weighs 160 lbs. The coefficient of static friction between block and table is 0.25. Find the maximum weight of block A for which the system will be in equilibrium.

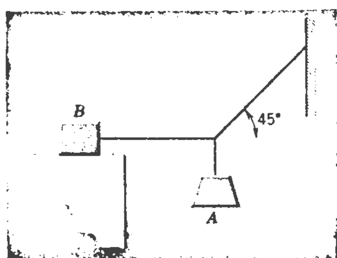


Figure 9

If the problem statement were not accompanied by a picture, then the expert would construct, as a second representation, his own picture displaying the essential elements of the problem situation and then would make a few notes on his sketch indicating relevant quantities and their values. Since a picture is already provided for this problem, all that the expert does in this case is copy the picture and add a few notes. The expert would produce a sketch like that in Figure 9a.

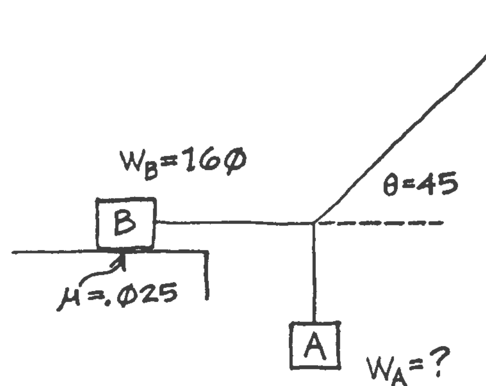


Figure 9a

The expert would then engage in a qualitative analysis of the problem and this would result in his constructing a third representation containing only "physics entities". The expert would begin by selecting an approach to the problem (ie, a set of physics principles); for this problem, force principles would be the most likely candidate. Then he might make some inferences about block B (that it is acted on by four forces: tension to the right, friction to the left, gravity down, normal force up). Because block B is an instance of a familiar system type (a block at rest on a horizontal surface), the expert would probably not add anything to his sketch.[2] But the fact that the tension force on block B cannot be directly related to known or desired quantities would guide his attention to a second system involving this tension force -- the juncture of the three strings. This system is sufficiently complex that he would probably draw a "free-body" diagram of it. This would result in a sketch like that in Figure 9b.

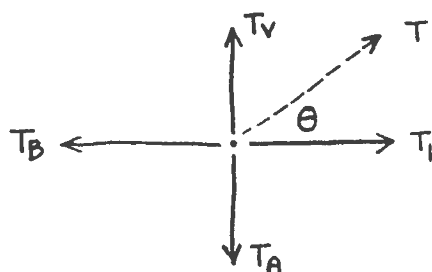


Figure 9b

---

2 It may be that the expert actually postpones his analysis of familiar systems until he begins to generate equations; we think it is more likely, however, that the systems are analyzed, but not drawn because the information inferred is easily remembered. We will return to this question in section IV when we describe how PH-632 deals with familiar systems.

The free-body diagram shows the directions of the two tension forces (T_A and T_B) due to the strings attached to blocks A and B, as well as the horizontal and vertical components (T_h and T_v) of the diagonal tension force due to the string attached to the wall. The fact that the vertical component, T_v, cannot be directly related to known or desired quantities would guide the expert's attention to block A; but again, as with block B, no free-body diagram would be drawn since block A is an instance of a familiar system type (a hanging block).

Finally the expert would construct a fourth representation of the problem by writing a set of equations; each of the equations below would be part of this quantitative representation:

Because forces balance on the juncture:

$$T_h = T_B; \qquad T_v = T_A$$

Because block B is at rest:

$$T_B = f$$

where f is the frictional force on block B. Combining the preceding, and relating the component forces T_v and T_h to the magnitude of the total tension force T due to the slanted string:

$$f = T(\cos\theta); \qquad W_A = T(\sin\theta)$$

where W_A is the weight of block A. Using the fact that θ is 45 degrees, and f equals the coefficient of static friction, μ, times the normal force which here is equal to the weight of B:

$$W_A = \mu W_B$$

$$W_A = 40 \text{ lb}$$

Skilled physicists almost invariably use all four of the representations illustrated above (the English problem statement, the labeled sketch, the sketch containing physics entities, and the equations). Our interest is in the utility of the two intermediate representations. It is not difficult to infer plausible reasons for the fact that the expert re-represents the problem pictorially. Much of the difficulty with mechanics problems lies in understanding the spatial relationships of the objects involved; a picture provides a way of representing this information compactly -- so that all of the information necessary for solving the problem is immediately available. Moreover, much of the information that is given in a typical English problem statement is irrelevant; in mechanics problems only a few attributes of objects (eg, mass, orientation, texture of surfaces) are of interest. Thus, the pictures that are drawn can be highly stylized -- a small set of symbols can be used to represent all of the objects involved.

But why does the expert spend a significant amount of his time constructing what we are calling a qualitative representation? There are at least three reasons: First, by doing so the expert can quickly determine the appropriateness of a particular approach to a problem. After constructing a free-body diagram, for example, he can check whether what he knows about the motion of a system is consistent with the forces acting on it. Secondly, before a mechanics problem can be solved, the forces or energies at work must be discovered and somehow represented. Since the spatial relationships among the forces or energies always has significance and since the force or energy information can be represented with a small set of symbols, a highly stylized pictorial representation of this knowledge is appropriate. Third, the problem can be attacked in a way that minimizes the possibility of the expert becoming confused or distracted. By decoupling the discovery of the forces or energies from the generation of equations, the expert reduces the amount of information that he must attend to at any one time; after adding force or energy information to his sketch, he can forget about it since it is in a semi-permanent and easily retrievable form.

III. PH-632: REPRESENTATIONAL ISSUES

In this section we turn from general comments about how skilled physicists solve mechanics problems to a consideration of the representational issues that arise when one tries to implement a system that can use the experts' problem solving strategy. After briefly discussing the way in which problems given to PH-632 are encoded, we consider, in some detail, the way in which PH-632's knowledge of physics is represented.

PH-632 is provided with an encoding of the picture of the problem it is to solve. Though it does not itself generate the pictorial representation (ie, it is not a natural language understanding system), Novak's work has shown how such a representation can be constructed. Using the labeled sketch, PH-632 constructs a qualitative representation and then generates the necessary equations and solves them. The problem description that PH-632 is given is encoded as sets of attribute value pairs. The description consists of three kinds of entities: (1) **objects**, (2) **contacts** between pairs of objects, and (3) **notes** containing quantitative information about the objects. These entities contain only that knowledge that could be generated by a naive perceiver (ie, by a perceptual system with no knowledge of physics).

Associated with each object is an indication of whether it has mass, whether it is moving (and if so, a description of the motion), what other objects it is in contact with and the direction of the contact, and its canonical type (eg, block, string, spring). For the

sample problem given above, PH-632 is given the following description of block B:

NAME	BLOK-B
TYPE	OBJECT
SUBTYPE	BLOCK
MASS	YES
MOTION	NO
CONTACT	(RIGHT STRNG1 CNTCT1)
	(DOWN TABLE CNTCT2)

Block B is in contact to the right with the object named STRNG1; information about the nature of this contact is contained in the contact entity, CNTCT1. CNTCT2 contains information about the contact between block A and the object, TABLE, which is underneath it.

Associated with each contact are the names of the two objects involved, the canonical type of the contact (eg, hook, surface), and for surface contacts, the orientation of the objects and their texture (eg, rough, smooth). PH-632 is given the following description of CNTCT2:

NAME	CNTCT2
TYPE	CONTACT
SUBTYPE	SURFACE
OBJ1	(BLOK-B BLOCK)
OBJ2	(TABLE SURFACE)
QUALITY	ROUGH
ORIENTATION	(RIGHT LEFT)

CNTCT2 is the contact between the two objects, block B (whose subtype is block) and TABLE (whose subtype is surface). The texture of the two adjacent surfaces is rough and their orientation is horizontal.

Associated with the notes is the knowledge of what is desired, what is given, and the value (if known) of each given. PH-632 is provided with the following description of what is given and what is desired in problem 9:

NAME	PRB9
TYPE	NOTES
DESIRED	(MASS BLOK-A)
GIVEN	(MASS BLOK-B)
	(ANGLE STRNG2)
	(MS CNTCT2)
CRITICAL	(MASS BLOK-A MOTION)
MASS	(BLOK-A M<A>)
	(BLOK-B 160)
ANGLE	(STRNG2 45)
MS	(CNTCT2 0.25)

The mass of block A is what is desired. The mass of block B, the angle of STRNG2 (the string attached to the wall), and the coefficient of static friction between block B and the table are all given.

In chosing a scheme for representing PH-632's knowledge of how to solve mechanics problems, we were guided by three aspects of the behavior of the human expert. First, as we have already indicated, the expert makes extensive use of pencil and paper in all stages of his problem solving. His set of sketches is a semi-permanent, auxiliary memory containing chunks of information that can be quickly accessed. Presumably the expert makes use of this memory in order to limit the amount of information that he has to attend to at any one time. Secondly, the expert appears to have his knowledge organized hierarchically. He takes the same general approach to all textbook mechanics problems; ie, he selects a set of physics principles, constructs physics entities, and then generates equations. Within each of these stages there is considerable consistency across problems of the same type; if for example, he is using force principles during his qualitative analysis, he will consider each object in the picture, looking for tension forces and frictional forces that act on that object. Third, the expert's choice of what to do at any given moment appears to be almost completely dependent on the peculiarities of the particular problem being considered. Though the ratio of knowledge that the expert has available to knowledge he uses is very high, the knowledge used is ordinarily just the knowledge needed for the problem at hand. For reasons that we will develop below, we decided that PH-632 should be implemented as a **production system** [Newell and Simon, 1972; Newell, 1973; Waterman and Hayes-Roth, 1978]; the particular production system architecture that we have used is called OPS2 [Forgy and McDermott, 1977; Newell, 1977; McDermott, 1978].

An OPS2 production system consists of a collection of productions held in **production memory** and a collection of assertions held in **working memory**. A production is a conditional statement composed of zero or more condition elements and zero or more action elements. Condition elements are templates; when each can be matched by an element in working memory, the production containing them is said to be instantiated. An instantiation is an ordered pair of a production and the elements from working memory that satisfy the conditions of the production. The production system interpreter operates within a control framework called the recognize-act cycle. In recognition, it finds the instantiations to be executed, and in action, executes one of them, performing whatever actions occur in the action side of the production. The recognize-act cycle is repeated until either no production can be instantiated or an action element explicitly stops the processing. Recognition can be divided into match and conflict resolution. In match, the interpreter finds the conflict set, the set of all instantiations of productions that are satisfied on the current cycle; OPS2 is implemented in such a way that the time needed to compute the conflict set is essentially independent of the size of production memory (see Forgy [1977]). In conflict resolution, it selects (on the basis of a few simple rules) one instantiation to execute (see McDermott and Forgy [1978]). The actions that can be performed include

adding elements to and deleting elements from working memory and building new productions composed of elements in working memory. In addition, operations can be performed on a **scratch pad memory** containing the description of the physics problem being worked on. Given the name of an entity, the system can view it; this results in the description of that entity being deposited in working memory. The system can scan for an entity satisfying a partial description; if it finds such an entity, the name of that entity is deposited in working memory. The system can also sketch new entities on its scratch pad memory and modify existing entities.

PH-632's production memory contains about 300 productions. With a few exceptions, each of these productions has as one of its condition elements the description of a goal; the set of productions that have the same goal condition element comprise a **method**. PH-632 has about 40 methods; some of these consist of only a few productions; others consist of 20 or 30 productions. By organizing the system's knowledge in this way, we effectively confine its attention to just that knowledge which is associated with the goals that it has not yet achieved. There is, of course, a danger in doing this: the system may have knowledge somewhere in production memory that is relevant to a particular goal but inaccessible because it is not associated with the system's current goal. We have guarded against this by organizing the methods hierarchically. The system has a few very general methods that generate subgoals; the methods sensitive to these goals generate more specific subgoals, and so on. Since knowledge can be put into a method at any level of this hierarchy, knowledge that has relevance to many different methods is simply associated with a very general method. Knowledge that the system has that is relevant to almost all of its methods is associated with a method whose productions have no goal condition element.

Our choice of a representation for PH-632's knowledge has proved adequate; it has enabled the system to construct and make effective use of both a qualitative and a quantitative representation. As we mentioned, our choice was guided by three aspects of the behavior of the human expert. First, like the human expert, our system makes extensive use of its scratch pad memory. Since PH-632 can quickly access the information in this memory, it can reserve working memory for information that is currently of interest. This implies that the only productions that are satisfied on a particular cycle are ones that are relevant to the current situation. When some entity in the scratch pad memory needs to be attended to, PH-632 can view that entity; the description of this entity will then drive the processing for a while -- until some other entity needs to be attended to. Secondly, our method hierarchy provides the stabilizing influence that is necessary in order for the system to make effective use of its multiple representations in the face of considerable variation among problems. Since the system is always operating within the context of some goal, the particular subgoals that it can generate are highly constrained; it can generate only those subgoals that can be of help in achieving the current goal. By associating knowledge with the method whose level of generality encompasses all expected uses of the piece of knowledge, the system has access to all of the knowledge that might be useful in any situation. Finally, the recognize-act cycle provides a control structure that makes the system continually responsive to the current state of the picture. Although methods impose a structure on production memory, the methods themselves have no internal structure. A method is just a collection of productions; each production is an autonomous rule indicating what needs to be remembered or done within the context of a particular goal given some aspect of the picture that is currently of interest. When a method is being used, it is ordinarily the case that several of that method's productions fire before the goal is achieved. But the particular sequence of firings is not written into the method; it is determined by the knowledge in working memory indicating what is currently of interest.

IV. PH-632: PERFORMANCE

In this section we will describe the behavior of PH-632 on the sample problem described above. We will focus on those aspects of the system's behavior that show the advantages of the qualitative representation. After describing how the system attacks and solves the sample problem, we will re-examine the advantages of using a qualitative representation in more detail than we did in section II.

The first thing that PH-632 does when it is given a problem is select an approach to try. For problem 9, it chooses to try force principles.[3] Given this choice, it scans the picture for an object with mass and finds block B. Since it recognizes block B as a familiar system type, it builds a production associating with block B the knowledge of what forces act on that system. In other words, since block B is an object with mass on a rough, horizontal surface, it stores in production memory the knowledge that block B has four forces acting on it -- a tension force, a frictional force, a normal force, and a gravitational force. From this point on, whenever it looks at block B, along with seeing the attributes of block B, it will be reminded that these four forces act on block B. Thus the effect of its quick analysis of block B, is essentially the same as its more extended analysis of less familiar systems; instead of subsequently referring to a free body diagram in order to remind

3 Since the system's knowledge is currently limited to mechanics, its choice is between force principles and energy principles.

itself of this system information, it simply recalls the information stored in its production memory. PH-632 could, of course, have drawn a free body diagram rather than storing the information in production memory; we opted for production building because experts seldom generate free body diagrams for simple systems. At the same time that it builds the production, it deposits in working memory the knowledge that the tension force is potentially problematic, ie, that it is problematic unless it can somehow be related to the givens in the problem. PH-632 then looks for a system which interacts with block B that can account for this tension force.

It finds the juncture of the three strings (the only system that interacts with block B). Since PH-632 does not recognize this system as a familiar type, it "draws" on its scratch pad memory a **system** entity (a free body diagram). Before trying to discover the forces acting on this system, it constructs the following description:

NAME	I0011
TYPE	SYSTEM
JUNCTURE	STRNG3
	STRNG2
	STRNG1
PREF-DIR	DOWN
	RIGHT

The system attribute PREF-DIR (preferred-direction) indicates that the forces acting in that direction and the opposite one are the forces that PH-632 should concern itself with first; in this case (because the system is a juncture), both horizontal and vertical forces are equally significant.

After defining the system entity, PH-632 determines what forces are acting on the system in the preferred direction. When it infers a force, it creates a **force** entity and also adds a description of the force to the system entity. Here, the first force that PH-632 finds is the force due to STRNG2 (the string attached to the wall). The entity that it creates looks like this:

NAME	I0012
TYPE	FORCE
ON	I0010
FROM	STRNG2
SUBTYPE	TENSION
DIR	(UP RIGHT)

Whenever PH-632 finds a force that is neither parallel nor perpendicular to the preferred direction, it creates two additional force entities that are the horizontal and vertical components of that force. Thus, in this case, two component force entities will be created as well as force entities for the tension forces along the other two strings. After PH-632 has found all five of the forces in the preferred directions, the system entity looks like this:

NAME	I0011
TYPE	SYSTEM
JUNCTURE	STRNG3
	STRNG2
	STRNG1
PREF-DIR	DOWN
	RIGHT
FORCE	(TENSION LEFT I0012)
	(TENSION (UP RIGHT) I0013)
	(COMPONENT UP I0014)
	(COMPONENT RIGHT I0015)
	(TENSION DOWN I0016)

Each of the forces is identified by type, direction, and name.

After the system entity has been extended to include all of the forces parallel to the preferred directions, PH-632 checks for anomolies (in this case for inconsistencies between forces and motion). In problem 9, all systems are at rest and each force is balanced by at least one force in the opposite direction. Thus PH-632 concludes that the force information that it has generated is plausible and continues with its analysis. The second stage in the analysis involves checking to make sure that its subsequent quantitative analysis will be feasible. Specifically, it checks whether the forces generated can be related to quantities given in the problem statement. PH-632 first checks to make sure that the component forces are unproblematic. It does this by looking at the notes entity to see whether the angle of STRNG2 is given; since it is, the component forces are unproblematic. There remain the two tension forces due to STRNG1 and STRNG3. The force due to STRNG1 is discovered to be unproblematic since it can be traced back to block B. Thus the only potentially problematic force is the tension force due to STRNG3. PH-632 looks for a system which interacts with the juncture that can account for this tension force.

It finds block A. As with block B, it recognizes block A as a familiar system type (a hanging block) and so it builds a production containing the knowledge that block A has two forces acting on it -- a tension force and a gravitational force. Of these, only the tension force is potentially problematic, and it is quickly discovered to be unproblematic since it can be traced back to the juncture. Thus, all three of the systems in the problem have been considered, and all of the forces have been accounted for.

Although PH-632 often concludes its qualitative analysis at this point, it performs a second test for anomolies in the common case of a problem about a "critical value" (a value characteristic of some change of behavior in the problem situation). In problem 9, for example, the quantity desired is "the maximum weight of block A for which the system will be in equilibrium", ie, the weight that is critical in determining whether or not the blocks move. Thus PH-632 makes one final check on the adequacy of the

approach it is trying. It considers two cases: block A with a very small mass, and block A with a very large mass. It reasons that if the mass of block A is small, then the force it exerts on the juncture will be small, and the force that the juncture exerts on block B will be small. Since the tension force on block B is opposed by a frictional force, if the tension force is small, block B will be at rest. But if block B is at rest, block A will be at rest. A similar chain of reasoning enables it to conclude that if the mass of block A is large, block A will be in motion.

At this point, PH-632 is sufficiently certain that the force principles approach to the problem will work that it begins its quantitative analysis, a process that is straightforward because the forces acting on each of the systems have already been found. PH-632 scans the picture for a system; it finds the free body diagram for the juncture and so starts with that. Using the information in the free body diagram, it generates the following equation:

```
NAME        10022
TYPE        NOTES
SYSTEM      (10011)
EQUATION    (+ (COMPONENT RIGHT 10015)
             - (TENSION LEFT 10012) = 0)
```

Then, using both the free body diagram and information given in the notes, it replaces, in so far as it can, quantities in the equation that are neither known nor desired with known quantities. In this case, all it can do is replace the component force. It does this by rewriting the equation:

```
NAME        10022
TYPE        NOTES
SYSTEM      (10011)
EQUATION    (+ (COS 45)
                   * (TENSION (UP RIGHT) 10013)
             - (TENSION LEFT 10012) = 0)
```

It then generates a second equation using the free body diagram of the juncture and combines this equation with the first. After generating a third equation using the information about block B that is stored in its production memory, generating a fourth equation using the information it has about block A, and doing a little algebra, it is left with a single equation. Using the notes, it replaces the symbols in the equation with the actual values of the quantities, simplifies the equation, and is left with:

```
NAME        10022
TYPE        NOTES
SYSTEM      (BLOK-B BLOK-A 10011)
EQUATION    (+ (M<A>) = + (40.0) )
```

PH-632 takes about 400 cycles to solve this problem.

In section II, we gave three reasons why a strategy that includes qualitative analysis of the initial picture is preferable to one that goes directly to quantitative analysis. Having considered in some detail the behavior of PH-632 on a sample problem, we can now provide a more complete justification for this strategy.

The first reason we gave for using the strategy is that it provides a way of checking, at various stages during the qualitative analysis, the appropriateness of a particular set of physics principles for some problem. In problem 9, for example, PH-632, as it develops its understanding of each of the systems, tests for anomolies and then tests the feasibility of its approach. When PH-632 recognizes some object as a familiar system type, it knows that that object will pass both tests -- ie, having the requisite characteristics is part of what it means to be a familiar system type. For unfamiliar systems, the anomolies test is performed as soon as all of the forces parallel to a preferred direction have been generated; this test enables PH-632 to determine with very little analysis whether its description of the system is plausible. If this test fails, PH-632 can be reasonably certain that it has drawn an incorrect inference from the picture, or that the picture does not accurately represent the problem, or that the problem is the work of a malevolent jokester. The feasibility test is performed after PH-632 generates the remaining forces acting on the system; this test enables PH-632 to determine whether the information given in the problem statement (as encoded in the notes entity) is sufficient for solving the problem using force principles. If this test fails, PH-632 can be reasonably certain that force principles (at least by themselves) are inadequate. In some problems, information is given that enables a third test to be made -- a test that determines whether the interactions among the systems is plausible. In problem 9, for example, the problem statement implies that the size of the mass of block A determines whether or not block A will move. To test whether the qualitative representation supports this implication, PH-632 examines the interactions among the three systems. As with the first test, if this test fails, PH-632 can be reasonably certain that there is something the matter with its representation of the problem.

The second reason for using the strategy is that by representing the physics entities (eg, systems and forces) pictorially, PH-632 has easy access to precisely the information that it needs to check the plausibility of the system interactions during its qualitative analysis and to generate equations during its quantitative analysis. When PH-632 checks, in problem 9, to determine whether the weight of block A affects its motion, it first views block A, sees that block A interacts with the juncture, then sees that the juncture interacts with block B, and sees that there is a frictional force on block B. This is essentially all that it has to do to determine that the systems interact in a plausible fashion. Likewise, when it generates equations, it simply views a system, writes

an equation containing the forces that act on the system in a preferred direction, replaces the forces that are neither given nor desired with other quantities, and then views another system.

The third reason for using the strategy is that it helps keep PH-632 from becoming confused or distracted. In problem 9, whether or not a qualitative representation is constructed, the systems and the forces acting on them have to be found. Thus if PH-632 had to solve the problem without constructing physics entities, it would need to keep a large amount of information in its working memory. One likely consequence of this is that productions with competing or inconsistent goals would sometimes be satisfied at the same time. This would make it difficult for PH-632 to behave in a coherent way.

V. UNDERSTANDING AND LEARNING

We alluded in the introduction to two serious weaknesses of PH-632: its inability to make effective use of its knowledge when it encounters problems of an unfamiliar type and its inability to refine its knowledge on the basis of its experience. The first of these is a problem of understanding, the second, a problem of learning. Because PH-632's knowledge is represented procedurally (ie, because PH-632 has no idea of what it will do in a particular situation until it sees what it has done), it has no way of knowing what it is going to do next. One implication of this is that if it gets in an unfamiliar situation (one for which it has no rule for how to behave), it cannot do anything; since it has no idea of what its methods can accomplish, it cannot accommodate one of the methods it does have to the unfamiliar situation. If some of its knowledge were encoded declaratively (so that it could examine the consequences of using one or other of its methods), finding a method that might help in an unfamiliar situation, and finding a mapping between what it wants and what the method will provide would be relatively straightforward. But if extensive declarative encoding were provided, many of the advantages of using a procedural representation would be lost.

PH-632's difficulty with learning stems from the same source. One of our reasons for selecting a procedural representation was to facilitate learning; and in one important respect the representation is supportive. Since PH-632's behavior is strongly driven by the picture that it elaborates, each of its productions is autonomous in the sense that a production need not know anything about the other productions -- neither the conditions under which they will fire nor which productions have already fired. All that is necessary in order for the system to extend its knowledge so that it can cope with unfamiliar situations is that a few productions be added to production memory. If this were all that there were to learning, learning would be easy for PH-632. But a system that can learn must be able to do more than just extend the domain of its knowledge. Much of our work with PH-632 has involved revising the productions that it does have as its behavior shows these productions to be inadequate (eg, they fire when they should not, they do not fire when they should). When several productions are satisfied at the same time, that production will fire which is satisfied by the largest number of most recent elements. Thus in order for the system to acquire productions that partially replace or extend the domain of a particular production, it needs to know the conditions under which that production is evoked.[4] But that knowledge is not available in a procedural representation.

The approach that we are taking to overcome these two weaknesses involves adding a small amount of knowledge to the system that describes the methods that it has. When the system acquires a new method, knowledge about that method is associated with a production that contains the method's goal condition element. This knowledge should help to overcome both weaknesses. When the system encounters an unfamiliar situation, various method description productions can be evoked by having the system generate descriptions of goals that are similar to its current goal. When a method description production fires, a subset of the conditions which must be satisfied in order for the productions comprising that method to be evoked are deposited in working memory. This enables the system to establish a mapping between the conditions that actually obtain and the conditions that must obtain in order for one of its methods to be used. The system can also make use of the method description productions when it discovers that one of its productions is in need of refinement. A method description production, by providing the system with a subset of relevant conditions, will enable it to build a new production (to partially mask or to extend the faulty production); the new production will contain this subset of condition elements together with a few additional elements that distinguish it from the production that it refines.

VI. CONCLUDING REMARKS

Our discussion of PH-632 has focussed on two things. Our main concern has been with the way in which the system represents the problems that it is given. It starts with a "pictorial" description of a problem and then during the qualitative stage of its analysis constructs a representation that contains all

4 For more extended discussions of the problems that arise in trying to instruct production systems, see Davis [1977] and Rychener and Newell [1978].

of the knowledge that can be inferred from the initial picture using a set of relevant physics principles. Only after it has made the inferences explicit does it generate the equations that will enable it to solve the problem. There are three reasons for doing the qualitative analysis: (1) the system can quickly find an approach to the problem that is likely to succeed, (2) by making its inferences explicit, it has immediate access to all of the knowledge about a particular system that it needs at any given time during its subsequent analysis; and (3) by making the knowledge easily retrievable, the amount of knowledge that it has to attend to at any one time is small, and thus it is unlikely to become distracted or confused. We have also focussed on the way in which PH-632 represents its knowledge of how to solve mechanics problems. PH-632 is implemented as a production system. Since it stores its knowledge of the problem it is working on in its scratch pad memory in a pictorial form, it can focus its attention on a small part of the problem without having to worry that it is neglecting some important piece of information. Since it is concerned with only a small amount of information at any one time, the productions that fire are always relevant to its immediate interest. Thus its behavior is coherent, but also highly responsive to the peculiarities of the problem.

REFERENCES

[Davis, 1977]
Davis, R. Interactive transfer of expertise: acquisition of new inference rules. **IJCAI, 5**, 1977.

[de Kleer, 1977]
de Kleer, J. Multiple representations of knowledge in a mechanics problem solver. **IJCAI, 5, 1977.**

[Forgy, 1977]
Forgy, C. A production system monitor for parallel computers. Technical Report. Department of Computer Science, Carnegie-Mellon University, 1977.

[Forgy and McDermott, 1977]
Forgy, C. and J. McDermott. OPS, a domain-independent production system language. **IJCAI, 5**, 1977.

[Halliday and Resnick, 1966]
Halliday, D. and R. Resnick. **Physics**. John Wiley and Sons, 1966.

[Hayes and Simon, 1974]
Hayes, J. R. and H. A. Simon. Understanding complex task instructions. In D. Klahr (ed.), **Cognition and Instruction**. Lawrence Erlbaum, 1974.

[Larkin, 1977a]
Larkin, J. Problem solving in physics. Technical Report. Group in Science and Mathematics Education. University of California, Berkeley, 1977.

[Larkin, 1977b]
Larkin, J. Skilled problem solving in physics: a hierarchical planning model. Technical Report. Group in Science and Mathematics Education. University of California, Berkeley, 1977.

[McDermott, 1978]
McDermott, J. Some strengths of production system architectures. **Proceedings of NATO ASI on Structural/Process Theories of Complex Human Behavior**. Sijthoff, 1978 (forthcoming).

[McDermott and Forgy, 1978]
McDermott, J. and C. Forgy. Production system conflict resolution strategies. In D. Waterman and F. Hayes-Roth (eds.), **Pattern-Directed Inference Systems**. Academic Press, 1978 (forthcoming).

[Newell, 1973]
Newell, A. Production systems: models of control structures. In Chase, W. (ed.), **Visual Information Processing**. Academic Press, 1973.

[Newell, 1977]
Newell, A. Knowledge representation aspects of production systems. **IJCAI, 5**, 1977.

[Newell and Simon, 1972]
Newell, A. and H. A. Simon. **Human Problem Solving**. Prentice-Hall, 1972.

[Novak, 1976]
Novak, G. Computer understanding of physics problems stated in natural language. Technical Report. Department of Computer Sciences. University of Texas, Austin, 1976.

[Novak, 1977]
Novak, G. Representations of knowledge in a program for solving physics problems. **IJCAI, 5**, 1977.

[Rychener and Newell, 1978]
Rychener, M. and A. Newell. An instructable production system: initial design issues. In D. Waterman and F. Hayes-Roth (eds.), **Pattern-Directed Inference Systems**. Academic Press, 1978 (forthcoming).

[Simon, 1975]
Simon, H. A. The functional equivalence of problem solving skills. **Cognitive Psychology, VII**, 2, 1975.

[Waterman and Hayes-Roth, 1978]
Waterman, D. and F. Hayes-Roth (eds.). **Pattern-Directed Inference Systems**. Academic Press, 1978 (forthcoming).

REPRESENTING MATHEMATICAL KNOWLEDGE

Edwina R. Michener

Department of Mathematics
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract

This paper describes a representation for mathematical knowledge that includes illustrative and heuristic aspects as well as logical elements. Three item / relation pairs -- results / logical support, examples / constructional derivation, concepts / pedagogical ordering -- establish three representation spaces for a mathematical theory: Results-space, Examples-space, and Concepts-space. In addition to intra-space relations, this paper introduces the dual idea, an inter-space relation, which is used to represent an intuitve notion of association amongst items. An epistemological classification of items based on their roles in the understanding of mathematics is developed. A brief overview is included of how these ideas have been applied to teaching situations and the design of interactive environments for professional and neophyte mathematicians, and how they could be applied to programs, such as non-resolution theorem provers, that need to retrieve and manipulate mathematical knowledge.

1. Introduction

To understand mathematics one must know much more than the deductive details of definitions, axioms, and theorems and their proofs. A mathematician or student who is in command of his subject uses other resources such as: the stock of examples he finds useful, and their organization; certain rules of thumb or heuristics, some telling which are good ideas to try and others warning him of trouble; items noteworthy for their simplicity, ubiquity or generality. In studying and solving problems, he has a sense of what to use and when to use it, and what is worth remembering. He also has images of how all his knowledge hangs together. In short, he knows and uses a great deal more than purely logical deductive knowledge.

This paper is concerned with this other, often extra-logical, knowledge that is critical to understanding. It reports on a study to understand the understanding of mathematics, in order to improve how we learn, teach, and do mathematics [10]. The aim is to develop a conceptual framework in which to talk precisely about the knowledge actually involved in the understanding of mathematics. This goal is largely epistemological, but it is a prerequisite to trying to mechanize or support that understanding.

2. Examples, Results and Concepts

Examining how mathematicians use and explain their knowledge of mathematics makes it clear that there are at least three categories of information necessary to represent mathematical knowledge: *results* which contain the traditional logical aspects of mathematics, i.e., theorems and proofs; *examples*, which contain illustrative material; and *concepts*, which contain mathematical definitions and pieces of heuristic advice.

Just as results can be organized by the relation of *logical support* in which A --> B means that result A is used to prove result B, examples and concepts can also be structured by relations.

2.1 Examples-space

Examples can be ordered by the relation of *constructional derivation* in which A --> B means that example A is used to construct example B. For instance, the Cantor set is constructed from the unit interval by the process of "deleting middle thirds" [17]:

define sequence of sets by deleting middle thirds:

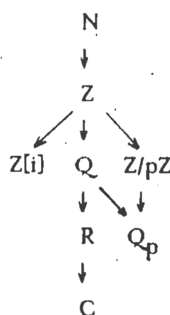
the unit interval:	0__________________ 1
delete (1/3, 2/3):	0_______( )_______ 1
delete (1/9, 2/9), (6/9, 7/9):	0__( )__( )__( )__ 1

etc.

limiting set is the Cantor set

"Cantor functions" and other "generalized" Cantor sets can also be constructed from the unit interval and the Cantor set [5], [16].

The following sequence of examples from arithmetic illustrates how a collection of examples can be organized according to its constructional relations. It starts with the natural numbers N. These beget the integers Z (by closure with respect to subtraction), which beget the rationals Q (by forming quotients), which beget the real numbers R (by completion of Cauchy sequences), which beget the complex numbers C (by algebraic closure). Many more examples needed in number theory, such as the Gaussian integers Z[i], the field of integers modulo a prime Z/pZ, and the p-adic numbers Q_p, can also be tied into this organization according to their constructional dependencies:



The p-adic numbers have arrows coming from both Q and Z/pZ since either can be used to construct Q_p. (Q_p can be constructed from Q by completion with respect to a metric just like the construction of the reals R from Q, or from Z/pZ by an algebraic construction involving "inverse limits" [3]). The point is that there are two constructional routes leading to Q_p, and thus a directed graph, not simply a directed tree, is needed to show the relations.

Some general properties of examples worth noticing are: (1) pictures are an integral part of examples; (2) constructions are like procedures; (3) the pictures need not be static, in fact those shown for the Cantor set are merely one frame from a sequence.

2.2 Concepts-space

Concepts include formal and informal ideas, that is, definitions and heuristics. A concept can be expressed either as a declarative statement, the familiar formulation of most mathematical definitions, or as a procedure or the result of a procedure. Some concepts are more naturally expressed in declarative form, while others, such as Gaussian elimination and Newton's method, are more naturally expressed as procedures. Some concepts can be expressed either way, such as the concept of "eigenvalue" which can be expressed as the λ of $Av=\lambda v$ [7] or as a root of the characteristic polynomial $\det(A - \lambda I) = 0$ [19].

Concepts can be structured by the pedagogical judgement that one should know about concept A before concept B; this relation is called *pedagogical ordering.* Sometimes it simply reflects the fact that concept A enters into the definition of concept B, at other times, expository tastes.

In this way, the three item / relation pairs -- *examples / constructional derivation, results / logical support, and concepts / pedagogical ordering* -- establish three representation spaces: *Examples-space, Results-space* and *Concepts-space.* They are best shown as directed graphs where the direction matches the predecessor-successor ordering inherent in the relations.

3. The Dual Idea

A theory item is related to other items in its representation space through the space's predecessor-successor relation. In addition it is related to items outside of its representation space. The *dual idea* concerns these inter-space relations. Specifically, *dual items* are defined as follows:

> The dual items of an example are: the ingredient concepts and results needed to discuss or construct it, and the concepts and results that are suggested by it.

> The dual items of a result consist of: the examples motivating it, the concepts needed to state and prove it, and the concepts and examples that are derived from it.
>
> The dual items of a concept are: the examples motivating it, the results laying the groundwork for it, and the examples illustrating it and the results proving things about it.

The dual idea thus associates to each item elements from the other two representation spaces:

$$dual(an\ example) = \{results\}, \{concepts\}$$
$$dual(a\ result) = \{examples\}, \{concepts\}$$
$$dual(a\ concept) = \{examples\}, \{results\}$$

The subset of examples in the dual set of an item is called the *examples-dual*, the subset of results, the *results-dual*, and the subset of concepts, the *concepts-dual*.

Dual items can also be categorized into sets of items which precede the item in the understanding or development of a theory, the *pre-dual*, which come after the item, the *post-dual*, and which have neither a strong "pre" or "post" flavor. To use Polya's words, some pre-dual items are "suggestive", and some post-dual items, "supportive" [15].

If two items share common dual items, they are said to be *related through the dual idea.* Dual relations are found throughout mathematics: concepts of countability and measure zero are related via the Cantor set; the examples of the real and the p-adic numbers are related via the concept of completion; Pythagoreas' Theorem and the Law of Cosines via an example of a right triangle; concepts of continuity and differentiability via the absolute value function; examples of ellipses, circles, parabolas, and hyperbolas via the concept of conic sections; concepts of stability, roots of unity and iteration via the example of $x^n=0$.

Relation via the dual idea is useful because it captures a way in which we associate items that are not closely related in the sense of the in-space relations, but which we easily link in our understanding. Dual relations tie the three representation spaces back together.

4. Epistemological Classes

Different items play different roles in our understanding. By grouping together items that serve similar and noteworthy functions, we establish *epistemological clases.* Since an item may serve more than one role in one's understanding, these classes are not necessarily disjoint.

4.1 Epistemological Classes of Examples

For instance, when we learn a theory for the first time, there are certain examples which we can understand immediately. These perspicuous examples get us started in a new subject by motivating basic definitions and results, and setting up the right kinds of intuitions. We call these *start-up examples.* The following is a summary of the use of the "circle and lines" start-up example in the study of curvature of plane curves as presented by Spivak [18]:

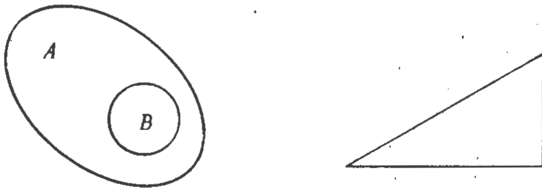
> We begin by considering circles and lines. We can agree that circles curve and that lines don't. Furthermore, small circles "curve more" than large circles. (This is consistent with our observations about lines, which are a limiting case.) We observe therefore that curvature is inversely related to the radius. So we say that for circles, the curvature is the reciprocal of the radius. Now what about more general plane curves? Well, we *lift* our circle-line definition to the general case by fitting circles onto the curves:

This simple example is a start-up example because it easily leads us to the formulation of the osculating circle definition of curvature. It provides a strong pictorial representation for curvature (circles) and an approach (the osculating circle) for calculating it.

A good start-up example should have the following properties: (1) it motivates fundamental concepts; (2) it can be understood by itself; (3) it is *projective* in the sense that it can be generalized; and (4) it provides a simple and suggestive picture.

Reference examples, another important class of examples, are examples that we refer to repeatedly throughout a theory. They provide a common node through which many results and concepts are linked via the dual idea. For instance, R^2, the real plane, is used as an example throughout all of real analysis; one invariably investigates concepts and results in this example to understand how they work. In number theory -- algebraic as well as elementary -- one always looks at the integers Z. Throughout his books, Polya refers to certain standard reference examples; for instance in the domain of plane geometry, he repeatedly uses such triangles as isosceles, equilateral, and isosceles-right triangles [15]. Reference examples are used as standard cases to check out one's understanding.

Model examples are generic examples. As paradigms they suggest to us the essence of a result or concept. For example, models for set inclusion and right triangles are:



Notice that the specific measurements in these pictures are unimportant; what counts is that they capture the essence of the situation.

Because of their generic nature, model examples are closely related to "without loss of generality" arguments. For instance, the model examples for conic sections are usually pictured as having their major axes aligned with the x- and y-axes (see any calculus book such as Thomas [20]); these diagrams are completely general because one can always use coordinate transformations (translation, rotation) to change variables so that the axes are indeed so aligned.

Model examples are flexible and manipulatable structures which summarize and suggest the expectations and assumptions about results and concepts. They usually must be fine-tuned to meet the specifics of a problem. For instance, the triangle model can be specialized to be a 3-4-5 right triangle, a 45-45-right or any other type of right triangle.

Counter-examples show a statement is not true or sharpen distinctions between concepts. Some counter-examples are also used as reference examples, such as the Cantor set which is used throughout measure theory after it's introduction as a counter-example to show that sets of measure zero need not be countable [8]. Other counter-examples are used once to establish a point and are not developed further; because these have a very limited use in the theory, our memory of them is often very short-lived.

In summary, some important epistemological classes of Examples-space are: *start-up examples, reference examples, model examples*, and *counter-examples.*

4.2 Epistemological Classes of Concepts

Concepts-space has two major epistemological classes in addition to the class of *definitions*: *mega-principles* and *counter-principles*. These other classes contain the heuristic advice that we use while working in a theory.

Mega-principles (*MP*'s) provide kernels of wisdom in the form of powerful suggestions or generally valid statements. They often state what is reasonable to expect. For instance, *Polynomial time means reasonable time* is a mega-principle from complexity theory. Others paraphrase definitions, such as *Continuity means you can draw the function without lifting your pencil*, a mega-principle familiar to most calculus students. The *MP(n=2)*: *Try the 2-by-2 case* offers powerful advice in the study of matrices. Another extremely useful piece of advice is *MP(0/1)* which suggests trying special cases involving only 0's and 1's. Thus some MP's provide imperatives or advice while others give an idea of what to expect. Mega-principles express broad "flavors" of a theory that are often remembered long after the details have been forgotten. Like model examples, they provide broad, suggestive descriptions and expectations.

Counter-principles (*CP*'s) are cautions against possible sources of blunders or troubles. For instance, everyone knows about the *CP: Watch out for division by 0*. The *dv* CP from calculus -- *when changing the variable in integration, don't forget to calculate the new differential:* $dv=v'(x)dx$ -- is a word of warning familiar to all calculus students. CP's are the distillations of many results, counter-examples, and failed attempts. Like counter-examples they add focus to our intuitions and serve to keep us from pursuing potentially unproductive lines of thought.

4.3 Epistemological Classes of Results

Results-space also has several epistemological classes of items, of which we mention only: *basic, culminating*, and *transitional* results.

Basic results establish elementary, but important, properties of concepts and examples. For example, the result: *λ is an eigenvalue of the matrix A iff det(A-λI)=0* is a result basic to the study of eigenvalues. It relates the *procedural* formulation (solving the characteristic equation) with the declarative definition of the eigenvalue concept. Other basic results link concepts with examples, such as: *The outer measure of an interval is its length.*

Culminating results are the goal results towards which the theory drives. To see if a result is a culminating result one asks, "if this result is omitted, has the main point of the theory been missed?" If the answer is yes, the result is a culminating result. For instance, the Fundamental Theorem of Algebra is a culminating result of high school algebra courses and the Fundamental Theorem of Calculus is a culminating result in calculus. Culminating results are often equivalency or classification results such as the theorem showing that all real vector spaces of a given dimension are isomorphic.

Transitional results provide logical stepping-stones or bridges between results. They are not as important as culminating results in one's understanding. Many results that are given the label *lemma* fall into this category.

There are many analogies between the epistemological classes: model examples, mega-principles, and culminating results are all important items within their categories; counter-examples and counter-principles serve a limiting function; basic results and start-up examples provide easy starting points in a theory.

5. The Representation

The foregoing sections have mentioned the following sorts of knowledge which are needed to understand a mathematical theory:

(1) Knowledge of the items themselves: for each we know its statement, diagram, proof, construction or procedural formulation;

(2) Knowledge of the individual representation spaces, such as predecessor-successor relationships;

(3) Knowledge of inter-space relations, such as the dual idea.

In addition, there is also knowledge of how a particular theory ties in with other theories.

These are the key ingredients in building our representation for a mathematical theory. We have already discussed (2) and (3). We shall now summarize some of the information needed in the description of an individual item in order to understand it.

In deciding what knowledge about examples, results and concepts to represent, it is evident that all three types share a great many similarities. For instance, each has a declarative aspect: for a concept, its formal mathematical definition; for a result, its (if-then) statement; for an example, a caption, describing what it shows. Each has a procedural aspect: for a concept, its procedural formulation; for a result, its proof; for an example, its construction. Also, each has sets of "pointers" to other items through its intra-space (predecessor/sucessor) and inter-space (dual) relations. In addition, each item has a worth rating (from no to four *'s indicating its importance), an ID, a NAME, and its epistemological CLASSes. Thus, we represent all three by the same fundamental structure with a few particular fields modified to reflect special features depending on the type and epistemological class of the item. Figure 1 shows some of the fields in the representation for the Cantor set example (instead of pointers or ID's, we show the name or quote the statement of entries in the various pointer fields). (The complete item framework is described in [10].)

Once we have decided on this representation scheme, to organize our mathematical knowledge in a particular domain in terms of it, we must make several judgments. For instance, to build up a representation for the arithmetic examples of Section 2, we must first choose the representation space for an item (e.g., Q, the rational numbers, could alternatively be presented as a definition), and secondly, the item must be tied into its chosen space by determining its predecessors and successors (e.g., Q points back to Z, and ahead to R and Q_p). Thirdly, we must link an item to its dual items (e.g., Q can be linked to concepts of division, completeness, density, and cardinality, and to results on the irrationality of $2^{1/2}$, and the Archimedian properties of the real line). Fourthly, we can order the dual items. While the specific representation we build clearly reflects certain personal, pedagogical, historical or esthetic biases, the representation scheme we have presented is perfectly general.

While our representation in fact is embedded in a very large semantic network, it differs from other representations such as

Figure 1. Elements of the representation for the Cantor set.

ID *E333* CLASS *Reference, Counter-example*
RATING *** NAME *Cantor Set*

STMNT SETTING *R*
CAPTION *The Cantor set is an example of a perfect, nowhere dense set that has measure zero. It shows that uncountable sets can have measure 0.*

DEMON-STRA-TION AUTHOR *standard*
MAIN-IDEA *Delete "middle-thirds"*
CONSTRUCTION
0. Start with the unit interval [0, 1];
1. From [0, 1], *delete the middle third* (1/3, 2/3);
2. From the two remaining pieces, [0, 1/3] & [2/3, 1], *delete their middle thirds,* (1/9, 2/9) & (6/9, 7/9);
3. From the four remain pieces, delete the middle thirds;
N. At Nth step, delete from the 2^{N-1} *pieces the* 2^{N-1} *middle thirds.*

The sum of the lengths of the pieces removed is 1; what remains is called the Cantor set.

PICTURE

0_____________________1

0_______( )_______1

0__( )__( )__( )__1

Limiting set is Cantor set

REMARKS *Cantor set is good for making things happen almost everywhere or almost nowhere.*

LIFTINGS *Construction of general Cantor sets.*

IN-SPACE POINTERS:
BACK *Unit Interval*
FORWARD *Cantor function, General Cantor set's, 2-dimensional Cantor set*

DUAL-SPACE POINTERS:
CONCEPTS: *countable, measure zero, geometric series*
RESULTS: *"Perfect sets are uncountable", "Countable sets have measure 0"*

BIBLIOGRAPHIC REFERENCES:
See Gelbaum and Olmstead for general Cantor sets.
See Royden for Cantor functions.

PEDAGOGUES *(Rudin, 9), (Hoffman, 12)*

---

[9] in that it distinguishes several types of items -- results, examples and concepts -- and several types of relations -- the intra-space relations of the three representation spaces and the dual relation. In particular, we recognize and use the constructional relations between examples.

6. Applications

The ideas presented in this paper have been used to teach a freshman mathematics course at MIT. The purpose of this course was two-fold: (1) to teach and explore the important theory of eigenvalues, such as the perturbation theorems from Ortega [13]; and (2) to make young mathematicians aware of the ingredients and processes in understanding mathematics. The results of this experiment were extremely encouraging: the students became excellent question-askers and displayed a great deal of maturity and poise in their attempts to prove and understand new mathematics. Also, the epistemological ideas presented in this paper seemed quite natural for them. They used the epistemologyy and representation both to keep track of old knowledge and to help generate new knowledge [12].

The epistemology and representation summarized in this paper also serves as the basis for the design of two interactive systems, the *GROKKER SYSTEM* (*GS*) and the *GROKKER LEARNING ADVISOR* (*GLA*) [10]. *GS* is a proposed interactive system that allows its user to retrieve and manipulate information in a spontaneous stream-of-impulse way. The interaction takes place at a graphics CRT. *GS* has several modes governing the CRT display which are designed for different user tasks. For instance there are three *perusal modes* which allow the user to view a single item, an individual representation space, or all three spaces at once.

The main classes of functions in *GS* are: (1) *field functions* which allow the user to extract information from the fields of the data structure of individual items; (2) *epistemological functions* which allow the user to investigate epistemological relations between items; (3) *pedagogical functions* which allow the user to follow the exposition of a particular pedagogue; and (4) *perturbative functions* which allow the user to vary the statement of an item and have *GS* return items closely related to the perturbed item.

GS can be augmented by the advsior program *GLA*. The purpose of the combined *GS*/*GLA* system is to help neophytes understand mathematics and to learn *how* to understand. It

forms its advice from its epistemological knowledge, its own model of expert understanding, and its assessment of the user's current level of understanding.

Such a system could enter into partnership with theorem provers, or analogy- or concept-generating programs [2], [9], that need to use previously established mathematics. *GS* has many of the facilities desirable as support for such programs [1]. *GS* could make it easy for them retrieve and manipulate mathematical knowledge and could guide their search for relevant knowledge. *GS/GLA* would also invoke such programs to prove statements that arise through user queries and perturbations of the *GS/GLA* knowledge base.

For instance, in advising a non-resolution theorem prover (NRTP) in its efforts to prove a proposed theorem, *GS/GLA* could advise the NRTP to: (1) try out the proposed theorem in the special case of a reference or model example and use this instantiation as evidence -- for or against -- the theorem in much the same way as Gelernter's program [6] used a "diagram filter", i.e., a model example, in the domain of geometry; (2) custom tailor a model example to the specifics of the proposed theorem and examine how the theorem "works" in this case, and then bootstrap from this special case to the proposed theorem; (3) find examples for the proposed theorem and then consider other theorems that share these examples, and in particular, check if the proposed theorem can be proved by methods lifted from these other (example-dual) related theorems; (4) look for and invoke MP's and CP's that apply to the proposed theorem; (5) find two or more items in a predecessor/successor chain of examples or concepts and try to abstract the procedural information inherent in the links; (6) look for counter-examples in the collection of known reference and counter-examples.

7. Conclusions

In this paper we have presented a structure for representing our knowledge of mathematics and have singled out noteworthy classes of items in it. We have examined several types of relations between items in our mathematical knowledge. The analysis has provided a vocabulary and framework in which to talk about mathematics. This representation can be used to define a data base and functions which could be used by other programs that need to deal with mathematical knowledge and to support mechanization of certain mathematical tasks.

References

[1] Bledsoe, W. W., *Non-Resolution Theorem Proving.* The University of Texas at Austin, Mathematics Department Memo ATP-29, 1975.

[2] Bledsoe, W. W., and M. Tyson, *The UT Interactive Theorem Prover.* The University of Texas at Austin, Mathematics Department Memo ATP-17, 1975.

[3] Eilenberg, S., and N. Steenrod, *Foundations of Algebraic Topology.* Princeton University Press, New Jersey, 1952.

[4] Feigenbaum, E. A., and J. Feldman, *Compuuters and Thought*, Mc-Graw Hill, New York, 1963.

[5] Gelbaum, B. R., and J. Olmstead, *Counterexamples in Analysis.* Holden-Day, California, 1964.

[6] Gelernter, H. "Realization of a Geometry Proving Machine" in [4].

[7] Halmos, P. R., *Finite-Dimensional Vector Spaces.* D. Van Nostrand, New Jersey, 1942.

[8] Hoffman, K. M., *Analysis in Euclidean Space.* Prentice-Hall, New Jersey, 1975.

[9] Lenat, D. B., *Automated Theory Formation in Mathematics.* Proceedings Fifth IJCAI, 1977.

[10] Michener, E. R., *Epistemology, Representation, Understanding and Interactive Exploration of Mathematical Theories.* Doctoral disssertation, MIT Department of Mathematics, 1977.

[11] Michener, E. R., *The Structure of Mathematical Knowledge.* Technical Report, MIT Artificial Intelligence Laboratory, forthcoming.

[12] Michener, E. R., *Understanding Understanding Mathematics.* To appear in Proceedings of Amherst Conference, late 1978.

[13] Ortega, J. M., *Numerical Analysis: A Second Course.* Academic Press, New York, 1972.

[14] Polya, G., *How To Solve It.* Second Edition, Princeton University Press, New Jersey, 1973.

[15] Polya, G., *Induction and Analogy in Mathematics,* Volume 1 of *Mathematics and Plausible Reasoning.* Princeton University Press, New Jersey, 1973.

[16] Royden, H. L., *Real Analysis.* Second Edition, Macmillan, New York, 1963.

[17] Rudin, W., *Principles of Mathematical Analysis.* Second Edition, McGraw-Hill, New York, 1964.

[18] Spivak, M., *Differential Geometry.* Vol. 2, Publish or Perish Press, Boston, 1972.

[19] Strang, G., *Linear Algebra and its Applications.* Academic Press, New York, 1976.

[20] Thomas, G. B., *Calculus and Analytic Geometry.* Fourth Edition, Addison-Wesley, Massachusetts, 1972.

BACON.1: A general discovery system*

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

1. Introduction

In recent years researchers in artificial intelligence have produced a number of systems for carrying out scientific discovery. The programs include DENDRAL (Buchanan, Sutherland, and Feigenbaum, 1969; Feigenbaum and Lederberg, 1971), meta-DENDRAL (Buchanan, Feigenbaum, and Sridharan, 1972), MYCIN (Davis, Buchanan, and Shortliffe, 1972), and AM (Lenat, 1976). The list is impressive, and gives hope that AI is arriving at a true understanding of discovery processes.

In this paper I describe BACON.1, a general discovery system. The program uses a general representation and a small number of heuristics to discover an impressive range of empirical laws. Thus BACON.1 is general in the same sense that Newell, Shaw, and Simon's (1960) General Problem Solver was general. It also has much in common with the General Rule Inducer proposed by Simon and Lea (1974). An earlier, less general version of BACON.1 was reported in Langley (1978).

I begin by presenting a sample protocol of how one <u>might</u> go about discovering an empirical law, in this case Kepler's third law of planetary motion. I then consider BACON.1's representation of data, hypotheses, and heuristics. Next I outline the structure of the program and consider its heuristics in more detail. I follow with BACON.1's solutions of some familiar discovery tasks. Finally, I consider both the generality and the limitations of the current system.

2. A sample protocol

In 1619, Kepler announced his third law of planetary motion -- *the cube of a planet's distance from the sun was proportional to its period squared.* This law can be restated as $d^3/p^2 = c$, where d is the distance, p the period, and c a constant. How might one discover such a law? Below I give a sample protocol that draws upon some very simple heuristics:

1. look for constancies and generalize if you find one;
2. if the values of two attributes go up together, consider their ratio;
3. if one attribute's values go up as another's go down, consider their product.

*This paper was supported in part by Grant HCS75-22021 from the National Science Foundation, in part by ARPA Grant F44620-73-C-0074, and in part by a grant from the Alfred P. Sloan Foundation. I would like to thank H. A. Simon, Eric Johnson, Bob Neches, and Marshall Atlas for many of the ideas presented in this paper.

The value of these heuristics can best be seen in their operation. The three planets considered below, A, B, and C, obey a version of Kepler's law where the constant is 1.

```
PLANET A
DISTANCE ?
*1.0
PERIOD ?
*1.0
PLANET B
DISTANCE ?
*4.0
PERIOD ?
*8.0
PLANET C
DISTANCE ?
*9.0
PERIOD ?
*27.0
DISTANCE AND PERIOD SEEM TO GO UP
    TOGETHER
THE SLOPES AREN'T CONSTANT SO I'LL CONSIDER
    THE RATIO OF DISTANCE AND PERIOD
WHAT SHOULD I CALL IT?
*distance-over-period
```

Here the second heuristic has been applied. The distance and the period have been observed to increase together, and a new attribute defined as their ratio is considered. Next, its values are calculated.

```
THE VALUE OF THE DISTANCE-OVER-PERIOD IS
    0.33333333 WHEN THE PLANET IS C
THE VALUE OF THE DISTANCE-OVER-PERIOD IS
    0.5 WHEN THE PLANET IS B
THE VALUE OF THE DISTANCE-OVER-PERIOD IS
    1.0 WHEN THE PLANET IS A

DISTANCE SEEMS TO GO UP AS
    DISTANCE-OVER-PERIOD GOES DOWN
THE SLOPES AREN'T CONSTANT SO I'LL CONSIDER
    THE PRODUCT OF DISTANCE AND
    DISTANCE-OVER-PERIOD
WHAT SHOULD I CALL IT?
*distance-squared-over-period
THE VALUE OF THE
    DISTANCE-SQUARED-OVER-PERIOD IS 1.0
    WHEN THE PLANET IS A
THE VALUE OF THE
    DISTANCE-SQUARED-OVER-PERIOD IS 2.0
    WHEN THE PLANET IS B
THE VALUE OF THE
    DISTANCE-SQUARED-OVER-PERIOD IS 3.0
    WHEN THE PLANET IS C
```

DISTANCE-OVER-PERIOD SEEMS TO GO UP AS
DISTANCE-SQUARED-OVER-PERIOD GOES
DOWN

By this point the third heuristic has been applied twice. Two more concepts have been defined, d^2/p and d^3/p^2. The latter of these is the most recently formed, and it is time to examine its values.

THE VALUE OF THE
DISTANCE-CUBED-OVER-PERIOD-SQUARED IS
1.0 WHEN THE PLANET IS C
THE VALUE OF THE
DISTANCE-CUBED-OVER-PERIOD-SQUARED IS
1.0 WHEN THE PLANET IS B
THE DISTANCE-CUBED-OVER-PERIOD-SQUARED
MAY ALWAYS BE EQUAL TO 1.0
THE VALUE OF THE
DISTANCE-CUBED-OVER-PERIOD-SQUARED IS
1.0 WHEN THE PLANET IS A
THE HYPOTHESIS ALSO WORKS WHEN THE PLANET
IS A

Finally, the first heuristic has paid off, for the new attribute d^3/p^2 has a constant value for two of the planets. This led our discoverer to propose that the attribute has this value for all planets. Upon looking at the third planet, he finds this does seem to be the case. More remains to be done in testing the hypothesis, but the main work in discovering Kepler's third law has been completed.

3. BACON.1's representation

As the reader may have guessed, the above protocol was in fact generated by the BACON.1 program. Of course, BACON.1 was not designed to produce fluent English; I have given it the ability to generate simple protocols only to help demystify the path it travels towards discovery. Below I attempt to clarify the nature of the program still further by considering the representations it uses for its data, its hypotheses, and its heuristics.

3.1. The representation of data

BACON.1 represents its data in terms of data clusters. A data cluster is a set of attribute-value pairs linked to a common node; it represents a series of observations that have occurred together. The program knows about two types of attributes, independent and dependent. The system has control over independent attributes; it can vary their values and observe the results. The results consist of the values of the dependent attributes; these are what the system is trying to explain.

The program also knows that it can make generalizations about the values of dependent attributes, but that independent attribute-value pairs can be used only for conditions on those generalizations. Figure 1a shows some data clusters for a traditional concept attainment task. In this case, the concept is *red*. There are three independent attributes, *size*, *color*, and *shape*, and one dependent attribute, the *feedback*.

Much of BACON.1's power comes from its ability to define higher level attributes in terms of more primitive ones. For example, the program can create a new attribute which is the product, the ratio, or the linear combination of two existing attributes. It can also create an attribute whose value equals the value of another attribute modulo 2, or some other number.

The system asks the programmer for the names of these new attributes and treats them like any other; they can be used to define new attributes as well, so the process is recursive. The generation of higher level attributes allows a parsimonious representation of the data; it allows normally complex rules to be stated as simple constancies. I show in Figure 2a two data clusters which obey Kepler's third law. The three higher level attributes, d/p, d^2/p, and d^3/p^2, are represented in the same fashion as the attributes which define them.

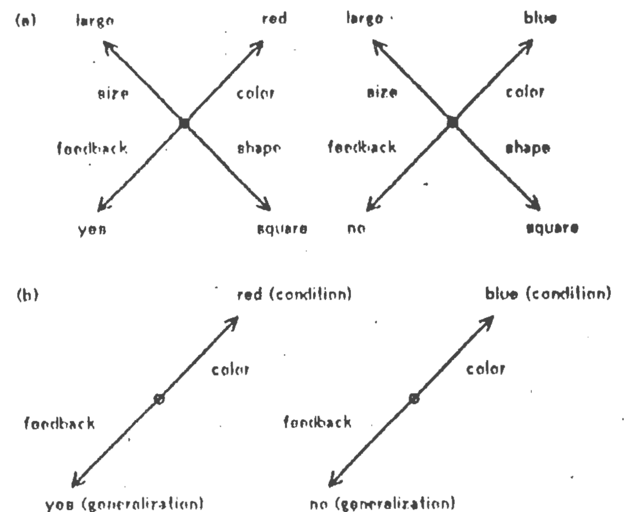


Figure 1. (a) Some data clusters for the concept red
(b) Hypothesis clusters for the concept red

3.2. The representation of hypotheses

BACON.1 represents hypotheses in much the same way as its data. An hypothesis cluster is also a set of linked attribute-value pairs. Some of these pairs are marked as generalizations, while others are marked as conditions on those generalizations. Thus the program specifies all rules it discovers as constancies, along with the conditions under which those constancies hold. The system makes no distinction between primitive and higher level attributes in its hypotheses.

In Figure 1b I give two hypothesis clusters describing the rule in the concept attainment task mentioned above. One cluster explains cases where the *feedback* is *yes*, while the other explains the *no* responses. Figure 2b shows a single hypothesis cluster for Kepler's third law. Only one cluster is needed since the constancy holds for all of the data being considered.

3.3. The representation of heuristics

BACON.1 is implemented in the production system language OPS2 (Forgy and McDermott, 1977). The heuristics of BACON.1, which I call regularity detectors, are implemented as OPS2 productions*. These consist of a set of conditions describing a general pattern that may be found in data, and

*These productions match against configurations of data and hypothesis clusters. Clusters are represented as value-attribute-node triples, each of which is an element in the OPS2 working memory.

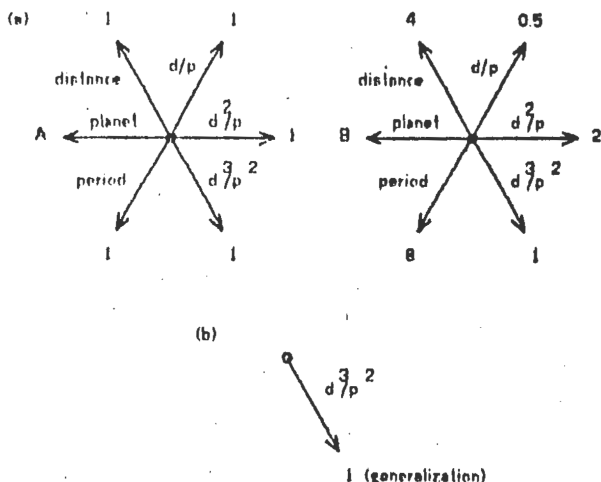


Figure 2 (a) Data clusters obeying Kepler's third law
(b) Hypothesis for Kepler's third law

an associated action; this action results in either the formulation or qualification of a generalization, or the definition of a higher level attribute. I discuss these rules more fully in the next section.

Production systems have been suggested as a general scheme for modeling complex thought processes by Newell and Simon (1972); they have listed a number of advantages, some of which are particularly relevant to discovery systems. First, production systems carry out a parallel search for the appropriate production to fire; this seems especially useful for searching a large set of data for constancies and trends. Second, production systems are data-driven, and any discovery system must clearly be responsive to the data it is trying to explain. Finally, production systems represent knowledge in relatively independent structures. This is an important advantage to a discovery system, for it means that knowledge (e.g., in terms of generalizations and higher level concepts) can be added incrementally and still interact in reasonable ways.

4. An overview of BACON.1

The BACON.1 program currently consists of 74 OPS2 productions. These can be divided into five major sets:

1. a set for setting up and running a factorial design to gather data;
2. a set to detect regularities in the data collected by the first and fourth sets;
3. a set that checks for repetitions in the higher level attributes suggested by the second set;
4. a set that calculates the values of the higher level attributes proposed by the second set;
5. a set to test generalizations set forth by the second set.

The first and fifth sets are standard mechanisms in any discovery system, and very similar components may be found in Simon and Lea's (1974) GRI program. The third set is simply a check for looping. The real innovations lie in the second and fourth sets, though the notion of higher level concepts is used extensively in Lenat's (1977) AM system.

I discuss all of the components in more detail below. In Figure 3 I present a top-level flow chart of the BACON.1 system. This departs somewhat from the standard formalism in order to better simulate the flow of control in the production system. The chart does not have an explicit stop, but the program halts when it has completed gathering its set of primitive data.

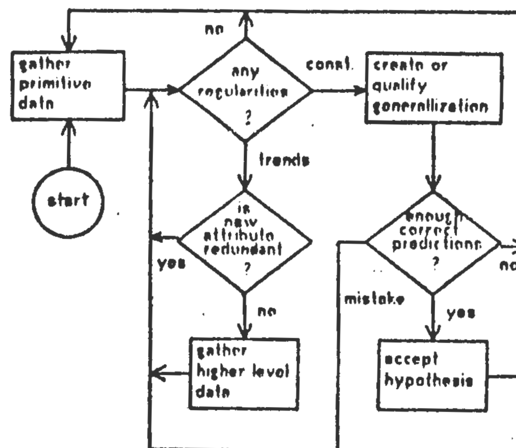


Figure 3 Top-level flow chart of BACON.1

4.1. Gathering data

This set of 20 productions asks the user a number of questions about the task the system is to solve. The system asks for the dependent and independent attributes involved. For independent attributes, it asks if they are interval or nominal scale. If interval, it asks for a suggested first value, a suggested increment, any limits on the possible values, and a suggested number of observations for this attribute. If nominal, it asks for a list of the attribute's possible values. No questions are asked about the dependent attributes' values.

As the productions get this information, they construct a set of more specific productions to carry out the experiment. When all the attributes have been considered and all the necessary productions added, the system begins its data collection. The data is collected through a traditional factorial design. Initially the first independent attribute is varied while the others are held constant. Then the value of the second attribute is changed, and the first is cycled through again in this new context. This continues until all the values of the second attribute have been cycled through. At this point the third attribute is modified, the above cycle is repeated, and so on.

For most of the tasks BACON.1 can handle, there is only a single independent attribute. In this special case, the values of the attribute are cycled through once and the system is finished collecting its data. Along with a given combination of independent attribute-value pairs, the system asks for the value of each dependent attribute.

The set of productions constructed for carrying out the data collection can be interrupted by the set of productions responsible for regularity detection. If both are true, the regularity detection productions win out over the data generation productions on the OPS2 conflict resolution principles. If a regularity is found, a generalization is made or a higher level attribute is considered. If a generalization is made, the testing productions compare its prediction to the existing data. If a higher level attribute is formed, it is tested for redundancy; if it is a new concept, its values are calculated and the regularity detectors examine these.

Eventually, the system's attention returns to its initial goal of collecting data, and it continues as if nothing has happened. The current version of BACON.1 does not use the

hypotheses and concepts it forms to help it search the data space intelligently. The one exception occurs when a constancy is found along an interval dimension; in this case, the system can jump ahead to look for a periodic relationship.

4.2. Discovering regularities

The second set of 20 productions are responsible for finding regularities in the data collected by the first set. The system's regularity detectors can be divided into a set of constancy detectors and a set of trend detectors. The first of these can deal with either nominal or numerical data, and leads to the postulation of generalizations and conditions on those generalizations. BACON.1's basic constancy detector can be paraphrased in English as

> *If an attribute has the same value across a number of data clusters,*
> *then hypothesize that the attribute always has that value.*

Note that this is simply a restatement of the traditional inductive inference rule; however, when combined with the ability to define higher level attributes, it gains in power considerably. The program's heuristic for finding conditions is nearly as simple; it may be restated as

> *If you are looking for a condition on a generalization,*
> *and the generalization is true in a number of data clusters,*
> *and attribute a has value v in those same data clusters,*
> *and the generalization is false in a number of other data clusters,*
> *and attribute a does not have value v in those data clusters,*
> *then propose the attribute-value pair a-v as a condition on the generalization.*

BACON.1's trend detectors operate only on numerical data. Some of these notice direct or inverse relations between attributes, such as the production

> *If the value of attribute a1 goes up as the value of attribute a2 goes up in a number of data clusters,*
> *then propose a direct relationship between a1 and a2,*
> *and calculate the slope of a1 with respect to a2.*

This production works in conjunction with related trend detectors that further analyze the data, such as

> *If there is a direct or inverse relationship between attributes a1 and a2,*
> *and the slope of a1 with respect to a2 is a constant m,*
> *then propose that a new attribute,* a1-m·a2, *be defined.*

and

> *If there is a direct relationship between attribute a1 and a2,*
> *and the slope of a1 with respect to a2 is not constant,*
> *and the values that led to the discovery of this trend were all positive,*
> *then propose that a new attribute,* a1/a2, *be defined.*

Similarly, an inverse relation will lead to the construction of an attribute for a1-m·a2 (which summarizes a linear relationship) or an attribute for a1·a2. Other combinations occur with different signs for a1 and a2.

The above productions lead to attributes which may be generalized over; however, higher level independent attributes may also be defined. One subset of the trend detectors looks for periodic relations in the data when the system needs a condition; these lead to the construction of modulus attributes. The major production may be stated as

> *If you are looking for a condition on a generalization,*
> *and the generalization is true in a number of data clusters,*
> *and there is an equal interval p between those data clusters along attribute a,*
> *then propose that a new attribute,* a modulo p, *be defined*

A similar rule can lead to the construction of linear combinations of independent attributes. Once such attributes are defined and their values examined, the condition detector can discover their relation to a generalization.

Once a dependent data triple is found to agree with an hypothesis, it is marked as explained. If more primitive triples were used in the calculation of this triple, they are similarly marked. The regularity detectors match only against unexplained data triples, so no confusion between different rules is produced.

Sometimes a qualified generalization is confirmed, but it fails to explain all instances of the generalization, since they do not satisfy all of the conditions. In this case, a new hypothesis is added which makes the same prediction, and a set of disjunctive conditions is searched for. Once this hypothesis is confirmed, if there remain other leftover triples, yet another hypothesis is added, and so on until all triples with this attribute-value pair have been marked as explained.

4.3. Checking dimensions

The third set of 12 productions check to see if a newly suggested product or ratio is identical with an already existing concept. For example, suppose the system has defined a new attribute, d/p, in terms of the primitive attributes d and p. If it finds p and d/p are inversely related, it will consider a new attribute, pd/p. Of course, this is equivalent to d, but BACON.1 doesn't know this since the meanings of its higher level attributes are opaque to it.

Accordingly, the identity-checking productions calculate the dimensionality of the new attribute. If this is the same as the dimensionality of an existing attribute, either primitive or higher level, the new concept is rejected. Also, a marker is added to memory telling the system to ignore this particular combination in the future.

If no identity is found, the system accepts the new concept and calculates its values. This check is performed only for products and ratios; it is inappropriate for linear combinations, since these are formed only when constant slopes are found. A comparable test is possible for modulus concepts, but is not implemented.

4.4. Calculating higher level values

The fourth set of 10 productions take the higher level concepts which pass the repetition check, and the values of

the attributes defining them, and use these to calculate the value of the new attribute. A separate production is used for each type of higher level attribute. Included in this set are productions which decide whether a new attribute is independent or dependent. If all the attributes making it up are independent, the new concept is marked independent as well; otherwise it is marked as dependent, and can be used in generalizations.

The productions for calculating values win out over those for data collection; as soon as the value of a higher-level attribute can be calculated, it will be. The only exception to this rule occurs if a regularity is detected among the new values. In this case, a generalization or still higher level attribute is formed and dealt with. Only after this new development is taken care of does the system return to its old computations.

4.5. Testing hypotheses

The final set of 12 productions is responsible for testing generalizations generated by the regularity detection productions. When a generalization is first made, a counter is created and set to zero. A set of preexisting test productions then compare the known data to the generalization.

For each agreement they find, they increment the counter by one and the data triple is marked as explained; if the counter reaches four, the hypothesis is accepted and the system moves on to other matters. However, if a data triple is found that disagrees with the generalization, the counter is reset to zero and a goal is set up to qualify the generalization. This gives control back to the regularity detectors which, hopefully, will discover a condition on the generalization.

When a condition has been found, one of the test productions adds a new set of test productions which incorporate knowledge of the new condition. These mask the old productions (through the OPS2 conflict resolution rules) with respect to the current hypothesis; the more general productions never have anything to say about this generalization again.

The new productions test the revised hypothesis in much the same way that the first ones did, except that they consider only those data clusters which satisfy the new condition. If the qualified hypothesis fits enough data, it is confirmed; if another counterexample is reached, a new condition is found, a new set of test productions is added which mask the last set, and the cycle begins again. This continues until the hypothesis is confirmed or until no useful conditions can be found to qualify a faulty generalization.

5. Some examples

Below I trace BACON.1's discovery path in two environments. The first contains data for the concept attainment task mentioned earlier, in which the *feedback* is *yes* if the *color* is *red*, and *no* when the *color* is *blue*. The second has data for a set of planets which obey an inverse square law, $ad^2 = 1$, where a is the acceleration and d is the distance from the sun.

The first task draws upon BACON.1's condition detectors; the second draws upon the trend detectors for direct and inverse relationships. Another task which I do not describe here, letter sequence extrapolation, uses both of these, as well as the trend detectors for periodic relationships. All of these tasks draw upon the core of BACON.1, the constancy detector.

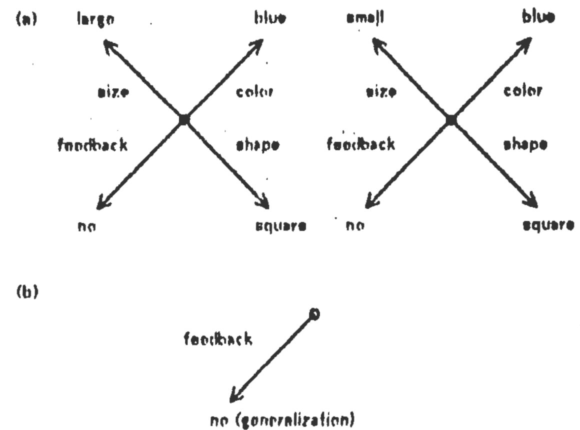


Figure 4. (a) Initial data clusters in the concept attainment run
(b) Initial hypothesis cluster in the concept attainment run

5.1. Concept attainment

In the concept attainment task, BACON.1 is told about three nominal independent attributes, *size*, *color*, and *shape*, and the values each can take. It is also informed of the single dependent attribute, the *feedback*. The system begins by systematically examining combinations of the three attributes. Since it was given *size* as the first attribute, it initially considers a *large blue square* and then a *small blue square*. The data clusters for these may be seen in Figure 4a. Since the *feedback* for both of these is *no*, the system generalizes by building the data cluster in Figure 4b.

BACON.1 next considers a *large red square* and a *small red square*; the data clusters for these are shown in Figure 5a. This time the *feedback* is *yes*. This causes the program to set up a goal to qualify its first hypothesis, and this goal is almost immediately satisfied. The condition-finding production proposes that the *feedback* is *no* if the *color* is *blue*. BACON.1 also sees that the *feedback* is sometimes *yes*, and finds that the *color* being *red* is a good condition for this generalization. Both hypothesis clusters are given in Figure 5b.

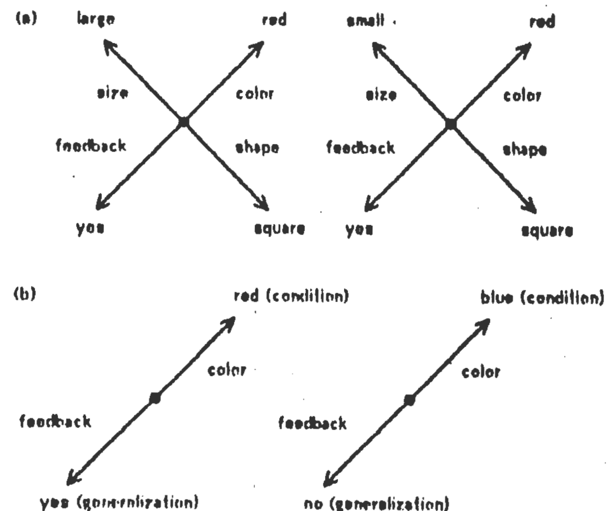


Figure 5. (a) More data clusters from the concept attainment run
(b) Final hypothesis clusters generated by BACON.1

So far all values of the *feedback* have been explained, but BACON.1 is not yet ready to accept its hypotheses. Two more data clusters must be found that agree with each rule before it will be confident. The combinations *large blue circle* and *small blue circle* do this for the first hypothesis, and the combinations *large red circle* and *small red circle* do it for the second. Both hypotheses are accepted, and productions are added to the system's permanent memory which will let it make predictions in the future. The program continues to gather data, making predictions and verifying them along the way, until its factorial design is completed.

5.2. The inverse square law

In the second task, BACON.1 is given one independent attribute, the *name* of a planet, and two dependent attributes, *a* for the acceleration, and *d* for the distance from the sun. The program examines the values of the latter attributes for three planets, and its trend detectors notice that the values of *a* increase as the values of *d* decrease. The slope of *a* with respect to *d* is calculated, but this is not constant. The initial data clusters can be seen in Figure 6.

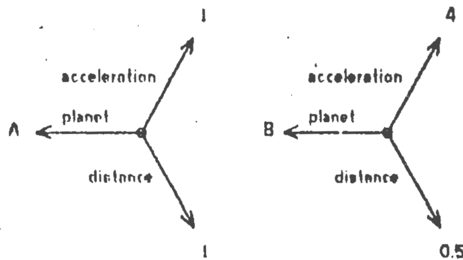


Figure 6 Data clusters satisfying an inverse square law

The relationship is inverse and the values positive, so a new attribute, *a·d*, is defined. The dimensionality of this attribute is unique, so its values of this attribute are calculated, and a new trend is detected. The attributes *a* and *ad* seem to be directly related; the slopes vary, and so a new concept, *ad/a*, is considered. However, after some calculation, BACON.1 realizes this attribute is equivalent to *d*, and rejects it.

Now another direct relationship is found, between *d* and *ad*; the attribute *a·ad*, or ad^2, is defined and its values calculated. The values of both *ad* and ad^2 can be seen in the extended data clusters given in Figure 7. The new attribute ad^2 is found to have a constant value of 1 for two of the data clusters. A generalization is made and tested. Two more data clusters are found that obey the law, and no exceptions. The rule is accepted and a production is built for making predictions. The system examines the remaining planets, and these fit as well, so it stops.

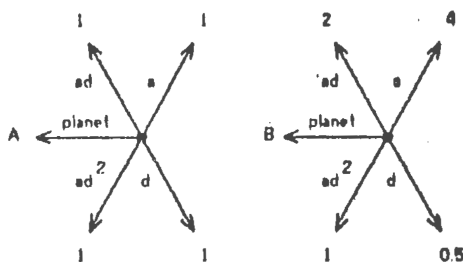


Figure 7. Higher level data for the inverse square law

6. Generality of the system

I have discussed three tasks BACON.1 can solve -- Kepler's third law, simple concept attainment, and the inverse square law. The program also succeeds on a number of tasks in which the laws are of a periodic nature.

6.1. Periodic discovery tasks

One of the periodic tasks BACON.1 can handle is an analogue of the letter sequence extrapolation tasks found on intelligence tests. These use sequences like *T E T F T G* --. BACON.1 must represent this sequence as *20 5 20 6 20 7* --; it must replace the letters with their position in the alphabet because it cannot represent the *next* relation between adjacent letters. A second is the class of sequences studied by Klahr and Wallace (1970), such as *blue square, red circle, red square, blue circle, red square, red circle, blue square*, --. In this example there are two dependent attributes, *color* and *shape*, whose values are nominal; also, the periodicities for each attribute are different.

BACON.1 can also handle sequences in which the periodicity is longer and some of the values are repeated within the period. An example is the sequence of pegs on which the smallest disk rests in the Tower of Hanoi puzzle. There are three *pegs*, the *initial peg*, the *goal peg*, and the *other peg*. The sequence for a five disk problem is *initial, goal, goal, other, other, initial, initial, goal, goal*, --. In each of these tasks, the program represents the *position* in the sequence as the single independent attribute.

Finally, Gardner (1969) has discussed a class of discovery tasks called Patterns. In these tasks there are two independent variables, the *row* and the *column*, each of which can vary from 1 to 6. The goal is to find some rule to predict the nominal *symbol* associated with each combination, and the rule is often periodic in nature. BACON.1 can solve a number of Patterns tasks.

6.2. Generality of the heuristics

Thus, BACON.1 can solve 6 classes of discovery tasks, if one groups the two power laws together. Moreover, nearly the same productions are used in the solution of every type of task. Of course, there are some exceptions to this.

In its discovery of power laws such as $d^3/p^2 = c$, there is no use of the heuristics for discovering periodicity or finding conditions. Similarly, in most of the sequence extrapolation tasks no use is made of the inverse and direct relation detectors. And 3 productions were added to enable the system to solve some of the Patterns tasks.

The general rule, however, is that most of the heuristics are used most of the time. And some, like the constancy detector and the productions for testing hypotheses, are used in all cases. Along with their simplicity, this suggests that they are truly general rules that any discovery system might use profitably. It also suggests that BACON.1 is a general discovery system in the same sense that GPS was a general problem solver.

7. Limitations of the system

We have seen that the BACON.1 program exhibits considerable generality. Moreover, it achieves this

generality through the use of a small number of simple heuristics. However, the systems does have serious limitations, and below I consider some of its drawbacks.

7.1. Restricted representation

What are the implications of BACON.1's data and hypothesis cluster representations? Restricting the data to attribute-value pairs means that relations between structures cannot be represented. For example, one cannot express an attribute for the distance between two objects. Neither can one deal with more complex concepts occurring in natural language which take a number of arguments, such as give or hit. BACON.1's representation can be extended easily enough, but its heuristics are designed to deal exclusively with data and hypothesis clusters, and are not so easily modified.

A related shortcoming lies in the nature of BACON.1's higher level concepts. So far, all of these have been numerical in nature. Even with an attribute-value representation, it is possible to define higher level nominal attributes, in terms of conjuncts and disjuncts of other attribute-value pairs. In fact, such as ability is necessary if the system is ever to carry out scientific classification. An extension along these lines is possible while retaining the program's main heuristics, and is a priority for future research.

7.2. Noise

The current BACON.1 operates on perfect data. Since real-world data is noisy, it would be nice if the program could handle it as well. Modifying the system to deal with exceptions would be fairly straightforward, since only the testing procedures need be changed. The new test productions would accept an hypothesis if the ratio of confirming to disconfirming data was above a certain value, and if a minimum number of data clusters had been examined.

Altering BACON.1 to handle random noise in its numerical data would be more complicated. In this case, the regularity detectors themselves would have to be changed. A partial solution might lie in re-representing the values of attributes as intervals instead of points. A constancy detector using this representation might be

> If the intervals for an attribute overlap across a number of data clusters,
> then hypothesize that the value of the attribute always falls within the the average of those intervals.

This is a promising path to explore, but the strategies for determining the size of the interval remain to be specified.

Once an hypothesis has been formed, the system might calculate the values of a new attribute defined as the difference between the attribute's values and the hypothesized value. This solution is especially attractive, since it takes advantage of BACON.1's strong point -- its ability to construct higher level attributes.

7.3. Searching the data space

As described earlier, BACON.1 collects its primitive data through a straightforward factorial design strategy. But it might use the discoveries it has made up to a certain point to let it gather data more effectively. For example, if the system were testing an hypothesis with conditions on it, then it might examine only those data clusters satisfying the conditions. Also, if higher level independent attributes have been defined, the program might consider varying their values in place of the original attributes' values.

8. Conclusions

In this paper I outlined BACON.1, a production system that discovers empirical laws. I described the system's representation of data and hypotheses, and I explained the heuristics used in its discovery process. I showed that BACON.1's behavior consisted of 5 major components -- gathering primitive data, discovering regularities in this data, checking for loops in higher level concepts, calculating the values of higher level attributes, and testing hypotheses.

I sought to describe the system's strategies still further by covering in some detail its solutions to two tasks. I also outlined the remaining tasks BACON.1 can solve, and presented the evidence for its generality. Finally I pointed out the program's limitations in representing concepts and relations, in dealing with noise, and in searching the data space. At the same time, I made some suggestions for future research.

In conclusion, the BACON.1 program has shown both generality and simplicity. By implication, the representation of data and hypotheses in terms of higher level concepts, and the representation of heuristics in terms of production rules are fruitful lines to pursue. Many challenges lie ahead, but the success to date suggests that the path is a promising one.

9. References

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

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.

Davis, R., Buchanan, B. G., and Shortliffe, E. Production rules as a representation for a knowledge-based consultation program. Stanford AI Laboratory Memo AIM-266, 1975.

Feigenbaum, E. B. and Lederberg, J. On generality and problem solving: A case study using the DENDRAL program. In B. Meltzer and D. Michie (ed.), *Machine Intelligence 6*. New York: American Elsevier Publishing Co., 1971, pp. 165-190.

Forgy, C. and McDermott, J. OPS2 Manual. Pittsburgh, Pa.: Carnegie-Mellon University, Department of Computer Science, 1977.

Gardner, M. Mathematical games. *Scientific American*, 1969, 221, 140-146.

Klahr, D. and Wallace, J. G. The development of serial completion strategies: An information processing approach. *British Journal of Psychology*, 1970, 61, 243-257.

Langley, P. BACON: A production system that discovers empirical laws. CIP Working Paper No. 360, Carnegie-Mellon University, 1978.

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

Newell, A., Shaw, J. C., and Simon, H. A. Report on a general problem solving program for a computer. *Information Processing: Proceedings of the International Conference on Information Processing*. Paris: UNESCO, 1960, pp. 256-264.

Newell, A. and Simon, H. A. *Human problem solving.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1972.

Simon, H. A. and Lea, G. Problem solving and rule induction: A unified view. In L. Gregg (ed.), *Knowledge and cognition.*) Potomac, Maryland: Lawrence Erlbaum Associates, 1974.

LEARNING STRATEGIES BY COMPUTER[1]

Yuichiro Anzai

Department of Psychology
Carnegie-Mellon University

Introduction

A cognitive system needs a *strategy*, i.e., a well-organized set of procedural knowledge, for efficiently solving a complex problem solving task. How such a system acquires strategies through experience is a fundamental question for artificial intelligence, because efficient procedures for artificial intelligence systems reduce practically limited execution time to a considerable extent, but are not always available before encountering the task. Work on acqusition of procedural knowledge, mostly done as piecewise rule learning (e.g., Buchanan & Mitchell, 1977), is quite important in this sense. Strategy learning, however, has a characteristic that rules generated should be organized as a strategy through a problem solving task. Little work has been done for computerized strategy learning, and we still need an intensive study on how strategies are represented in, and learned by, humans and computers.

Along this direction, the author has developed a theoretical exploration (Anzai, 1978), an experimental analysis (Anzai & Simon, 1977), and construction of computer programs for modelling processes and representation of strategy learning. The well-known Tower of Hanoi puzzle was used for the experiment, because it can be solved by various structurally different strategies (Simon, 1975).

Following the theoretical and experimental studies, this paper presents the third and final part of the work, a detailed report on strategy learning behavior of a computer program written in an adaptive production system language called HAPS. The program was written for discovering strategies in the five-disk Tower of Hanoi problem, and the current version has succeeded in learning three very different kinds of strategies successively while solving the problem six times from the initial situation. Structure of the program is general, and it is now being extended to learning strategies in another problem domain. Many ideas are embedded in it, but the basic hypothesis is simple: strategies are learned through repeatedly solving the same problem, and productions newly generated will be used extensively for learning new strategies.

The paper first summarizes briefly results from the theoretical and experimental analyses. Then, as a main part of the paper, it describes computational results of strategy learning in the Tower of Hanoi problem. Last, it mentions implication of the work and its relation to other works.

Review of Theoretical and Experimental Analyses

The theory proposed (Anzai, 1978) asserts that strategies can be learned through repeated solution of the same problem. Initially a system does not always have information about what state is "good", but usually has general heuristics for detecting a "bad" situation; e.g., returning to a previously visited state is bad. So generally learning begins with collecting "bad" instances by heuristic search, and proceeds to a *move-pattern strategy* by creating rules for avoiding generation of the bad instances. Then, after heuristic search is made more efficient by the newly generated rules, learning advances toward a *means-ends strategy* by collecting "good" instances and creating rules for generating subgoals. Finally the system learns a *working-forward strategy* which generates merely a sequence of a small number of operators that can be applied successively to the initial knowledge state. The sequence may be obtained by discovering some pattern in sequential structure of previously used operators.

Also a representational scheme is presented in the theoretical exploration (Anzai, 1978) as a hierarchical adaptive production system (HAPS) that incorporates subroutine and recursive calls of productions, and an ability to create new productions. Formally, HAPS is a LISP-based interpreter for hierarchical adaptive production systems with a single working memory (WM). A HAPS program consists of WM, which is an ordered set of lists, and a hierarchically and recursively structured set of productions. With its hierarchical structure, a HAPS program for strategy learning reflects generally a strategy learning process that consists of various subprocesses such as heuristic search, rule induction, rule generation, state transfer in a problem space, and so on. Each subprocess may be represented by a subroutine in the program.

[1] This research was supported by Research Grant MH-07722 from the National Institute of Mental Health. The author appreciates Pat Langley, Bob Neches, Dave Neves and Herbert Simon for their helpful comments on this work. The author's present address is: Department of Administration Engineering, Keio University, 3-14-1, Hiyoshi, Yokohama, Japan.

The experimental results (Anzai & Simon, 1977) for strategy learning by a human in the five-disk Tower of Hanoi problem clearly reflect the above theoretical framework. In the experiment, a human subject who had never solved the problem tried to find good solution procedures by solving the puzzle four times repeatedly. Starting with heuristic search and planning future moves, the subject finally succeeded in learning two strategies like the *goal recursion* and *inner-directed goal recursion strategies* referred to by Simon (1975). In the experiment, learning each strategy was guided by a mixture of previously learned strategies: one of those strategies was retrieved and used whenever needed.

Computational Results of Strategy Learning

The current version of the program includes 164 productions before learning. Mechanisms for learning strategies, detailed in Anzai (1978), are embedded in the program. Computation proceeds just as indicated by the mechanisms. The program made six runs successively from the initial disk configuration. It generated 13 new productions, and learned three different strategies: a move-pattern strategy, a means-ends strategy, and a working-forward strategy. Among the thirteen generated productions, seven were used for the first strategy, four were for the second, and the remaining two constitute a part of the third strategy.

<*Run 1*>

The program tried to collect bad moves using heuristic search, induced conditions for avoiding those moves, and generated productions for avoiding them. The size of trees generated during the heuristic search process became smaller and smaller, which implies that the search was made more and more efficient during Run 1 by generation of new productions. Fig. 1 illustrates dynamic behavior of the tree search. In the figure, (X Y Z) denotes a disk configuration: disks in X are on Peg A, disks in Y are on Peg B and disks in Z are on Peg C. For instance, (12345 NIL NIL) means that the configuration is such that Disks 1, . . . , 5 are on Peg A, and no disks are on Pegs B and C. We assume that Peg A is the initial peg and Peg C is the goal peg. Each tree in Fig. 1 grows from a root node, which is an actual configuration. Configurations in trees are *imaginary* ones. At any moment, the system retains information about past, current and imaginary problem states.

The system includes productions for finding a legal move, and an evaluation heuristic that taking disks off the initial peg is good. Using them, in the first tree search with the root node of (12345 NIL NIL), two good successive moves were generated: (MOVE 1 PEGA PEGB) and (MOVE 2 PEGA PEGC). (MOVE X P Q) denotes moving Disk X from Peg P to Peg Q. Applying these two move operators generated the disk configuration (345 1 2)[2]. Fig. 1 shows that this configuration, as a root node, generated a large search tree. In this part of the search process, three new productions for avoiding poor moves were created. As an example, let us describe how the first new production was acquired.

At the configuration (PEGS (3 4 5) (1) (2)), a subroutine for finding a legal move generates an *imaginary* move, (IMOVE 1 PEGB PEGA). It is then applied to generate a new imagined disk configuration, (IMAGE (1 3 4 5) NIL (2)). The system does not decide if it was a good move or not, and the search is made further. The same subroutine then happens to generate (IMOVE 1 PEGA PEGB). This provides another imagined configuration, (IMAGE (3 4 5) (1) (2)). Now since the new imaginary configuration is the same as the current real one, (PEGS (3 4 5) (1) (2)), the system detects return to a previously visited state by a pattern detecting production:

(PEGS $A $B $C) (IMAGE $A $B $C) --> (DEP (BAD))[3].

An atom headed by $ denotes a variable. Then, triggered by (BAD) which was DEPosited in WM, a subroutine[4] for collecting information relevant to storing (BAD) works to generate a knowledge element, (GOT (2 PEGA PEGC) (1 PEGB PEGA) (1 PEGA PEGB)). This implies that the two imaginary moves (IMOVE 1 PEGB PEGA) and (IMOVE 1 PEGA PEGB), and the past move (PASTMOVE 2 PEGA PEGC) might be responsible for causing the return to a previous state.

After the GOT information is obtained, the system continues tree search. The depth of search is limited to 2. Thus, here control is given back to the node (1345 NIL 2)[5] shown in Fig. 1. At this node, the subroutine for finding a legal move generates (CAN 1 PEGA PEGC) as a legal move not yet tried. But actually, before generating this move, based on (IMAGE (1 3 4 5) NIL (2)) and (GOT . . .), the system generates:

(COND (PEGS (1 3 4 5) NIL (2))), (COND (PASTPASTMOVE 2 PEGA PEGC)), (COND (PASTMOVE 1 PEGB PEGA)), (COND (ABSENT (TRIED 1 PEGA PEGB (1 3 4 5) NIL (2)))) and (ACTION (DEP (TRIED 1 PEGA PEGB (1 3 4 5) NIL (2)))).

The first COND is derived from the current disk configuration. The second and third CONDs, and the ACTION are from the argument of GOT. The fourth COND is a modified copy of the ACTION.

At this point, specific values in the CONDs and ACTION are substituted by variables, and then a new production is generated by juxtaposing the arguments of CONDs in the condition side and putting the argument of the ACTION in the action side. The production is named <N1>, and shown in Table 1. Semantic definitions of production elements appearing in this paper are given in Table 2.

After generating <N1>, the system continues tree search, first by applying a legal and applicable move, (CAN 1 PEGA PEGC).

[2] In the program, this configuration is represented as (PEGS (3 4 5) (1) (2)).

[3] Actually this is a simplified version of the production in the program.

[4] Note that the pattern detecting production involves no information about move operators. The subroutine is a knowledge-based mechanism for causal inference: it picks up recently applied operators, based on state information in the production.

[5] Corresponds to (IMAGE (1 3 4 5) NIL (2)) in the program.

Fig. 1 Heuristic search process in Run 1

("==" denotes external process.
"--" denotes internal process.)

```
(12345 NIL NIL) ==> (2345 1 NIL) ==> (345 1 2) ==> (345 NIL 12) ==>
       |                                /       \
 (2345 1 NIL)                   (1345 NIL 2)    (345 NIL 12)
       |                        /     |     \          \
  (345 1 2)          (345 1 2) (345 NIL 12) (1345 2 NIL)  (45 3 12)
  very good             bad          |         good       very good
                                 (345 1 2)
                                    bad

==> (45 3 12) ==> (145 3 2) ==> (145 23 NIL) ==> (45 123 NIL) ==>
      /              \                              \
  (145 3 2)            (45 13 2)                    (45 123 NIL)
      |             /        \         \                  \
(145 23 NIL) (145 3 2) (45 3 12) (245 13 NIL)         (5 123 4)
    good         |        bad        good             very good
             (45 3 12)
                bad

==> (5 123 4) ==> (5 23 14) ==> (25 3 14) ==> (125 3 4) ==>
      /      \                     /     \
(5 23 14)  (45 123 NIL)      (125 3 4)   cannot
    |           |                |
(25 3 14)   (45 23 1)      (125 NIL 34)
  good          |               good
           (145 23 NIL)
               bad

==> (125 NIL 34) ==> (25 1 34) ==> (5 1 234)
          |                            |
      (25 1 34)                  (5 NIL 1234)
          |                            |
      (5 1 234)                  (NIL 5 1234)
      very good                  Run termination
```

Table 1 Productions learned by program

I. Productions learned in Run 1

<N1> (PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $R $P) (ABSENT (TRIED $Y $P $R $A $B $C)) --> (DEP (TRIED $Y $P $R $A $B $C))

<N2> (PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $R $P) (ABSENT (TRIED $Y $P $Q $A $B $C)) --> (DEP (TRIED $Y $P $Q $A $B $C))

<N3> (PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $P $R) (ABSENT (TRIED $X $Q $P $A $B $C)) --> (DEP (TRIED $X $Q $P $A $B $C))

<N4> (PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $R $Q) (ABSENT (TRIED $Y $Q $P $A $B $C)) --> (DEP (TRIED $Y $Q $P $A $B $C))

<N5> (PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $R $Q) (ABSENT (TRIED $Y $Q $R $A $B $C)) --> (DEP (TRIED $Y $Q $R $A $B $C))

<N6> (PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $R $P) (ABSENT (TRIED $X $Q $P $A $B $C)) --> (DEP (TRIED $X $Q $P $A $B $C))

<N7> (PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $P $R) (ABSENT (TRIED $Y $R $P $A $B $C)) --> (DEP (TRIED $Y $R $P $A $B $C))

II. Productions learned in Run 3

```
<N8>  (PEGS $A $B $C) (GOAL $X $P $R) (*EQUAL $X (*TOP
      $P $A $B $C)) (*LESS (*TOP $R $A $B $C) $X)
      (ABSENT (GOAL (*NEXTSMALLER $X $R $A $B $C) $R
      (*OTHER $P $R : PEGA PEGB PEGC))) --> (DEP (GOAL
      (*NEXTSMALLER $X $R $A $B $C) $R (*OTHER $P $R :
      PEGA PEGB PEGC)))

<N9>  (PEGS $A $B $C) (GOAL $X $P $Q) (*EQUAL $X (*TOP
      $P $A $B $C)) (*LESS (*TOP $Q $A $B $C) $X)
      (ABSENT (GOAL (*NEXTSMALLER $X $Q $A $B $C) $Q
      (*OTHER $P $Q : PEGA PEGB PEGC))) --> (DEP
      (GOAL (*NEXTSMALLER $X $Q $A $B $C) $Q (*OTHER
      $P $Q : PEGA PEGB PEGC)))

<N10> (PEGS $A $B $C) (GOAL $X $P $Q) (*LESS $X (*TOP
      $Q $A $B $C)) (*LESS (*TOP $P $A $B $C) $X)
      (ABSENT (GOAL (*NEXTSMALLER $X $P $A $B $C) $P
      (*OTHER $P $Q : PEGA PEGB PEGC))) --> (DEP
      (GOAL (*NEXTSMALLER $X $P $A $B $C) $P (*OTHER
      $P $Q : PEGA PEGB PEGC)))

<N11> (PEGS $A $B $C) (ABSENT GOAL) (PASTMOVE $X $P
      $Q) --> (DEP (GOAL (*NEXTSMALLER $X (*OTHER $P
      $Q : PEGA PEGB PEGC) $A $B $C) (*OTHER $P $Q :
      PEGA PEGB PEGC) $Q))
```

III. Productions learned in Run 5

```
<N12> (PEGS $A $B $C) (ABSENT (GOAL (*NEXTLESS 2)
      (*OTHER (*XTR3 (PGOAL 2 $Q $R)) (*XTR4 (PGOAL 2
      $Q $R)) : PEGA PEGB PEGC) $R)) (ABSENT (GOAL 2
      $Q $R)) (ABSENT (GOAL (*NEXTSMALLER 2 $Q $A $B
      $C) $Q (*OTHER (*XTR3 (PGOAL 2 $Q $R)) (*XTR4
      (PGOAL 2 $Q $R)) : PEGA PEGB PEGC))) (PGOAL 2 $Q
      $R) (*P-ON 2 $Q $A $B $C) --> (DEP (GOAL
      (*NEXTLESS 2) (*OTHER (*XTR3 (PGOAL 2 $Q $R))
      (*XTR4 (PGOAL 2 $Q $R)) : PEGA PEGB PEGC) $R))
      (DEP (GOAL 2 $Q $R)) (DEP (GOAL (*NEXTSMALLER 2
      $Q $A $B $C) $Q (*OTHER (*XTR3 (PGOAL 2 $Q $R))
      (*XTR4 (PGOAL 2 $Q $R)) : PEGA PEGB PEGC))) (REM
      (PGOAL 2 $Q $R))

<N13> (PEGS $A $B $C) (ABSENT (PGOAL (*NEXTLESS $X3)
      (*OTHER (*XTR3 (PGOAL $X3 $R $Q)) (*XTR4 (PGOAL
      $X3 $R $Q)) : PEGA PEGB PEGC) $Q)) (ABSENT
      (GOAL $X3 $R $Q)) (ABSENT (PGOAL
      (*NEXTSMALLER $X3 $R $A $B $C) $R (*OTHER
      (*XTR3 (PGOAL $X3 $R $Q)) (*XTR4 (PGOAL $X3 $R
      $Q)) : PEGA PEGB PEGC))) (PGOAL $X3 $R $Q) (*P-
      ON $X3 $R $A $B $C) (*LESS 2 $X3) --> (DEP
      (PGOAL (*NEXTLESS $X3) (*OTHER (*XTR3 (PGOAL
      $X3 $R $Q)) (*XTR4 (PGOAL $X3 $R $Q)) : PEGA
      PEGB PEGC) $Q)) (DEP (GOAL $X3 $R $Q)) (DEP
      (PGOAL (*NEXTSMALLER $X3 $R $A $B $C) $R
      (*OTHER (*XTR3 (PGOAL $X3 $R $Q)) (*XTR4 (PGOAL
      $X3 $R $Q)) : PEGA PEGB PEGC))) (REM (PGOAL $X3
      $R $Q))
```

Table 2 Semantic definitions of production elements

Only definitions necessary for understanding Table 1 and other notations appearing in this paper are listed. The original HAPS incorporates some more predicates, functions and actions.

A. Predicates

(*EQUAL X Y) is true iff X and Y are equal expressions.

(*LESS X Y) is true iff X and Y are numbers and X is less than Y, or X is a number and Y is NIL.

(*P-ON X P A B C) is true iff X > 1, and all 1, . . . , X are included in A if P = PEGA, in B if P = PEGB, or in C if P = PEGC.

(*PSHAPE P A B C) is true iff A is PSHAPEd if P = PEGA, B is PSHAPEd if P = PEGB, or C is PSHAPEd if P = PEGC. X is PSHAPEd iff X = (1 . . . k) for some integer k > 1.

B. Functions

(*NEXTLESS X) returns the integer one less than X. X must be an integer larger than 1.

(*NEXTSMALLER X P A B C) returns the number nextsmaller than X in the list A if P = PEGA, B if P = PEGB, or C if P = PEGC. X need not be included in the corresponding list A, B, or C. If no such number exists, NIL is returned.

(*OTHER I J K . . . : P Q R . . .) returns an element among I J K . . . other than P Q R . . . . The number of I, J, K, . . . must be exactly one less than the number of P, Q, R, . . .

(*TOP P A B C) returns the leftmost element in the list A if P = PEGA, in B if P = PEGB, or in C if P = PEGC. If the list (A, B or C) is NIL, NIL is returned.

(*XTRi X) returns X's i-th element counting from left: i = 1, . . . , 6.

C. Actions

(DEP X A B C . . .) deposits into WM a list X after A, B, C, . . . , are executed. A, B, C, . . . are called substitutions. A substitution has the form: (Y I J K . . . : P Q R . . .). Its execution means that any of P Q R . . . other than I J K . . . is substituted into any occurence of Y in X.

(REM X A B C . . .) removes from WM any occurence of X in WM after substitutions, A, B, C, . . . , are executed.

D. Negative conditions

(ABSENT <list>) is true iff <list> matches no element in WM.

(ABSENT <atom>) is true iff no list headed by <atom> exists in WM.

The system continued heuristic search in this way, but gave up solving when the disk configuration (IMAGE NIL (5) (1 2 3 4)) was imagined. It was done by a heuristic introduced just for making the system quit a tedious trial-and-error search on the half way.

After productions fired 914 times, this transfer of the biggest disk to Peg B was detected and Run 1 was terminated. During that time, seven productions were created. They are listed in Table 1. The table shows that structure of the new productions involves some commonalities: <N1> is similar to <N5> and <N7>, <N2> similar to <N4>, and <N3> to <N6>. It is because they were generated by the same mechanism. Productions in the first class are for avoiding immediate loop moves. Productions in the second category are used for avoiding two-step moves of the same disk. Productions in the third are for avoiding return to the peg on which the disk was put in the next-to-most-recent past. It is easy to see that some conditions in those productions are redundant: we can define a smaller number of productions functionally equivalent to the seven productions.

<Run 2>

Run 2 was used for examining whether the system had learned a well-formed strategy in Run 1. At the same time, in Run 2, the system transformed weak subgoals to stronger ones[6].

The seven productions, <N1> - <N7>, created in Run 1 were retained at the top of one of subroutines. They were extensively used in Run 2 for discarding legal but bad moves. For example, at the configuration (PEGS (3 4 5) (2) (1)), <N7> fired for discarding the legal move (MOVE 2 PEGB PEGA), and <N3> fired for eliminating (MOVE 1 PEGC PEGA). After those two productions fired, only one legal move remained: (MOVE 1 PEGC PEGB).

Run 2 ended successfully when the final goal (PEGS NIL NIL (1 2 3 4 5)) was attained. Productions fired 656 times in Run 2. The fact that no error move was made in Run 2 indicates that <N1> - <N7>, with the learned declarative knowledge (INITIALLY TRIED 1 PEGA PEGB), meaning that, at the initial configuration, moving Disk 1 from Peg A to Peg B should be avoided, were sufficient for determining a unique right move in each disk configuration encountered. Thus, those productions, with (INITIALLY . . .), and productions for generating legal moves and updating problem states, constitute a well-defined strategy. We call it the ***negative move-pattern strategy***.

It should be noted that this strategy discovered by the program was not discussed in Simon's work on comparison of various strategies in the Tower of Hanoi puzzle (Simon, 1975). Different from his move-pattern strategy (Simon, 1975), which selects a correct move ***positively*** based on a move-generating pattern, our strategy chooses moves only ***negatively*** by discarding poor moves.

[6] A subgoal is ***weak*** if it only partially defines a subgoal operator, e.g., "putting a disk on some peg." A subgoal is ***strong*** if it provides a well-defined operator, e.g., "transferring a disk from some peg to some other peg."

Adequate transformation of subgoal information is important for learning subgoal-type strategies, and also suggested by the experimental data. The program transforms initial weak subgoals to stronger ones through heuristics related to the problem solving process: subgoals are transformed through the problem solving process itself.

First such transformation was done in Run 2. For example, a weak subgoal, (GOAL-OFF 3 PEGA), asserting that a subgoal is to take Disk 3 off Peg A, was transformed to (GOAL 3 PEGA PEGC) when 3 was taken off Peg A for the first time and transferred to Peg C. The resulting stronger subgoals were used in Run 3 for generating a means-ends strategy.

<Run 3>

Initially the system has the following (stronger) subgoals in WM:

> (GOAL 1 PEGA PEGC), (GOAL 2 PEGA PEGB), (GOAL 3 PEGA PEGC), (GOAL 4 PEGA PEGB), (GOAL 5 PEGA PEGC), (GOALSTACK 2 PEGB PEGC), (GOALSTACK 4 PEGB PEGC).

As seen above, the system generated two kinds of subgoals in Run 2; GOALs and GOALSTACKs. GOALs were transformed from "taking disks off the initial peg," and GOALSTACKs were derived from "putting disks on the goal peg." If taking-off and putting-on subgoals were attained simultaneously, only a GOAL is created. The system assumes priority ordering between them: GOALSTACK-type subgoals are considered only when no GOAL-type subgoal exists in WM.

Now since (PEGS (1 2 3 4 5) NIL NIL) and (GOAL 1 PEGA PEGC) are initially in WM, Disk 1 is at the top of Peg A, and no disk in on Peg C, the production:

> (PEGS $A $B $C) (GOAL $X $P $Q) (*EQUAL $X (*TOP $P $A $B $C)) (*LESS $X (*TOP $Q $A $B $C)) --> (REM (GOAL $X $P $Q)) (DEP (MOVE $X $P $Q))

fires, REMoves (GOAL 1 PEGA PEGC) from WM, and DEPosits (MOVE 1 PEGA PEGC) into WM.

By assumption from the theory, the above production for finding a legal move using a subgoal dominates the production for detecting a legal move by combinatorial search.

After (GOAL 1 PEGA PEGC) and (GOAL 2 PEGA PEGB) are transformed to (MOVE 1 PEGA PEGC) and (MOVE 2 PEGA PEGB) by the above production, the system reaches (PEGS (3 4 5) (2) (1)). Now, no GOAL is immediately applicable, and the system cannot help using the previously learned negative move-pattern strategy.

However, at the same time, the program notices that (GOAL 3 PEGA PEGC) is in WM, Disk 3 is currently at the top of Peg A, but 3 is larger than the top disk on Peg C. This implies that, in terms of a means-ends analysis, the system is already ***near*** to a state, S, where 3 can be moved from Peg A to C, and the ***difference*** between S and the current state is that Peg C's top disk is smaller than Disk 3; in terms of HAPS notation, (*LESS (*TOP PEGC (3 4 5) (2) (1)) 3) is true. The system then tries to find a more immediately

applicable subgoal using these kinds of information. It is done in the following manner.

At (PEGS (3 4 5) (2) (1)), the system first remembers the current situation by depositing into WM (COND (PEGS (3 4 5) (2) (1))), (COND (GOAL 3 PEGA PEGC)), (COND (*EQUAL 3 (*TOP PEGA (3 4 5) (2) (1)))) and (COND (*LESS (*TOP PEGC (3 4 5) (2) (1)) 3)). Then information for disk configurations is generalized by substituting variables $A, $B and $C into (3 4 5), (2) and (1).

After that, the system continues problem solving by the negative move-pattern strategy. It generates the correct move, (CAN 1 PEGC PEGB), and the current configuration becomes (PEGS (3 4 5) (1 2) NIL). Now the current state satisfies (*LESS 3 (*TOP PEGC (3 4 5) (1 2) NIL)). Also the system retains CONDs deposited earlier. Thus, the system understands now reaching the point where the difference detected earlier by the means-ends analysis does not exist. One of subroutines then picks up the move operator applied most recently, which is (MOVE 1 PEGC PEGB), and transforms it to (ACTION (DEP (GOAL 1 PEGC PEGB))). (GOAL 1 PEGC PEGB) is the operator which is directly responsible for elimination of the difference: it changes the configuration to (PEGS (3 4 5) (1 2) NIL), which makes (*LESS 3 (*TOP PEGC $A $B $C)) true.

The specific values 1, PEGB and PEGC in the ACTION must be substituted by functions of some of specific values and variables appearing in CONDs stored in WM for constructing a well-defined production. In the present case, peg-names PEGA and PEGC are substituted by variables $P and $R, respectively. The disk-name 3 is substituted by $X. PEGB and 1 appearing *only* in the ACTION must be represented by those $P, $R and $X. First, the system uses a function *OTHER to represent PEGB: PEGB is equal to (*OTHER PEGA PEGC : PEGA PEGB PEGC). Second, the system retrieves a function *NEXTSMALLER to represent 1: 1 is equal to (*NEXTSMALLER 3 PEGC (3 4 5) (2) (1)). (See Table 2 for definitions of functions.) Last, the system generalizes those lists to generate (*OTHER $P $R : PEGA PEGB PEGC) and (*NEXTSMALER $X $R $A $B $C). As a result, the following lists are now retained in WM:

(COND (PEGS $A $B $C)), (COND (GOAL $X $P $R)), (COND (*EQUAL $X (*TOP $P $A $B $C))), (COND (*LESS (*TOP $R $A $B $C) $X)), (COND (ABSENT (GOAL (*NEXTSMALLER $X $R $A $B $C) $R (*OTHER $P $R : PEGA PEGB PEGC)))), (ACTION (GOAL (*NEXTSMALLER $X $R $A $B $C) $R (*OTHER $P $R : PEGA PEGB PEGC))).

The last COND in the above is a modified copy of the ACTION. It will be necessary for refractory inhibition in production firing.

Finally, the system creates a production <N8> from the above CONDs and ACTION. It is shown in Table 1.

The system continued problem solving and production creation in the above manner. Whenever no GOAL-type subgoal resided in WM, a GOALSTACK-type subgoal was popped up, and used as a subgoal. Run 3 ended after productions fired 371 times. Four new productions were generated during the run, which are shown in Table 1.

It should be noted that, in the earlier stage of Run 3, the negative move-pattern strategy was used to help problem solving. The general production-creation mechanism used in Run 3 needs some other method for problem solving while it is *waiting* for *seeing* elimination of the difference between current and goal states. Fig. 2 illustrates firing behavior of created productions. Data in the interval 1600 - 1700 show clearly how productions <N1> - <N7>, i.e., the negative move-pattern strategy, worked for learning a second strategy. The negative move-pattern strategy was thus even *necessary* for learning a means-ends strategy in our computational process.

<Run 4>

Run 4 examines whether productions created in Run 3 are sufficient for a well-defined strategy. Initially the system had (PEGS (1 2 3 4 5) NIL NIL) and (GOAL 5 PEGA PEGC) in WM. Using them, <N10> fired four times successively, and deposited (GOAL 4 PEGA PEGB), (GOAL 3 PEGA PEGC), (GOAL 2 PEGA PEGB) and (GOAL 1 PEGA PEGC) into WM. Applying these subgoals, and creating other new subgoals by <N8> - <N11>, the program succeeded in solving the problem. The results show that <N8> - <N11> actually constitute a means-ends strategy, coordinating with some other a priori productions[7]. We call this newly learned strategy the *recursive subgoal strategy*.

Productions fired 280 times in Run 4. Since nothing but execution of the recursive subgoal strategy was done in Run 4, 280 is the precise number of times productions fired in running the strategy. Comparing it with 656 in Run 2, we realize that the recursive subgoal strategy may be efficient in time.

<Run 5>

After the recursive subgoal strategy was learned, the system need not use the negative move-pattern strategy unless it wants some information necessary for running the recursive subgoal strategy. Thus now the system is not bothered by attending to various disks and pegs and searching for many legal moves. It is only necessary to treat with GOALs generated by the productions <N8> - <N11>.

This facilitates the system to discover some pattern in subgoal structure. The theory indicates that the strategy learning process proceeds from a means-ends type strategy to a working-forward one. Run 5 is devoted to this: the system tries to learn a working-forward strategy by finding pattern in subgoal sequences.

It is not the way for the system to generate a very long sequence of GOALs, stores it in WM, and tries to induce pattern in it. Rather, a different kind of information is used for *restricting* a sequence of GOALs to a very short one. Pattern induction is then tried for that short sequence. The used information is a perceptual predicate called PSHAPE. A peg is said to be PSHAPEd when Disks 1, . . . , k (k > 1) are the only disks on that peg (see Table 2). A sequence of GOALs generated and deposited into WM is restricted in the following way: if PSHAPE of some peg was detected, the

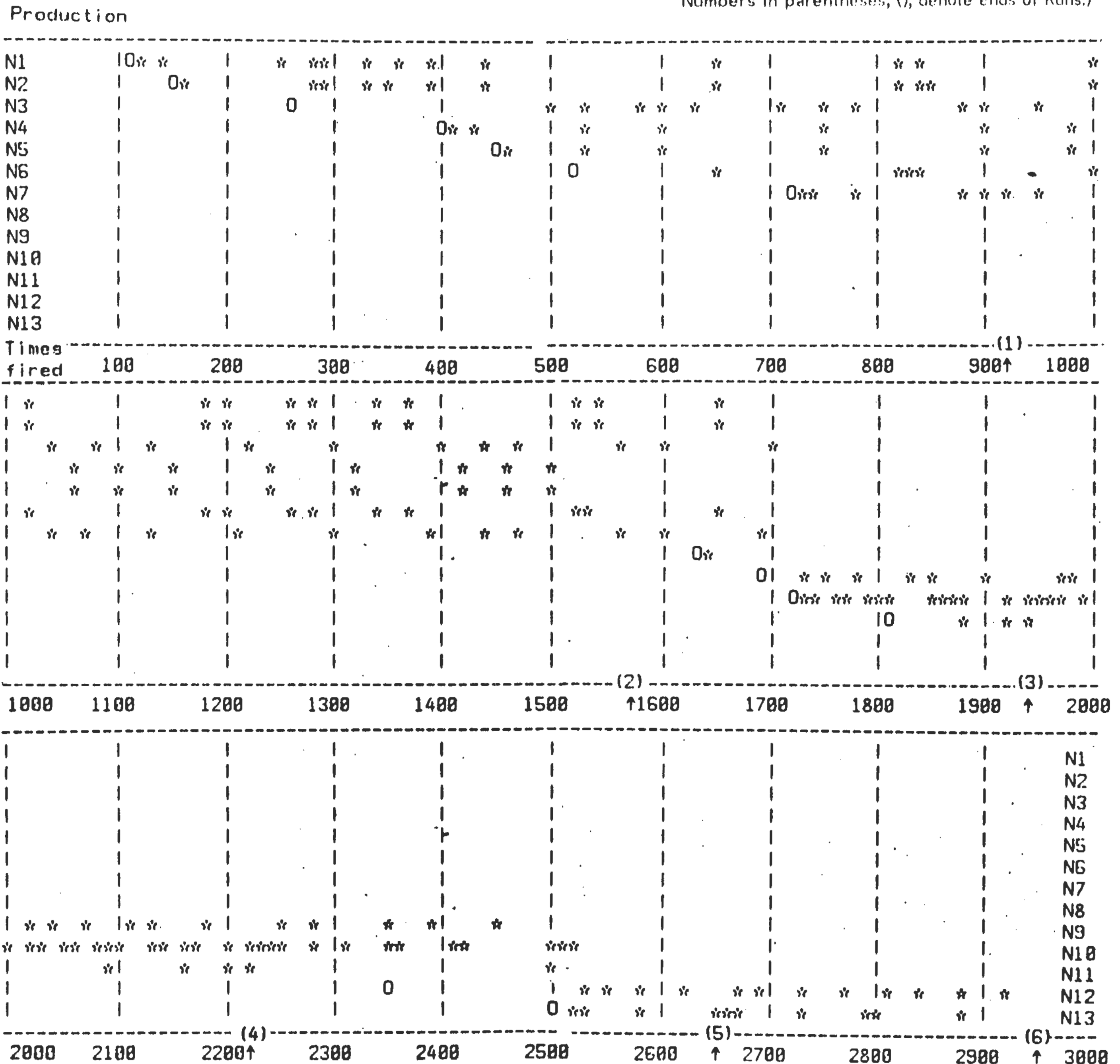
[7] After <N9> was created, <N8> never fired because it was dominated by <N9>. Thus, <N8> was necessary for learning the strategy, but useless after the strategy was established.

system starts labelling GOALs generated thereafter. Then when the system perceived that disks in that PSHAPE were all moved to another peg, labelling is terminated. The sequence of labelled GOALs is served for pattern induction. Furthermore, perception of PSHAPE occurs only when some unapplicable subgoal triggers it. It will be shown by an example below.

Suppose that, using the recursive subgoal strategy, the system reached the disk configuration, (PEGS (4 5) (1 2) (3)). At that moment, the system retains an unapplicable subgoal (GOAL 4 PEGA PEGB). This subgoal makes the system attend to Peg B, the goal peg in that subgoal. Then PSHAPE is perceived at Peg B, since attainment of the subgoal is interfered only by the *subpyramid* (1 2) on Peg B. The perception lets the system deposit (GOALSEQ (PGOAL 2 PEGB PEGC) ((GOAL 1 PEGB PEGC))) and (FOR-PGOAL 4 PEGA PEGB) into WM after the next proper subgoal (GOAL 1 PEGB PEGC) is created by the recursive subgoal strategy. In the first list, (GOAL 1 PEGB PEGC) is just a copy of the most recently generated GOAL. (PGOAL 2 PEGB PEGC) denotes a subgoal for the perceived subpyramid. The second argument in GOALSEQ is a subgoal stack into which GOALs related to moving the subpyramid will be stored.

Fig. 2 Production-firing process for whole run

(O denotes generation of a new production.
* denotes firing of a new production.
Numbers in parentheses, (), denote ends of Runs.)



After GOALSEQ is generated, the system continues solving the problem using the recursive subgoal strategy. In that process, GOALs are added to the second argument of GOALSEQ if those GOALs are found to be applied. This growing process terminates when the system attained (PEGS (4 5) NIL (1 2 3)) using the recursive subgoal strategy. The resulting GOALSEQ has the form: (GOALSEQ (PGOAL 2 PEGB PEGC) ((GOAL 1 PEGA PEGC) (GOAL 2 PEGB PEGC) (GOAL 1 PEGB PEGA))). Here the system finds that the currently attended subgoal (GOAL 4 PEGA PEGB), which has been noticed for a while by retaining (FOR-PGOAL 4 PEGA PEGB) in WM, is immediately applicable.

The program then tries to relate specific values in the second argument of GOALSEQ to values in its first argument. It is done by using functions such as *OTHER and *NEXTSMALLER. For instance, 1 in (GOAL 1 PEGB PEGA) is the disk NEXTSMALLER than 2 of (PGOAL 2 PEGB PEGC) at the disk configuration (PEGS (4 5) NIL (1 2 3)). PEGA in (GOAL 1 PEGB PEGA) is the OTHER peg than PEGB and PEGC both appearing in (PGOAL 2 PEGB PEGC).

Next the system substitutes variables into specific values in GOALSEQ. Then CONDs and ACTIONs are generated from GOALSEQ and other current knowledge elements:

(COND (*P-ON 2 $Q $A $B $C)), (COND (PGOAL 2 $Q $R)), (COND (ABSENT (GOAL (*NEXTSMALLER 2 $Q $A $B $C) $Q (*OTHER $Q $R : PEGA PEGB PEGC)))), (COND (ABSENT (GOAL 2 $Q $R))), (COND (ABSENT (GOAL (*NEXTLESS 2) (*OTHER $Q $R : PEGA PEGB PEGC) $R))), (COND (PEGS $A $B $C)), (ACTION (REM (PGOAL 2 $Q $R))), (ACTION (DEP (GOAL (*NEXTSMALLER 2 $Q $A $B $C) $Q (*OTHER $Q $R : PEGA PEGB PEGC)))), (ACTION (DEP (GOAL 2 $Q $R))), (ACTION (DEP (GOAL (*NEXTLESS 2) (*OTHER $Q $R : PEGA PEGB PEGC) $R))).

The first and sixth CONDs were derived from information other than GOALSEQ. The second COND is the first argument of GOALSEQ. The third, fourth and fifth CONDs are modified copies of some of ACTIONs. The first ACTION was derived from the first argument of GOALSEQ. The other ACTIONs were mapped from the second argument of GOALSEQ.

Finally, the system creates a production from these CONDs and ACTIONs. The production, <N12>, is shown in Table 1. It generates a sequence of three GOALs equivalent to a PGOAL given in the condition side. Note that the disk-name, 2, is not generalized in <N12> since the system regards 2 as one of small and *specific* disks. This special assumption was derived from the experimental data. Thus, <N12> fires only for (PGOAL 2 . . .). For example, if (PGOAL 2 PEGB PEGC) made <N12> fire, the *sequence* of subgoals (GOAL 1 PEGA PEGC), (GOAL 2 PEGB PEGC) and (GOAL 1 PEGB PEGA) are stored into WM in this order. <N12> is a *working-forward* production, since it generates a sequence of GOALs which can be applied not backwards, but forwards.

After <N12> is created, the system continues problem solving by the recursive subgoal strategy. At (PEGS (5) (4) (1 2 3)), the only GOAL the system retains is (GOAL 5 PEGA PEGC). Thus, in the same manner as above, this subgoal triggers perception of PSHAPE, (1 2 3), on Peg C. Hence the system executes the same process as above: first constructs (GOALSEQ (PGOAL 3 PEGC PEGB) ((GOAL 1 PEGC PEGB))). (GOAL 1 PEGC PEGB) is the immediate next subgoal generated by the recursive subgoal strategy. When (PEGS (1 2 5) (4) (3)) is encountered, GOALSEQ has grown to: (GOALSEQ (PGOAL 3 PEGC PEGB) ((GOAL 1 PEGB PEGA) (GOAL 2 PEGC PEGA) (GOAL 1 PEGC PEGB))). At this point, the system finds using the *experience*[8] that the second argument of GOALSEQ is equivalent to (PGOAL 2 PEGC PEGA). Thus GOALSEQ here is transformed to (GOALSEQ (PGOAL 3 PEGC PEGB) ((PGOAL 2 PEGC PEGA))).

Then the system continues problem solving. When the subpyramid (1 2 3) was moved to Peg B, i.e., (PEGS (5) (1 2 3 4) NIL) was attained, GOALSEQ has the form: (GOALSEQ (PGOAL 3 PEGC PEGB) ((PGOAL 2 PEGA PEGB) (GOAL 3 PEGC PEGB) (PGOAL 2 PEGC PEGA))). As same as before, the system generalizes this list, and creates CONDs and ACTIONs from it. Finally, a new production, <N13> shown in Table 1, is generated. Different from the case of creating <N12>, the disk-name, 3, is also generalized in the present case because the system notices that 3 is one of bigger, *general* disks in the given problem.

Though the system can solve the problem using the recursive subgoal strategy after this point, we make the following assumption here: once the system is accustomed to perceiving PSHAPEs, it can generate a PGOAL if no PGOAL is available and a PSHAPE is attended to. This heuristic is stored in the program in the form of a production. Thus at (PEGS NIL (1 2 3 4) (5)), where no PGOAL is in WM and Peg B is PSHAPEd, the system can generate a new subgoal (PGOAL 4 PEGB PEGC). This is done first by generating (GOAL 4 PEGB PEGC) using <N11> (a production for the recursive subgoal strategy), and then transforming it to (PGOAL . . .) using the fact that Peg B is PSHAPEd.

Run 5 ended successfully after productions fired 431 times and two new productions <N12> and <N13> were generated. These productions, together with some other a priori productions, may constitute a new strategy. That it is actually so will be examined in Run 6. Note that no heuristic search was made in Run 5, and the recursive subgoal strategy was fully used for generating new productions. It is clearly seen in production-firing behavior between 2200 - 2650 of Fig. 2. Also note that a somewhat special perceptual predicate, PSHAPE, was effectively used for limiting data for pattern induction in subgoal stacks. This mechanism was derived essentially from our experimental data.

<Run 6>

Run 6 examines whether productions learned in Run 5 are sufficient to constitute a strategy. Run 6 performs only this task. In this sense, it corresponds to Run 4, and weakly to Run 2.

As the system was then used to PSHAPEs, it initially generated a subgoal (PGOAL 5 PEGA PEGC) by attending to the initial pyramid, (1 2 3 4 5), on Peg A. Then, using this subgoal, and productions <N12> and <N13>, the system succeeded in attaining the final goal with no error. Thus the system learned a new well-defined strategy. We call it the *working-forward strategy*, since <N12> and <N13> generate subgoal sequences that can be applied not backwardly, but forwardly. Run 6 consumed 296 times of production firing.

[8] Of course the system should have remembered the former GOALSEQ information for doing this.

Comparing with Run 2's 656 times and Run 4's 280 times, the working-forward strategy may be much more efficient than the negative move-pattern strategy but comparable with the recursive subgoal strategy, with respect to time.

The number of productions learned is seven for the negative move-pattern strategy, four for the recursive subgoal strategy, and two for the working-forward strategy. A more sophisticated mechanism would decrease the number for the negative move-pattern strategy down to three (or even to two). Thus, efficiency in the number of productions necessary to be learned would be comparable for the three strategies if slight modification is allowed to the current version of the program.

As for load on WM, the recursive subgoal strategy seems to be the smallest. Fig. 3 shows dynamic behavior of the number of elements contained in WM. The negative move-pattern strategy puts heavy load on WM by its combinatorial attention to pegs and search for legal moves. The working-forward strategy also puts burden on WM by its power for generating several subgoals at once. Fig. 3 also illustrates clearly a qualitative change of operators in the learning process. The number of move operators gradually decreased and became more stable, whereas subgoal operators played an active role in the later stage.

In spite of the above results, people seem to regard the working-forward strategy as the most sophisticated strategy for solving the Tower of Hanoi problem. This is caused from the fact that the working-forward strategy involves subgoals like PGOALs. PGOALs are higher-level subgoals, which may state very simply a subgoal sequence that can be applied successively to the initial problem state, and bring the system even to the final goal. (PGOAL 5 PEGA PEGC) is decomposed to the sequence (PGOAL 4 PEGA PEGB), (GOAL 5 PEGA PEGC) and (PGOAL 4 PEGB PEGC). The three subgoals in this sequence are applied successively to the initial state, and provide the final state. The theory suggests that this capability of directly connecting the initial state to the final goal is one of ultimate results in the long-term strategy learning process. As Run 6 shows, the program finally learned this capability.

Implication of Work and Its Relation to Other Works

The computational results presented in this paper imply two things: (1) the program is one of first efforts toward modelling human learning behavior using an adaptive production system, and (2) there is a way that a computer can learn strategies through its own experience. The second point is particularly related to research on artificial intelligence systems that have capabilities of discovering complex procedural knowledge. The paper provided a positive evidence for possibility of computational study on strategy learning processes.

Much effort has been made so far toward designing artificial intelligence systems that are able to learn procedural knowledge. One of the mainstreams in the effort is utilization of adaptive production systems, i.e., production systems that incorporate an ability for creating production rules, which was first constructed formally by Waterman (1974). Meta-DENDRAL of Stanford Heuristic Programming Project is one of earliest successes in application of such study. The work presented in the paper and other two papers (Anzai, 1978; Anzai & Simon, 1977) may be considered as a successor of this stream, applied particularly to learning complex procedural knowledge. Broadly speaking, work on automatic programming (e.g., Green, 1977) may be considered as a part of the effort. Actually the results presented in the paper can be regarded as a small computational experience on automatic programming in a more general problem-solving oriented domain.

Also, in parallel with the work in artificial intelligence, learning procedural knowledge has been studied recently by cognitive psychologists (e.g., Anderson, 1977). Those two trends, with general study of induction in complex problem solving tasks (e.g., Simon & Lea, 1973), are now converging to work on systems for learning complex procedural knowledge.

This paper is hopefully one of initial steps toward this direction. The results are encouraging, but still limited. Various problems which did not arise in the theoretical exploration have been made explicit in this computational study: the problem of error recovery, relative speed of learning, finer structure of representation, time efficiency of production systems, generality of the program, and so on. Most of these issues are currently being attacked in the continuing project on computerized strategy learning.

References

Anderson, J. R. Induction of augmented transition networks. *Cognitive Science*, 1977, *1*, 125-157.

Anzai, Y. How one learns strategies: Processes and representation of strategy acquisition. Submitted for publication, 1978.

Anzai, Y., & Simon, H. A. Strategy transformation in problem solving: A case study. CIP Working Paper 372, Dept. of Psychology, Carnegie-Mellon University, Pittsburgh, Pa., 1977.

Buchanan, B. G., & Mitchell, T. M. Model-directed learning of production rules. Comp. Sci. Report STAN-CS-77-597, Stanford University, Stanford, Calif., 1977.

Green, C. A summary of the PSI program synthesis system. Proc. 5th IJCAI, 1977, 380-381.

Schank, R. C., & Abelson, R. P. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Assoc., 1977.

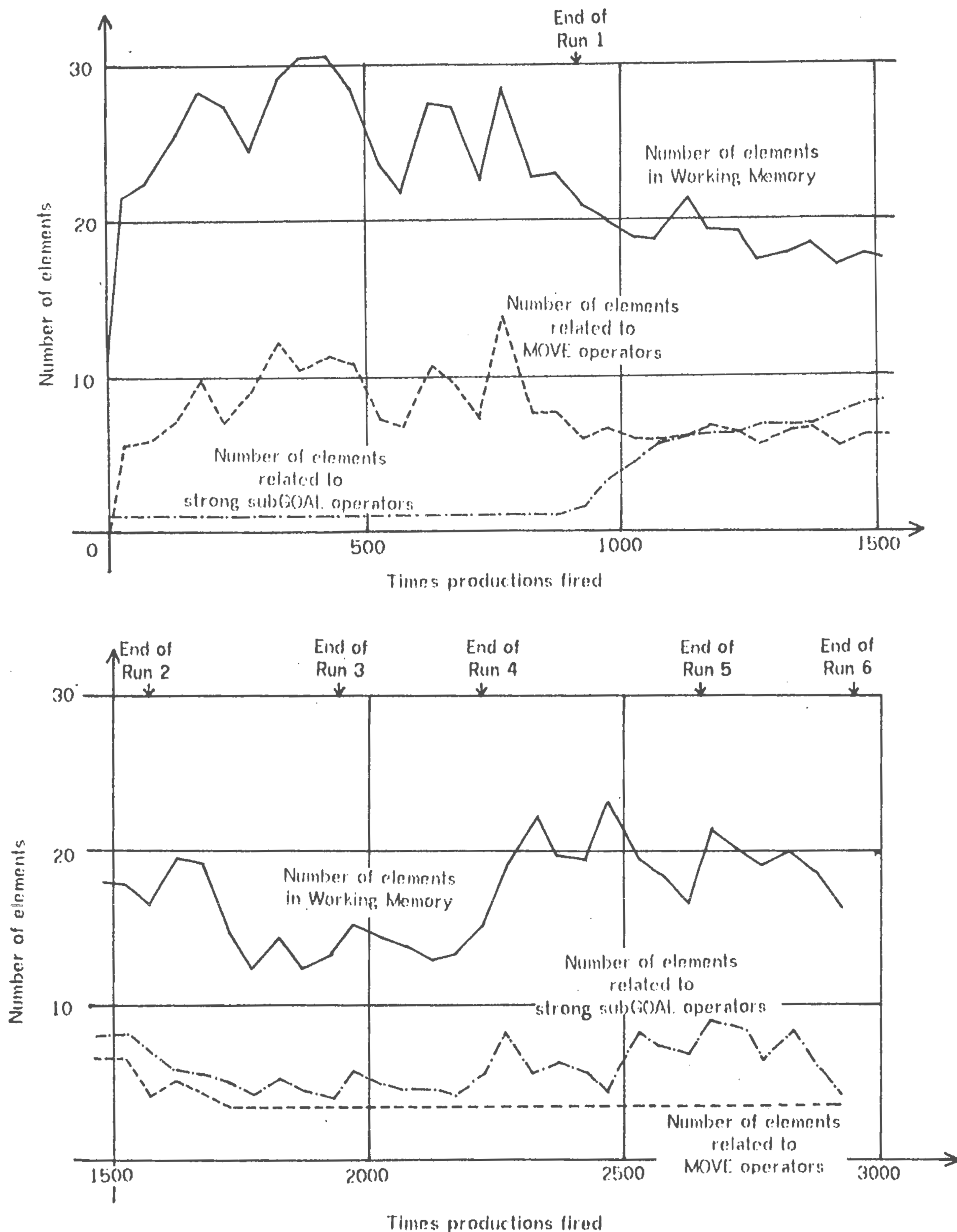
Simon, H. A. The functional equivalence of problem solving skills. *Cognitive Psychology*, 1975, 7, 268-288.

Simon, H. A., & Lea, G. Problem solving and rule induction: A unified view. In L. W. Gregg (Ed.), *Knowledge and Cognition*, Lawrence Erlbaum Assoc., 1974, 105-127.

Waterman, D. A. Adaptive production systems. CIP Working Paper 285, Dept of Psychology, Carnegie-Mellon University, Pittsburgh, Pa., 1974.

Fig. 3 Number of elements in Working Memory

(Numbers averaged for every 50 times of production firing)



A computer program that learns algebraic procedures by examining examples and by working test problems in a textbook

David M. Neves
Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

This paper describes a computer program, written in LISP, that acquires procedures by examining worked-out example problems and by working test problems in an algebra textbook, Stein & Crabill (1972).

The paradigms of learning by example and learning by doing have received much attention in the past few years (e.g. Waterman(in press), Anzai(1978), Sussman(1975)). A common characteristic of this recent work is that the learned procedures take the form of production systems (Newell, 1973; Newell & Simon, 1972; Davis & King, 1975). Production system languages consist of two memories, a production memory and a data (working) memory. Production memory consists of a set of condition/action production rules. The condition part is compared to working memory and if true, the action side is executed. Production systems are especially well-suited for the learning of procedures because of their modularity (see Waterman, 1975). A learned rule can simply be added to production memory.

The basic idea in learning by example is to induce the production rules used by the expert who generated the example. Each pair of lines, or states, in the example leads to the learning of one production rule, with some part of the input (the first line) as a condition and the operation performed on the first line as the action. In learning by doing, the example trace is generated by the student and not by the expert. It could lead to the induction of the same productions but would take much longer than by learning from an existing example.

In constructing mechanisms for learning, the nature of what is learned, and the method of learning must be specified. The result of learning can be characterized as a performance system as defined by Newell & Simon (1972). Such a system (like the General Problem Solver) needs a goal, a representation of the problem, operators, a "table of connections" (a data base that indexes the operators by the changes they make) and working-forward production rules of the form, "If X occurs, then do Y". The learning system described here learns all the above except for the representation.

At the present time the program learns to solve linear algebraic equations from a textbook. The textbook provides explanatory text, annotated example problems, and work problems at the end of the section. The program uses the examples and test problems to learn and ignores the written text.

THE SYSTEM

The program is given a knowledge of arithmetic: the representation (objects and relations between objects), operators, table of connections,and goals. It is also given the representation of algebra. The program then learns the goal state of solving linear algebraic equations, learns the operators (composed of condition/action production rules), and adds to the table of connections a recognition of algebraic operators. Figure 1 shows what the result of the learning process might look like. The six productions solve an equation by moving numbers to the right side of the equation, by moving terms with an "X" to the left, and by combining like terms.

A simple flowchart of the program is shown in Figure 2. There are three main components, Example, Perform, and Learn. Example takes as input a work-out example problem and calls Learn. Perform takes as input a problem to be solved. It uses general problem solving techniques (Newell, 1969) to generate its own example trace and also calls Learn. Learn takes as input two lines and creates a production rule.

P1. **If there is a number on the left hand side of an equation, then subtract it from both sides.**

P2. **If there is a term with "X" in it on the right hand side, then subtract it from both sides.**

P3. **If there are two like terms on the left hand side, then combine them.**

P4. **If there are two like terms on the right hand side, then combine them.**

P5. **If the equation is reduced to "<number> * X = <number>", then divide both sides by the number in front of the "X".**

P6. **If the equation is reduced to "X = <number>", then STOP.**

Figure 1. A production system for algebra

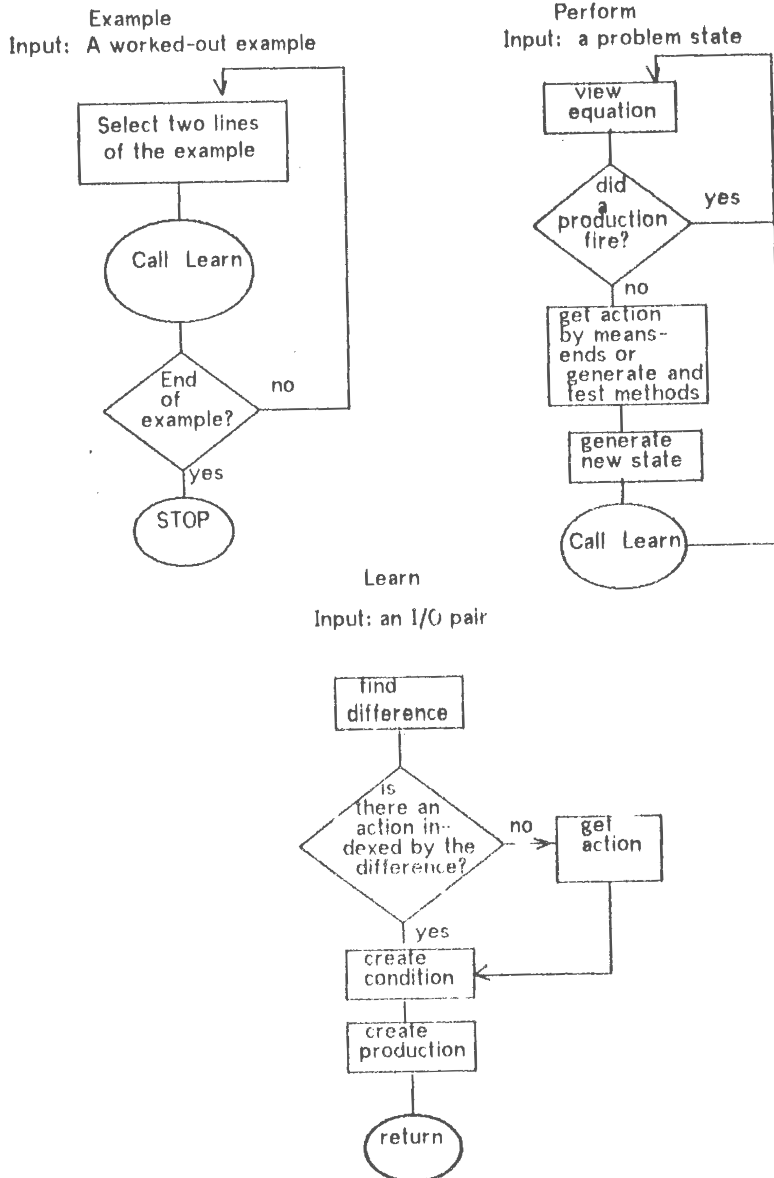


Figure 2. Flow charts of the three programs

EXAMPLE

As stated above, the Example program inputs a worked-out example. It sends pairs of consecutive lines to Learn, where a production is built. When Example reaches the last line two things are done. First, it interprets the example as the workings of an operator, with an input (the first line) and an output (the last line). In order to recognize that operation in the future (as part of another example) the change is computed between the input and output lines. The operator is then indexed by that change in the table of connections. Second, the last line is represented and stored on the goal list of the operator. So, Example adds to the table of connections and also learns the goal states of operators.

LEARN

The Learn program takes as input a pair of lines from the Example or Perform programs. It generates a production rule which, when presented with the first of the two lines, or similar input, will execute the appropriate operator. This program is the most important in the system and each of its components is described in detail below.

Represent

The two lines are first sent to a function that represents them. The lines are lists of characters. This function chunks characters into objects (e.g. consecutive digits are put together to form an integer) and relates objects to other objects in the line (i.e. terms are represented according to their relationship to the equal sign).

The knowledge of how to represent algebraic equations is given to the program and so is not learned. Below is an example of how two lines are represented.

x - 15 = 2x	(left +x)(left -15)(right +(2*x))
-x - 15 = 0	(left -x)(left -15)(right +0)

The form for the representation is (<relation><object>). In the above example "left" means to the left of the equal sign and "right" means to the right of the equal sign. The sign of each term (+ or -) is chunked with the term. Also, the representation is put in a canonical form, i.e. 2x is changed to 2*x.

Find Difference

Next, the two represented lines are sent to a general procedure that computes the difference between the two lines. The difference is a list of symbols that have been REMoved from the first line, TRANSformed from the first line, and symbols that have been ADDed to those already on the first line of the example. For the two lines below

```
x - 3 = 5     (left +x)(left -3)(right +5)
x = 5 + 3     (left +x)(right +5)(right +3)
```

the difference is: (rem (left -3))(add (right +3)).

That is, a minus 3 has been removed from the left and a plus 3 has been added to the right.

The difference is computed in two steps. First, the two lines are checked to see which symbols have been added and removed. For example, if a symbol in the first line is not equal to any symbol on the same side of the equation in the second line then it has been removed. Then the program checks to see if there have been symbols added and removed on the same side of the equation (i.e. they have the same relation -- left or right). If so, the REMove is changed to a TRANSform. This creates three kinds of changes; removing, transforming, and adding. The example below show these two steps.

```
x - 3 = 5     (left +x)(left -3)(right +5)
x = 8         (left +x)(right +8)
```

The initial difference is:
(rem (left -3))(rem (right +5))(add (right +8))

which then becomes:
(rem (left -3))(trans (right +5))(add (right +8))

Generalize

Next, the objects in the difference are generalized. Knowledge of what to generalize and the scope of generalization are given to the system and are assumed as part of the representation. For algebra the function generalizes over numbers. For any number it substitutes the symbol "N8", which is the concept name for number, so that (left -3) becomes (left -N8).

Creating the production

At this point we have characterized the difference between the two lines. This difference is used to access the action that produced it, and is used in creating a condition for a condition/action production rule. Suppose the following two lines are input to the Learn program.

```
x - 3 = 5
x = 8
```

The difference is:
(rem(left -n8))(trans(right +n8))(add(right +n8)).

The program checks the table of connections to find an operator that will produce that difference. The program may find the operator ADD-AND-SIMPLIFY, which is made up of a set of productions that will add a number to both sides an equation, and then simplify it. This operator was learned in a previous example given to the system.

The task, now that an action has been retrieved, is to attach the appropriate condition to the action and store it away as a production. The most specific condition would simply be the first line of the example, (x - 3 = 5). That is, if you see "x - <number> = <number>" then add the second number to both sides and simplify. However, intuitively, the appropriate condition would seem to be just the negative number on the left hand side of the equation, (left -n8). One way of whittling down a condition side would be to present several examples where the same operator is applied and keep only those symbols which are common to all of the examples. This is similar to the concept learning scheme used by Winston (1975). However, there is no such instruction in textbooks and there is often only one example in the book before the test problems.

The rationale for the way the condition is determined is as follows: Algebra is a domain in which objects are manipulated: removed, transformed, or copied. The mode of operation is to notice something and then change it. If we work backwards from this notion we can infer the reason (or condition) for an action by looking at what it affects.

Two heuristics are used to find the condition for an action.

> 1) If the result of an action on the environment is observed, then the probable condition for that action was the group of symbols that was affected by the action.

This heuristic is based on the reasoning described above. It examines the *difference* between two states, which contains the changed symbols.

> 2) Only a subset of the changed symbols are used as the condition for the action. The kind of change determines whether it is included in the condition.

If there are REMs in the difference then only the symbols that were removed are put on the condition side. If not, and if there are TRANSs, those symbols alone are put on the condition side. Otherwise, another procedure is executed for the ADDs. In the above example, the difference has one symbol removed,(left -n8), and so it is put on the condition side of the production. The operator for the action side was retrieved using the table of connections. The resulting production is:

(left -n8) --> ADD-AND-SIMPLIFY(n8).

That is, if there is a negative number on the left side of an equation, then add it to both sides and simplify.

If the program is not able to retrieve an operator from the table of connections, then several other procedures must be executed in order to determine what operator (or group of operators, in the case where steps are skipped) has been applied. There are five subprocedures that can be called when the difference has not retrieved an action. These procedures search for an action. The procedures are:

1) Ask the instructor for the name of the procedure.
2) Use primitive operators for the action.
3) Use means-ends to fill in skipped steps.
4) Use a partial match on the difference.
5) Try another representation.

Three of these procedures are explained below.

Ask for the name of the procedure

The examples in the textbook are annotated. Although they do not give the conditions for actions, they generally give the action applied to each line in the example. The program does not make use of this information. However, if it did, it would translate the verbal description in the example to the name of an action in memory. As a crude approximation to this process, the name of the action can be directly supplied to the program.

Use primitive physical operators

If the system is unable to retrieve an action that will produce the difference, then it has the option of creating its own action with its primitive operators. There are three primitive physical operations that correspond to the three kinds of changes in the difference. Prim-add adds a symbol to the environment(writes it on paper), Prim-rem removes a symbol from the environment, and Prim-trans transforms a symbol into another symbol.

In the example below a minus 2 has been added to both sides of the equation.

$x + 2 = 5$
$x + 2 - 2 = 5 - 2$

The primitive action, Prim-add (which should not be confused with an arithmetic operator) could be used as the action. The two Prim-add actions would be:

(Prim-add (left -n8)) -- add a number to the left
(Prim-add (right -n8)) -- add a number to the right

Use another representation

Information in the table of connections is stored in a represented form. The operators in arithmetic, for example, cannot be used or recognized unless the arithmetic representation is being used. So, when learning algebra the program must be able to go back to an arithmetic representation to use its arithmetic operators.

When first learning about algebra, the example below is not recognized as addition.

Algebra example:
x = 2 + 5 (left +x)(right +n8)(right +n8)
x = 7 (left +x)(right +n8)

Difference:
(trans (right +n8))(trans (right +n8))(add (right +n8))

Below we see how the same addition is represented in arithmetic.

Arithmetic example:
2 + 5 (linear +n8)(linear +n8)
7 (linear +n8)

Difference: (trans (linear +n8))(trans (linear +n8)),
(add (linear +n8))

The right-hand side can be recoded into the arithmetic representation and will be seen as the transformation of two positive numbers into another number. This is an operation known in arithmetic.

After the program retrieves an action with one of the above methods it adds the action to the table of connections, indexing it by the difference. The action will be recognized in later example problems. Then a working forward production is created with the appropriate condition.

Summary of Example and Learn

In the above examples, we have seen that the two programs learn procedures (operators composed of production rules), the goals of the procedures, and they learn how to recognize those procedures by the differences that they produce. Although there are many mechanisms involved in learning the few pages in the textbook, these may be general mechanisms that can be used elsewhere in the textbook.

PERFORM

After the Example program is given several examples, the Perform program is given the test problems at the end of the section to work on. Perform uses the working forward productions built up by the Learn program to solve the problems. For the first few problems, productions will be available that will recognize what to do. However, at some point the production system will halt because no production fires. At this point Perform uses means-ends analysis (using the existing table of connections), or the generate and test weak problem solving methods (Newell, 1969) to generate the next step in the solution. When it has come up with an action that reduces the difference between the current state and the goal state, it applies the Learn program to create a production.

Summary and Conclusion

Three programs were described that learn from examples and learn by doing. The Example program scans two lines of an example problem at a time, sending them to the Learn program. The Learn program determines the operation that took place to create the second line from the first. It then uses two heuristics to determine the condition for the application of the operator. Once it has the condition and the action it creates a production rule and stores it in production memory. The Perform program works on test problems, using general, but time-consuming problem solving techniques to solve the problems. As it generates each step it calls the Learn program to create a production rule with an appropriate condition attached to the operator that was applied to the equation. It is interesting to note that no new data structures, other than the ones used in a performance system, like GPS, needed to be created for the Learn program. The table of connections is used both to access operators while going through an

example and to retrieve operators that reduce the distance to the goal while problem solving.

There are several reasons for the success of the system. The use of production systems as a representation for procedures makes the learning process easier. It would be more difficult to assimilate the learning into a more complex control structure like that of Fortran, for example. The two heuristics described earlier enable the system to immediately create productions of more general applicability than would be possible using a discrimination procedure. Finally as the system learns operators it also learns to recognize when those operators have been applied. This learning greatly increases the recognition power of the table of connections, so that more complex examples can be used. So, not only do the programs learn components for a complete performance system, like operators and goals, but their learning facilitates later learning by the indexing of higher level operators in the table of connections.

The key part of the system is determining the condition for an operator applied to the first of a pair of lines in an example. The system is able to do this because all the relevant information is contained in the equations. However, in some domains the external stimulus might not contain all the information relevant to the condition. For example, Waterman (in press) has constructed a system that observes a person performing various functions on a computer, such as retrieving data files from other computer sites. One piece of information that is put on the condition side of some of his productions is the type of operating system of the computer that the files are being accessed from. It might be the case that this piece of information cannot be determined by the example lines being examined. The system must know that the information is needed and must be able to retrieve it. In Waterman's system, the knowledge of what is relevant information for the condition is given to it. It is not clear how a system could learn about such relevant information.

Status and Future work

As of May, 1978, the Example program has been implemented and debugged. The Learn program has been implemented and mostly debugged, while the Perform program has been implemented, but not debugged.

After testing the system's ability to learn from the textbook, the capabilities of the system will be increased. The system does not learn the representation of the domain it works on. It is not clear yet what kinds of additional mechanisms will be needed to enable the system to do this. Also the current system does not test the rules it learns. As rules are learned from an example their generality could be tested by simulating them on the example problem. If a rule applies before it should (compared to the example) then it may be too general. Its specificity could be increased by adding more elements to the condition side of the rule. Finally the system will be extended to domains such as physics to test the generality of its mechanisms.

Acknowledgments

This research was supported in part by NIMH Grant GH-MH-06718, and in part by ARPA Grant F44620-73-C-0074.

I gratefully acknowledge the assistance of Jola Jakimik during the preparation of this paper, and thank Herbert Simon for his help in all phases of this research.

REFERENCES

Anzai, Yuichiro. Learning strategies by computer. Paper presented at The Second National Conference of the Canadian Society for Computational Studies of Intelligence, Toronto, Canada, 1978.

Davis, R. & King, J. An overview of production systems. Report STAN-CS-75-524, Memo AIM-271, Department of Computer Science, Stanford University, 1975.

Newell, A. Heuristic Programming: Ill structured problems. In J. S. Aronofsky (Ed.), *Progress in Operations Research* (Volume III). New York: John Wiley & Sons, 1969.

Newell, A. Production systems: Models of control structures. In W. C. Chase (Ed.), *Visual Information Processing*. New York: Academic Press, 1973, pp. 463-526.

Newell, A. & Simon, H. *Human Problem Solving*. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Sussman, Gerald J. *A Computer Model of Skill Acquisition*. New York: American Elsevier Publishing Co., 1975.

Waterman, D. A. Adaptive production systems. Proceedings of the Fourth International Joint Conference on Artificial Intelligence. Tbilisi, USSR, 1975, pp. 296-303.

Waterman, D. A. Exemplary programming. In D. Waterman & R. Hayes-Roth (Eds.), *Pattern-directed Inference Systems*. New York: Academic Press, in press.

Winston, P. H. Learning structural descriptions by examples. In P. H. Winston (Ed.), *Psychology of Computer Vision*. New York: McGraw-Hill, 1975.

COOPERATIVE RESPONSES: AN APPLICATION OF DISCOURSE INFERENCE TO DATA BASE QUERY SYSTEMS*

S. Jerrold Kaplan and Aravind K. Joshi
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pa. 19104

For Natural Language (NL) systems to interact effectively with non-expert users of computers, they must be capable of dealing in an appropriate way with the expectations that the user has of a cooperative speaker of the language. Because people do not have a coherent set of expectations about the way computer systems use NL (the way they do, say, about small children), users will expect these systems to provide cooperative and appropriate conversational behavior approximating that of human speakers. In a cooperative human dialog, participants observe a variety of specific conventions and principles that promote effective communication (some of these are culture specific). Failure to follow these conventions and principles results in inappropriate and/or misleading utterances.

Questions in NL do a great deal more than request information. Even simple questions frequently encode aspects of the questioners goals and intentions, as well as his or her state of knowledge. Questions 1A-1C below illustrate the encoding of goals and intentions by the different responses that they will reasonably admit.

1A. Did John borrow my coffee cup?
1B. Was it John that borrowed my coffee cup?
1C. Was it my coffee cup that John borrowed?
1D. No, it was Bill.
1E. No, it was your sugar.

Superficially, all three questions appear to convey the same request for information. A closer examination reveals that although 1D and 1E are both appropriate responses to 1A, 1B favors 1D while 1C favors 1E. 1B indicates that the questioner is interested in who borrowed the coffee cup, while 1C indicates that the questioner is interested in what John borrowed. Computational Linguists have barely begun to explore the usefulness of such cues in computational systems.

In a cooperative discourse, the secondary communication of goals and intentions is not incidental - it provides a context for an appropriate response, as illustrated by example 1. The hearer is expected to compose a response that is relevant to the questioners needs, as indicated by the question. A failure to produce appropriate, relevant responses is known as "stonewalling"*.

Many conversational conventions, while falling under the rubric of "Pragmatics" in Linguistics, are sufficiently regular and consistent as to be formalizable within a computational framework. The purpose of this paper is to describe some aspects of conversational cooperation that are essential to an effective NL system, and present an implemented query system that incorporates these conventions in a practical way. The system demonstrates that many cooperative principles can be formalized and embodied in general computational procedures to be applied to the task of data retrieval from a standard (CODASYL) Data Base (DB) query system. Projecting the more general problem of cooperation in unrestricted discourse onto the domain of a query system provides a method of both sharpening certain linguistic intuitions and reducing the problem to a tractible form without trivializing the problem or rendering the solutions ad-hoc. It is our belief that the mechanisms described here, while motivated by the domain, provide an approach that can be applied to a significantly wider class of NL processing problems.

---

* This work partially supported by NSF grant MCS 76-19466

*Stonewalling is a term used for uncooperative yet technically correct responses to questions. It was popularized during the Senate Watergate Hearings to describe the behavior of several White House witnesses.

WHY A THEORY OF COOPERATION IS ESSENTIAL

Besides the obvious frustrations accompanying the use of an uncooperative NL system, there are real dangers posed by the use of such systems in a conversational environment. In particular, it is frequently the case that correct responses to questions cause a questioner to draw incorrect or spurious inferences.

NL questions often indicate that the questioner presumes certain things to be true. It is not only possible to detect, check, and correct these conversational presumptions, but it is expected in a cooperative exchange. Failure to correct these presumptions results in an implicit affirmation of their correctness in the questioner's mind. Therefore, a failure to contradict a false conversational presumption, no matter how innocently omitted, will actively reinforce the questioners mistaken impression. In a cooperative discourse, the expectation of a cooperative response is so strong that a failure to contradict false conversational presumptions is highly inappropriate. Consider the following question-answer pairs.

2A. Did John invite his mother to his wedding?
2B. No.

3A. What is the name of Bill's first wife?
3B. Sally.

4A. Which of the B-52s with a range of 2500 miles or more are based at Camp David?
4B. None.

The response 2B to question 2A reaffirms, through its failure to state otherwise, the questioner's presumption that John got married. If 2B were uttered in a context where John is a bachelor, however, it still affirms the questioner's presumption, even though that presumption is false. Although the response is literally correct in such a context, it misleads the questioner by reinforcing, or even creating, the false belief that John got married. (A more appropriate response would be "John is not married.") 3B reinforces 3A's presumption that Bill was married more than once. In a context where Bill is still happily married to Sally, 3B is correct, but misleading. (Again, a more cooperative response would be "John has only one wife: Sally.") Similarly, 4B is a misleading response to 4A if B-52s have a maximum range of 1000 miles. "No B-52s have a range of 2500 miles or more." would be a more cooperative response to 4A. These cooperative responses are Corrective Indirect Responses in our terminology, since they respond indirectly by contradicting a false presumption.

Without a theory of cooperation, interactive NL systems will produce correct but misleading responses, fostering and even creating false impressions in their users. A formalism is presented below which produces corrective indirect responses to arbitrary DB queries. The procedures predict both when the responses are required, and what these responses should be.

AN APPLICATION TO THE DATA BASE QUERY AREA

By limiting the domain of discourse to the area of data retrieval from a DB system, it is currently feasible to produce general computational mechanisms for responding cooperatively to NL DB queries. Only by applying a theory of cooperative responses to a domain as limited as data retrieval is it possible at this time to provide computational solutions with some degree of breadth and generality. In this domain, cooperative responses can be used, for instance

1) to aid a user in formulating a suitable followup query when a precise response to the initial query would be uninteresting, useless or meaningless,

2) to inform a user about the nature of the domain (structure and content of the data base) when s/he is unfamiliar with its complexities, and

3) to organize relevant information in a fashion most suitable for its intended use.

Corrective indirect responses are particularly important in this domain, where occasional or non-expert users frequently make erroneous presumptions about the structure or content of the DB in their queries. (Example 4 above could illustrate such a failure.)

The choice of the DB query area for an implementation is a careful one. One of our objectives is to demonstrate that the DB environment provides a way of sharpening certain linguistic issues (albeit restricting them in some fashion). The area is sufficiently rich to display a wide variety of linguistic problems: problems of anaphora, opaque reference, and discourse cooperation are present in various forms, to name a few.

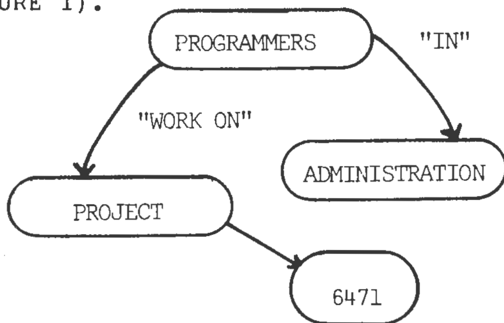
A preliminary implementation has now been completed, and a sample session is displayed below. It demonstrates that a portable, modular design for a NL DB query system for use by "naive" users is practical, and attainable given present technologies. This implementation operates with a standard DB management system (the

SEED system, available from International Data Base Systems, Philadelphia, Pa.), and incorporates recent innovations such as the use of the DB as an extension of the lexicon [Harris 77]. The NL procedures in the query system rely entirely on the lexicon and the DB as sources of world knowledge, and so can be transported to new domains with little or no reprogramming. In particular, the procedures required to detect the need for a cooperative response, and select an appropriate one, do not rely directly on domain specific knowledge other than that already encoded in standard ways in the lexicon and the DB system.

CO-OP QUERY SYSTEM DESIGN

The approach taken to the design of the COOPERATIVE QUERY SYSTEM (CO-OP) is to analyze questions as presenting a set from which a selection is to made by the respondent (following [Belnap 76]). The parser produces an intermediate representation, called the Meta Query Language (MQL), which is a connected graph structure. The nodes of the graph represent sets (as "presented" by the user), without regard to how those sets may be realized in the DB. The arcs represent (binary) relations defined on those sets (again, as "presented" by the user). The structure, therefore, is a non-procedural description of an N-place relation (where N is the number of sets) defined by composing the sets on all of the relations. The effect of this composition is to select the appropriate subsets of the presented sets. This N-place relation constitutes the direct response.

For example, consider the query "Which programmers in Administration work on project 6471?" (In the test DB, Administration is one of the "superdivisions".) This query is parsed as presenting 4 sets: programmers, Administration, projects, and 6471 (see FIGURE 1).



Meta Query Language (MQL) representation for "Which programmers in Administration work on project 6471?"

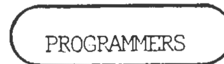
FIGURE 1

While some of these sets may appear to be counterintuitive (particularly the singleton sets "Administration" and "6471"), the intended interpretation is that these sets are presumed by the user to exist somewhere in the DB (as values, as it turns out in this case). The direct response to the query is the subset of the programmers in the DB that "survive" the composition of the relations, in the example, yielding those programmers in Administration that work on project 6471. While this is not the way the query is actually executed, it is a convenient conceptualization. The MQL expression is passed through several levels of translation, and ultimately emerges as an executable query on a CODASYL DB.

GENERATING CORRECTIVE INDIRECT RESPONSES

Should the query fail, in the sense that it returns an empty set, control is passed to a corrective indirect response generator, which attempts to check and correct any false presumptions made by the user.

Determining a large class of presumptions that the user has made is facilitated by observing that the MQL query presumes the non-emptiness of its connected subgraphs. In particular, a direct answer of "None." (the empty set) is inappropriate if the system is able to determine that a connected subgraph of the MQL also represents an empty set. Since any connected subgraph itself constitutes a well formed query, its emptiness can be checked by simply passing it through the interpretive components and executing it against the DB. Should the result be the empty set, the appropriate corrective indirect response is generated. In the example, the various subgraphs and their corresponding corrective indirect responses are as given in FIGURE 2.



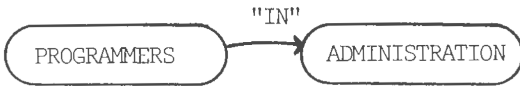
"I don't know of any programmers."

"I don't know of any Administration."

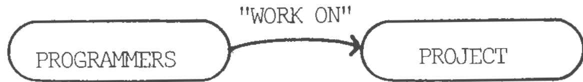
"I don't know of any Projects."

6471

"I don't know of any 6471."

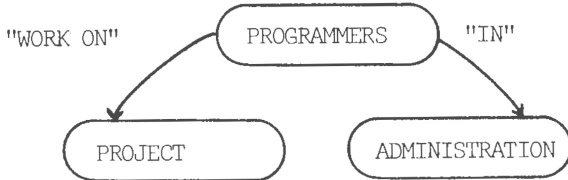


"I don't know of any programmers in Administration."

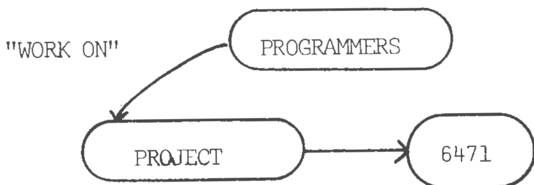


"I don't know of any programmers that work on projects."

"I don't know of any project 6471."



"I don't know of any programmers in Administration that work on projects."



"I don't know of any programmers that work on project 6471."

MQL subgraphs and corresponding corrective indirect responses.

FIGURE 2

Suppose that the query has been posed in an environment where there is no project 6471 in the DB. While the direct, correct response to the query is "None.", this response misleads the user by implicitly confirming that there is a project 6471. Rather than presenting the direct response to the user, the control structure begins executing the presumptions (subgraphs) against the DB. It will discover that the subgraph corresponding to "I don't know of any project 6471." returns an empty response set, and consequently will produce this corrective indirect response, rather than the direct one. All corrective responses generated by this technique will entail the direct response to the query, since they will entail the emptiness of the direct response set.

Several aspects of this procedure are worthy of note. First, although the selection of the response is dependent on knowledge of the domain (as encoded in a very general sense in the DB system - not as separate theorems, structures, or programs), the computation of the presumptions is totally independent of domain specific knowledge. Because these inferences are driven solely by the parser output (MQL representation), the procedures that determine the presumptions (by computing subgraphs) require no knowledge of the DB. Consequently, producing corrective indirect responses from another DB, or even another DB system, requires no changes to the inferencing procedures. Secondly, the mechanism for selecting the indirect response is identical to the procedure for executing a query. No additional computational machinery need be invoked to select the appropriate indirect response. Thirdly, the computational overhead involved in checking and correcting the users presumptions is not incurred unless it has been determined that an indirect response may be required. Should the query succeed initially, no penalty in execution time will be paid for the ability to produce the indirect responses. In addition, the only increase in space overhead is a small control program to produce the appropriate subgraphs (the linguistic generation of the indirect response is essentially free - it is a trivial addition to the paraphrase component already required in the parsing phase). For these reasons, corrective indirect responses, made possible by a careful choice of representations and associated algorithms, are produced in a domain transparent fashion with minimal system overhead using knowledge already available in the DB.

SOME OTHER COOPERATIVE RESPONSES

In addition to facilitating corrective indirect responses, the MQL provides a convenient representation for producing other types of cooperative responses.

One such response is a Suggestive Indirect Response. In human conversation, questions are normally phrased to expect a positive or non-trivial answer. When negative responses occur, it is frequently a signal to the respondent that the questioner has gotten "off the track". It is then appropriate to include some additional, potentially relevant information in the response. 5B and 6B are examples of such responses.

5A. Is John a senior?
5B. No, he's a junior.

6A. Are there any more trains to N.Y. this evening?
6B. No, but there are 3 buses.

We call such responses suggestive indirect responses, because after answering the questions, they go on to suggest some additional information.

The key observation here is that these responses are usually answers to slightly different questions. This system incorporates a mechanism for producing suitable variants of queries under appropriate circumstances. In the proper environment, the query in FIGURE 1 would result in the response: "None, but here are the programmers that work on project 6471..." As with corrective responses, the suggestive indirect response procedures mainipulate the MQL in a domain transparent way.

A common problem in NL query systems is the production of unacceptably terse answers. Consider a response of 7B to 7A, and 8B to 8A.

7A. What grades did students get in CSE110?
7B. A,B,B,A,C,...

8A. What are the phone numbers of managers in Marketing?
8B. 293-4958, 584-7945, 293-7754...

Obviously, both the grades and the phone numbers are likely to be useless without the associated names. These questions require Supportive Indirect Responses - responses that provide the supporting information necessary to interpret the answer.

The production of supportive indirect responses is facilitated by the MQL. The present system provides information on each presented set (except those that are singletons), supporting the selection made from the primary presented set.

The CO-OP system currently provides other types of indirect and direct cooperative responses, in addition to these.

A SHORT DEMONSTRATION

APPENDIX A contains an example interaction with the preliminary implementation the CO-OP system as it now stands. This session is abstracted and commented here.

CO-OP is currently implemented on a DEC KI-10 under TOPS-10 mainly in UCI-RUTGERS-LISP, and runs in 95K words, including the SEED DB system. The DB is a live one used by the National Center for Atmospheric Research (NCAR) in Boulder, Co. to keep track of their computer users, projects, research sponsors, etc. The primary purpose of the DB is to track the use of NCAR facilities, for accounting and other uses.

The user in the example is a hypothetical Computer Science department chairperson who has heard that the NCAR DB contains information about who is sponsoring research in various areas, and is interested in securing some research funds. S/he believes (incorrectly, it turns out) that funds are available to study mass storage. S/he begins by asking:

Q: Who sponsors active projects in mass storage?

R: I don't know of any projects in mass storage.

The system is unaware of any projects in mass storage, and so explicitly corrects the users presumption that there are some. The user tries a more general question.

Q: Who sponsors research in computer science?

R: I don't know of anyone that sponsors research in computer science.

Again, a corrective indirect response - but this time stating that no sponsors are known, rather than that no suitable projects are known. In addition to being able to generate corrective indirect responses, the system can choose the most appropriate one when more than one is applicable. This is done by observing that the subgraphs of an MQL query are partially ordered according to a subgraph relation, and it is most appropriate to correct the "least" failing subgraphs with respect to this ordering.

Getting nowhere, the chairperson now asks:

Q: What are the areas?

and receives a list of the areas of interest (s/he knew that the projects at NCAR are grouped according to a predetermined list of areas of interest), identified by the area number and name. S/he observes that #6 (data processing) and #12 (numerical methods) look interesting, and follows up with:

Q: Who sponsors projects in area 6?

The response is a list of sponsor names with a supportive indirect component of the projects they sponsor in area 6, the name of the area (because only the number was supplied - the system doesn't remember that it just provided the area name to the

user), and the project numbers of the sponsored projects. The user now decides that Nasa Headquarters looks the most promising (s/he has already checked with NSF), and so asks:

Q: What is sponsored in numerical methods by Nasa Headquarters?

After checking the DB, the system discovers that Nasa Headquarters doesn't sponsor anything in numerical methods. Additionally, it is unable to detect any failed presumptions on the part of the user. It therefore provides a negative response followed by a suggestive indirect response listing the projects that Nasa Headquarters sponsors in any area, in the hope that this will be helpful to the user.

R: I don't know of anything in numerical methods that Nasa Headquarters sponsors. But you might be interested in anything that Nasa Headquarters sponsors...

After perusing this list, the chairperson concludes that although the projects don't look very promising, s/he will get in touch with Nasa Headquarters. S/he asks:

Q: Who is the contact at Nasa Headquarters?

It turns out that there is a contact at Nasa Headquarters for each project sponsored, and so the system prints out the list (sorted by contact), along with the projects they sponsor. Although the user has presupposed that there is only one contact at Nasa Headquarters, the system provides the entire list, without objecting. This and other forms of sloppy reference are tolerated by the system.

CONCLUSION

This work demonstrates the feasibility of producing cooperative responses from a NL DB query system in a practical and domain transparent way. A more robust implementation is currently underway, in the hope that this system can be put into active use at NCAR. CO-OP, as designed and currently implemented, produces other types of responses not detailed here. Approaches to portablility, transparency of DB update, sloppy reference, modularity, a new method of parsing and treating parse failure, and sensitivity to a users "view" of the domain as reflected in their questions, are all incorporated to some degree in this system.

Any practical NL system that will be subjected to typically naive users must address the issues of cooperation addressed here, if it is to function acceptably. A careful choice of representations and associated algorithms can produce an acceptable level of cooperative behavior without encoding large chunks of domain-specific knowledge or maintaining a detailed user model.

REFERENCES

Austin, J.L., *How To Do Things With Words*, J.O. Urmson, Ed., Oxford University Press, N.Y. 1965.

Belnap, N. D., and T. B. Steel, *The Logic of Questions and Answers*, Yale University Press, New Haven, Conn., 1976.

Gerritsen, Rob, *SEED Reference Manual*, Version C00 - B04 draft, International Data Base Systems, Inc., Philadelphia, Pa., 19104, 1978.

Grice, H. P., "Logic and Conversation", in *Syntax and Semantics: Speech Acts*, Vol. 3, (P. Cole and J. L. Morgan, Ed.), Academic Press, N.Y., 1975.

Harris, L. R., "Natural Language Data Base Query: Using the Data Base Itself as the Definition of World Knowledge and as an Extension of the Dictionary", Technical Report #TR 77-2, Mathematics Dept., Dartmouth College, Hanover, N.H., 1977.

Joshi, A. K., S. J. Kaplan, and R. M. Lee, "Approximate Responses from a Data Base Query System: An Application of Inferencing in Natural Language", in Proceedings of the 5th IJCAI, Vol. 1, 1977.

Kaplan, S. Jerrold, "Cooperative Responses from a Natural Language Data Base Query System: Preliminary Report", Technical Report, Dept. of Computer and Information Science, Moore School, University of Pennsylvania, Philadelphia, Pa., 1977.

Keenan, E. L., and Hull, R. D., "The Logical Presuppositions of Questions and Answers", in *Prasuppositionen in Philosophie und Linguistik*, (Petofi and Frank, Ed.), Athenaum Verlag, Frankfurt, 1973.

Lee, Ronald M. "Informative Failure in Database Queries", Working Paper #77-11-05, Dept. of Decision Sciences, Wharton School, University of Pennsylvania, 1977.

APPENDIX A

Following is a transcript of a brief sample session of the CO-OP system. The user is a hypothetical Computer Science Department chairperson who is looking for research funds (see text).

Some notes: In order to maintain the integrity of the system in an environment where the DB is subject to updates, lexical items appearing solely as data values are not duplicated in the NL components. Although the design of the system calls for the DB to operate as an extension of the lexicon, this link has not yet been implemented. Consequently, data values encountered in the questions are currently presented to the user with a menu of possible categories for disambiguation. This inconvenience will be eliminated in the new version, currently under development.

Paraphrases are presented to the user for approval to avoid misinterpretations. Unfortunately, the paraphrases are much too close (usually identical) to the user's phrasing. The new version of the system will incorporate a more reasonable theory of paraphrase. The data presented here has been altered in the interest of privacy. Other than these changes, the transcript is an exact copy of the output produced by the system.

```
(CO-OP)


Welcome to the COOPERATIVE QUERY system, Version 0. This system
provides the casual user with informative responses to Natural
Language queries. It will perform only Data Retrieval - no calculations
or corrections to the Data Base can be made. Currently, the following
conventions/restrictions must be observed:

    1. Each question must begin with a WH-word, or 'How...'.
    2. Do not refer to dates or clock times.
    3. Explicit conjunction, disjunction, and negation are not supported
       ('and', 'or', 'not').
    4. Neither are quantifiers (except as in 1. above), or comparatives.
    5. No smoking while the query system is in operation.

What is the maximum # of entries you want in the responses?  15

Please state your question:

 WHO SPONSORS ACTIVE PROJECTS IN MASS STORAGE?

PARSING...

I'm not quite sure what "MASS STORAGE" means...

Please enter the number of the category the word belongs in.

0         SOMEWHAT UNCERTAIN
1         NONE OF THE BELOW
2         DIVISION NAME
3         SUPERDIVISION NAME
4         SPONSOR NAME
5         ADVISOR NAME
6         PERSON NAME
7         AREA OF INTEREST
8         PROJECT TITLE
9         GROUP ID

YOUR SELECTION:  7


I understand your question to mean:

WHO SPONSORS PROJECTS ACTIVE IN MASS STORAGE ?

Is that approximately what you intended (yes, no or stop)?  YES - SORT OF.


I DON'T KNOW OF ANY PROJECTS IN MASS STORAGE .


Any more questions (yes or no)?  YES
```

Please state your question:

WHO SPONSORS RESEARCH IN COMPUTER SCIENCE?

PARSING...

I understand your question to mean:

WHO SPONSORS RESEARCH IN COMPUTER SCIENCE ?

Is that approximately what you intended (yes, no or stop)? YES

I DON'T KNOW OF ANYONE THAT SPONSORS RESEARCH IN COMPUTER SCIENCE .

Any more questions (yes or no)? YES

Please state your question:

WHAT ARE THE AREAS?

PARSING...

I understand your question to mean:

DISPLAY THE AREAS .

Is that approximately what you intended (yes, no or stop)? YES

(AREA OF INTEREST #)	(AREA OF INTEREST)
1	AEROSOL PHYSICS
2	ASTROPHYSICS
3	BOUNDARY LAYER, PLANETARY
4	CHEMISTRY, ATMOSPHERIC
5	CLIMATE
6	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE
7	DYNAMIC METEOROLOGY (GENERAL)
8	ECONOMIC AND SOCIETAL IMPACT STUDIES
9	GENERAL CIRCULATION
10	GEOPHYSICAL FLUID DYNAMICS (BASIC)
11	MESOSCALE AND REGIONAL-SCALE MODELS
12	NUMERICAL METHODS
13	OBJECTIVE ANALYSIS AND DIAGNOSTIC STUDIE
14	OCEANOGRAPHY
15	RADAR METEOROLOGY

Any more questions (yes or no)? YES

Please state your question:

WHO SPONSORS PROJECTS IN AREA 6?

PARSING...

I understand your question to mean:

WHO SPONSORS PROJECTS IN AREA #6 ?

Is that approximately what you intended (yes, no or stop)? YES

(SPONSOR NAME)	(PROJECT TITLE)	(AREA OF INTEREST)	(PROJECT)
AIR FORCE CAMBRIDGE RESEARCH	AIRCRAFT + RADAR STUDIES OF PRECIP. SYSTEMS	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35371001
AMERICAN METEOR. SOCIETY	AMS QUESTIONNAIRE ANALYSIS	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	36051000
BUREAU OF RECLAMATION	SAN JUAN AIRCRAFT DATA ANALYSIS	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35711008
ENVIRONMENTAL PROTECTION AGNCY	HEAT + VAPOR FLUX MEASRMNTS FROM NCAR Q AIR	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35711009
ERDA-BATTELLE NORTHWEST LABS	WYOMING WIND ENERGY	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35711011
NASA HEADQUARTERS	DATA ANAL FOR COLO HAO OSO I INSTRUMENT	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35071000
NATIONAL SCIENCE FOUNDATION	AIRCRAFT + RADAR STUDIES OF PRECIP. SYSTEMS	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35371001
	GATE DATA ANALYSIS AND RELATED STUDIES	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35371005
	HEAT + VAPOR FLUX MEASRMNTS FROM NCAR Q AIR	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35711009
	KNOLLENBERG PROBE DATA ANALYSIS	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35711007
	LAB SIM. OF GLOBAL ATMOS. CIRCULATION	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35111021
	METROMEX AIRCRAFT DATA REDUCTION	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35061000
	S.PARK CONTINENTL SUMMRTME CUMULS EXPT.	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35081018
	SNOW SHOWER CLOUD PHYSICS-DATA REDUCTION	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35061015
	STATISTICAL ASPECTS OF METEOR.DATA ARCHIVING	DATA PROCESSING (AIRCRAFT, BALLOON, OTHE	35261001

Any more questions (yes or no)? YES

Please state your question:

WHAT IS SPONSORED IN NUMERICAL METHODS BY NASA HEADQUARTERS?

PARSING...

I understand your question to mean:

WHAT IN NUMERICAL METHODS DOES NASA HEADQUARTERS SPONSOR ?

Is that approximately what you intended (yes, no or stop)? YES

I DON'T KNOW OF ANYTHING IN NUMERICAL METHODS THAT NASA HEADQUARTERS SPONSORS .
BUT YOU MIGHT BE INTERESTED IN ANYTHING THAT NASA HEADQUARTERS SPONSORS ...

(PROJECT TITLE)	(SPONSOR ADDRESS)	(PROJECT/ACCT #)
ACCRETION + ELECTRICAL HEATING MODELS	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35021001
CHEMICAL MODIFICATION OF IONOSPHERE-PHASEII	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35661009
DATA ANAL FOR COLO HAO OSO I INSTRUMENT	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35071000
DYNAMICS OF PLANETARY MAGNETOSPHERES	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35101008
ELECTRONMAGNETIC SCATTERING	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35021006
MARKOV CHAIN RADIATIVE TRANS. FOR CALCULATIN	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	36081000
OUTER ATMOSPHERES OF F,G,+K DWARF STARS	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35071048
PERTURBATIONS OF STRATOSPHERIC OZONE	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35121013
PLANETARY HEATING MODELS - SOLAR WIND	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35021007
RADIATIVE TRANSFER IN SPHERICAL SHELL ATMOS.	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35311027
STRATOSPHERIC PHOTOCHEMISTRY + TRANSPORT	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35201010
THEOR MODEL FOR DETER OF CHRGD PARTICLE DIST	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35201017
TRANS. NON-EQUILIB. OF ELECTRONS IN SOLAR RE	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35071040
WAVE INDUCED PRECIP.OF ENERGETIC PARTICLES	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35661007
X-RAY SCATTERING IN SOLAR FLARES	PLANETARY PROGRAMS/SL, WASHINGTON, DC 20546	35661005

Any more questions (yes or no)? YES

Please state your question:

WHO IS THE CONTACT AT NASA HEADQUARTERS?

PARSING...

I understand your question to mean:

DISPLAY THE CONTACT AT NASA HEADQUARTERS .

Is that approximately what you intended (yes, no or stop)? YES

(SPONSOR CONTACT)	(PROJECT/ACCT #)
Benson, Bernard	35071040
Candler, L.M.	35101008
	35121013
Farell, Simon V.	35661009
Handler, J.	35021006
Kenig, Lana P.	35201017
	35021001
	35311027
King, Jr, James	35661005
Marshall, B.	35201010
Myers, David	35201012
Noble, Paul H.	35661007
	35021007
Schrager, A. L.	35071048
	35071000

Any more questions (yes or no)? NO

A Progress Report on the Discourse and Reference Components of PAL

Candace Sidner
M.I.T. Artificial Intelligence Laboratory
Cambridge, MA U.S.A.

Abstract: This paper reports on research being conducted on a computer assistant, called PAL. PAL is being designed to arrange various kinds of events with concern for the who, what, when, where and why of that event. The goal for PAL is to permit a speaker to interact with it in English and to use extended discourse to state the speaker's requirements. The portion of the language system discussed in this report disambiguates references from discourse and interprets the purpose of sentences of the discourse. PAL uses the focus of discourse to direct its attention to a portion of the discourse and to the database to which the discourse refers. The focus makes it possible to disambiguate references with minimal search. Focus and a frames representation of the discourse make it possible to interpret discourse purposes. The focus and representation of the discourse are explained, and the computational components of PAL which implement reference disambiguation and discourse interpretation are presented in detail.

Keywords: reference disambiguation, discourse interpretation, discourse purposes, natural language, focus, frames.

1. Introduction

Every discourse in English consists of one or more sentences which create a general context of people, places, objects, times and actions. The speaker of the discourse generally will not relate references from one sentence to the previous in any direct fashion nor indicate how the requests or assertions of each sentence in the discourse are connected. For the hearer to interpret the speaker's discourse and decide what the speaker is requesting or asserting, the hearer must complete two tasks, among others: (1) disambiguate the referential terms for their inter-sentential and extra-sentential links, and (2) determine the purpose of each sentence in the discourse. The first of these two tasks makes it possible to know what entities the speaker is referring to. The second task results in establishing a connected discourse and understanding what the speaker wants to communicate. Interpreting the discourse purposes of various sentences explains why D1 is acceptable below (even though D1-2 does not mention the party) while D2 is unacceptable. A theory of reference disambiguation will explain the disambiguation of *his* to Bruce and not to Mike, in D3.

D1-1 John is having a party at his house.
2 I think the guest of honor is Mary as they are going to announce the publication of Mary's book.

D2-1 Henry wants to meet with Harold.
2 Sing a song before 3 on Thursday.

D3-1 I want to have a meeting this week.
2 $\underline{\text{Bruce}_i}$ will be the guest lecturer.
3 He will speak on slavery in ant colonies.
4 Mike wants to read $\underline{\text{his}_i}$ report before the talk.

An explanation of these phenomena underlies the research being conducted at the MIT AI lab on PAL. While PAL is designed to understand the English form of requests for arranging various events, the design depends upon a theory about how to interpret a speaker's[1] extended discourse. PAL acts as a model of a hearer in these discourse situations. Two problems that must be solved before PAL can understand requests in extended discourse are referential disambiguation and discourse purpose interpretation. This paper reports on progress on these two problems.

A sample scenario of what PAL is designed to do is given in D4 below.

D4-1 I want to schedule a meeting with Dave.
2 It should be at 3 p.m. on Thursday.
3 We can meet in his office.
4 Invite Bruce.

To understand this discourse, PAL must have several natural language skills:

a. parsing for the syntactic structure.

---

1. I will use the term *speaker* to refer to the producer of a spoken or written discourse and *hearer* to refer to the receiver of the discourse.

b. interpretation of predicate-argument relations.
c. mapping of the words of each sentence to a representation used by the underlying database and programs.
d. disambiguation of the referential terms.
e. interpretation of each sentence for its discourse purpose.

The first two of these skills constitute the parser and case frame interpreter developed by Mitch Marcus. The representation mapping was developed by the author. These three modules are discussed in Marcus [1978]. To present a clearer picture of what PAL must be able to do, consider a sentence by sentence interpretation of the above dialogue.

I want to schedule a meeting with Dave.

PAL interprets an internal representation of the speaker as referent of "I," and an internal representation of "David McDonald" as the referent of "Dave."

PAL creates a new internal representation with features to be discussed later to be the referent of "a meeting."

PAL interprets "want to schedule a meeting" to be a request for a scheduling operation which may extend over several sentences.

PAL interprets the whole sentence to be asserting that the meeting has two participants, the speaker and Dave McDonald.

It should be at 3 p.m. on Thursday.

PAL interprets "it" as co-referring to the meeting under discussion.

PAL disambiguates the time phrase to a frame form used by the scheduler.

PAL interprets the sentence as asserting additional information about the meeting at hand.

We can meet in his office.

PAL determines that the speaker and other participant are the co-referent of "we."

PAL finds in its internal representations of things, an entity which "his office" can refer to.

PAL accepts the sentence as providing more information about the meeting at hand and asserts that fact.

Invite Bruce.

PAL finds an internal representation of the person referred to as "Bruce."

PAL determines that the ellided event which Bruce is to attend is the meeting under discussion.

PAL accepts the invite command as asserting another participant of the meeting.

<end of discourse>

PAL interprets the scheduling reuqest as complete and carries out the scheduling command with the meeting as it has been specified in the discourse.

In order to perform these tasks, a theory about the nature of discourse and some of its components has been developed and will be reported on here. Following that discussion, a closer look at the rules used by an implemented running version of PAL will be discussed.

2. Definition of Discourse

First, a "discourse" must be defined. I take a discourse to be any connected piece of text or spoken language of more than one sentence or independent sentence fragment. Ideally, every discourse is about some central concept which is then elaborated by the clauses of a discourse. Speakers often produce discourses which fail to meet this specification because they talk 1a) about several concepts without relating them or 1b) without informing the hearer that several concepts will be discussed at once or 2) because there is no central concept in their discourses. However, this idealization will serve to introduce some important terms. Multi-concept discourses do occur, and can be described using an approach which is a generalized version of that presented in this paper. Some cases of multi-concept discourse are discussed in Bullwinkle [1977]. However, the theory presented here has been tested in a running implementation of PAL, and this paper is restricted to that tested model.

In previous work [Winograd, 1971; Rieger, 1973; Charniak, 1972] various structures for referencing were assumed. Winograd used lists of entities of the same semantic type and chose referents for anaphoric terms based on recency and likelihood in the proper semantic class. His mechanism was too simple and failed to account for numerous anaphoric cases as well as being limited to objects in a closed world. Rieger postulated memory structures from a conceptual dependency representation of the sentences of a discourse. The memory structures were used to infer other information that could be unified to determine co-reference. His algorithms suffer from the explosive number of inferences that can be made from each memory structure. Charniak supposed that there were large collections of inference rules, called demons, which knew what to do with a small piece of the total knowledge, and which fired whenever that knowledge was encountered. This theory represents overkill; if one could have as many demons as Charniak supposed and get them to fire when that knowledge occurred, the mechanism could be used to predict co-referentiality of referencial terms. However, controlling the multitude of demons is difficult[2], and furthermore one cannot imagine how such a collection of knowledge is learned in the first place.

To interpret definite noun phrases and anaphors, a different approach is taken in PAL. It is assumed that discourse contains a structure, which when represented, can

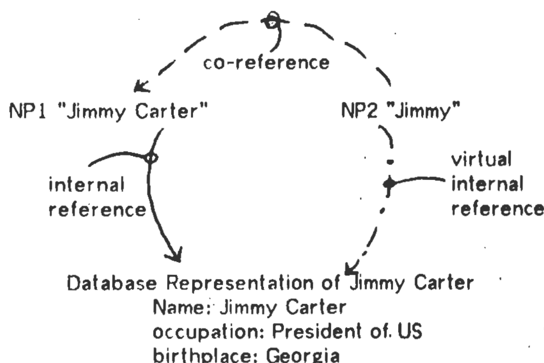
---

2. Rosenberg [personal communcation] has created a device called sentinels which may partially solve this problem.

constrain the interpretation of referential terms. From the discourse structure, rules have been discovered which govern the acceptability of referential terms in different discourse situations. The interpretation of references is not strictly deterministic; it is like knowing which of several places to look in the discourse for a co-referent and trying out the term found there.

The theory underlying PAL distinguishes two kinds of referring. The first is an internal reference between a noun phrase and some pre-existing database object. That database object represnts a real world entity. In Figure 1 below internal reference links the noun phrase NP1 "Jimmy Carter" to a representation of Jimmy Carter (who is described as president of the US, etc.). How that database object refers to the real world is the classical semantic problem of reference (cf. Kripke [1972] among others) and is beyond the scope of this work. The other kind of referring is co-reference. Co-reference links a noun phrase to another noun phrase. The two noun phrases are said to co-refer, and both refer to the same database object. In Figure 1, the dashed link from NP2 "Jimmy" to NP1 is a co-reference link. The dot-dash link from NP2 to the database object is a virtual internal reference link which results from the co-reference link from NP2 to NP1 and from the internal reference link from NP1 to the database object. Internal reference and co-reference links are distinguished because co-reference links can be established more easily using discourse structure. In the remainder of this paper when I speak of internal reference, I will drop the phrase "internal" and use only "reference."

Fig. 1. Reference Links Between Noun Phrases



3. The Concept of Focus

The central concept of a discourse may be elaborated by several sentences of the discourse and then either discontinued in favor of a related concept, or dropped in favor of a new concept. This central concept of a discourse is called the discourse focus or simply the focus. This term was first used by Grosz [1977]. A simple example of focus is *meeting* in D4 repeated below:

D4-1 I want to schedule a meeting with Dave.
2 It should be at 3 p.m. on Thursday.
3 We can meet in his office.
4 Invite Bruce.

All four sentences give information about the focussed entity. The focus is what makes a text or a set of utterances a discourse.

In this work the focus is assumed to be a concept to which other concepts are associated. Some of the association links are "built-in" in the sense that they exist previous to the discourse. For example with *meeting*, built-in association links include that a meeting has a time, a place, a set of participants, and a topic of discussion. These association links are distinguished in the sense that the concept has explicit links to these concepts while no explicit links exist to other concepts such as color, cost or age. The discourse often serves the purpose of specifying more about the concepts linked to a focus. In D4-1, there is certain information about who the participants are, while D4-2 specifies the time. D4-3 causes the hearer to infer that the office is a place for a meeting, because the focus *meeting* has a place associated with it, and because PAL expects to be informed about the concepts associated to a meeting.

In PAL the association links between concepts are easily expressed in the frames structure of FRL [Goldstein and Roberts, 1977]. A frame for a meeting has slots for times, places, participants and so on. It is exactly these slots that serve the purpose of association links to other concepts. One purpose of a discourse with PAL is to fill those slots with values and required information. As I will discuss in the section on the use of definite noun phrases, the values given to those slots are also useful in interpreting co-reference and in understanding the purpose of a sentence of the discourse.

Focus also serves as the central index point for co-referencing. The focus is what is going to be talked about in the discourse. When it is introduced, it is new information. Thereafter it is the given information, and more new information is added to it. Knowing what the focus is helps determine co-reference relations because old information can be pronominalized while new information cannot. If a focus is

seen not just as an entity by itself but connected to other entities, focus indicates how those entities can be co-referents as well. In D4-(2-4), the focus of *meeting* can be used to determine the co-reference of *it*, *we* and *his* of *his office*: *it* must co-refer to the focus, *we* to those individuals associated to the focus who include the speaker, and *his* to an individual associated to the focus who is not the speaker and has male gender. The focus is used as an access function for retrieving the co-referent of a particular noun phrase. Later in this paper, rules governing the use of anaphora by means of the focus of the discourse will be discussed.

In the current version of PAL, focus is chosen as the first noun phrase following the verb if one exists, else the subject is used as focus. This method of choosing focus is adequate for current PAL discourses but not sufficient for the most general case. See Sidner [forthcoming] for a full discussion of focus choice. Once a focus is chosen, it can be used in succeeding sentences to determine the co-reference of pronouns or definite noun phrases as well as to check to see if the discourse is still connected. A sentence like (1a) below followed by (1b) is a disconnected discourse because the co-referential terms in (1b) are unrelated to the focus of (1a) based on the association links present in the database.

(1a) I want to meet with Henry.
(1b) Give me an ice cream cone.

The focus of the discourse can be changed while maintaining a connected discourse. The chief means are end of discourse remarks and focus-shift. End of discourse remarks can be explicitly stated ones like "That's all," or implicit ones, such as the act of simply ending the input stream. A less reliable, implicit marking of the end of discourse is to use a sentence with unrelated co-referential terms. In the case above, (1a) followed by (1b) could be assumed to be two separate discourses. This case is less reliable because it is impossible to tell if the speaker assumes that the ice cream cone is related (as is often the case with a non-ideal speaker) or whether the speaker intends to change the discourse to a new one. At present PAL does not accept this kind of abrupt discourse change; instead PAL indicates that such a sentence is not intelligible in the discourse. A more sophisticated PAL might request that the speaker explain how it is that (1b) is related to the discourse.

The other means of changing the focus I call focus-shift. A discourse may expand various aspects of a focus and then choose one aspect of the focus to describe in detail. For example, in a discourse about meetings, we may want to spend several sentences specifying the time for the meeting, why that time is best and so on. When time is being discussed, one would like to know that the focus has changed so that assertions or requests can be taken to be about time. However, the meeting focus may be brought back into the discussion later. To maintain both foci, the meeting focus is stacked for later use.[3] Detecting this focus change is the process of focus-shift.

Focus shifts cannot be predicted; they are detectable only after they occur. To detect the focus shift, the focus shift mechanism takes note of new phrases in sentences following the introductory sentence. Any new phrase is a potential focus. an anaphoric term in a sentence which follows the potential focus sentence may co-refer to either the focus or the potential focus. If the potential focus is an acceptable co-referent, it is the co-referent of the anaphoric term, and the focus shifts to the potential focus. The choice of office as co-referent of *it* in D5-3 results from focus-shift. The co-referent of *it* to meeting in D5-3' results from the rejection of the potential focus office as the co-referent.

D5-1: I want to schedule a meeting with George, Jim, Steve and Mike.
2 We can meet in my office.
3 It's kind of small, but the meeting won't last very long anyway.
3' It won't take more than 20 minutes.

Rejection of a co-referent results from semantic information about the type of verb and the type of semantic entities it accepts. Semantic information has been proposed for use with co-reference (see Winograd [1971], among others). PAL uses this information only to reject or confirm choices made by the focus and focus-shift mechanisms, rather than to suggest classes of co-referents.[4]

4. Modules of PAL

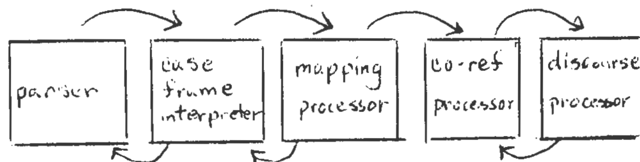
The preceding description of co-reference interpretation has been incorporated into a series of modules for PAL. These modules are depicted in Figure 2 below. The arrows represent flow of control between modules.

Each English sentence presented to PAL by a speaker is interpreted via a parser, case frame interpreter and representation mapping program [Bullwinkle, 1976; Marcus, 1978] into a set of FRL frames. The sentence "Schedule a meeting in my office," is represented by the following simplified frames (slot and slot values are listed also).

---

3. Grosz [Deutsch, 1975] gave the first specification of discourse shifts using the concept of focus. These are discussed further in Grosz [1977].
4. The mechanism of focus-shift is discussed in more detail in Bullwinkle [1977], where the term "sub-topic shift" is used.

Fig. 2. Modules of PAL



frame	=	schedule201
a-kind-of	=	schedule
type	=	"imperative"
actor	=	PAL
event	=	meeting203

frame	=	meeting203
a-kind-of	=	meeting
place	=	office207
determiner	=	"a"

frame	=	office207
a-kind-of	=	office
determiner	=	my209

frame	=	my209
a-kind-of	=	my

Given these frames, PAL is expected to determine what my209, office207, and meeting203 co-refer to. PAL also must decide what the purpose of an imperative scheduling request (represented by schedule201) is relative to its database collection of actions. Each of these modules will now be discussed in detail.

5. Interpretation of Discourse Purposes

To interpret discourse purposes, a discourse module creates a model of the discourse and controls the process of focus identification. Since the beginning, middle and end of a discourse each require different actions by the PAL scheduler, the discourse component models each differently. The first sentence of the discourse is assumed to specify what the nature of the user's communication is. This is a simplified view of the real communication process. Many discourses do not simply state their object and then elaborate the relevant information. Instead many speakers begin a discourse as in D6 below in which the first sentence contains a reason for some other action, which is requested in a later sentence. Other discourses may introduce individuals or objects to the hearer for later comment on them.

> D6: I am going on vacation the beginning of next week. John wants to see me, so schedule our regular meeting session before I leave.

The current version of PAL uses the simplified view of discourse to choose a discourse purpose. Introductory sentences are assumed to be making some sort of request. The PAL discourse module chooses which request on the basis of the verb and any associated modals, or on the basis of verbs of desire (*want, wish, would like*) and the verb complement. A request consists not only of the request type, but of some object which the request is about (intransitive verbs are not relevant to PAL since telling PAL to laugh, run or groan is inappropriate). The focus of the discourse is used for this purpose. This choice is plausible not only because the focus is closely associated with the object of the verb, but also because a discourse centers discussion on some particular entity, and that entity is captured by the focus.

Once a focus has been designated, sentences occurring in mid-discourse are assumed to be about the focus until the co-reference module predicts a focus-shift and/or until the verbs used are inconsistent with the discourse request. Mid-discourse sentences often do not explicitly co-refer to the focus as has been shown previously in D1 and D4; they may contain an implicit focus co-reference. Use of focus for co-reference disambiguation has the added benefit that sentences containing implicit focus co-references are easily recognized by the discourse component. Once an implicit focus relation is established, the module can go onto predictions of focus shift. Knowledge that the speaker is co-referring to the focus, either explicitly or implicitly, makes possible the prediction that the discourse is not yet complete, and the prediction that the speaker is making a coherent request. Since neither prediction can be assumed trivially true, the focus is important to the communication process.

In addition to the focus, the discourse module contains knowledge allowing the module to decide if the verb of a new sentence is consistent with the discourse request. Thus in D7 below, the second sentence uses a verb that is consistent with the scheduling request while in D7', the verb is odd.

> D7: Henry wants to meet with Harold. Choose a time before 3 on Thursday.
> D7': Henry wants to meet with Harold. Sing a song before 3 on Thursday.

The knowledge needed to predict consistency is represented in the frames database in two ways. First the frame for the discourse request contains information about what other requests can be sub-requests of the discourse. Second a set of mapping frames contain information which determine how a verb can be interpreted as making a certain request. For example, the verb *be* can be associated with scheduling and re-scheduling activities. However, the intention of the speaker in a sentence like (2) is different within the context of a

scheduling or a re-scheduling request.

(2) The time should be 3 pm.

In a scheduling context, (2) can be interpreted to request that the time be established as 3 pm while (2) in re-scheduling can have an interpretation of changing the time from whatever it was to 3 pm. PAL captures the intention of the speaker relative to a request context by an inference mechanism which is a matcher that determines that (2) represented as a frame[5] can be associated with scheduling requests by a simple mapping between two frames. This correspondence coupled with the use of focus makes it possible to understand (2) as part of a discourse.

In addition, the mapping functions tell how to interpret the current sentence into one of the commands which the scheduler can perform. Included in this process are how to map the slots of one frame into a frame which is the scheduling action. For example, the verb frame for "We can meet in 823" is mapped from a "meet" frame into a frame called "assert" with a slot for the object asserted, which is the focus, and a slot for what is asserted about that object, in this case the place as 823.

The end of a discourse is currently interpreted as being the end of the speaker's input stream. A more sophisticated means of interpreting discourse end is possible, though not implemented, given the focus mechanism: when the needed slots of the focus are filled, the speaker can be considered to have finished this discourse. Upon sensing the end of the discourse, the discourse module informs the scheduler that it can carry out the action requested at the discourse beginning. At first glance this may appear as if the discourse request specified at the beginning is ignored in favor of other requests. In fact the initial request is used in interpreting mid-discourse sentences. However, many discourse actions like scheduling require that the action of scheduling be delayed until all the necessary information for scheduling is presented. This process normally cannot be stated in a single sentence, and a whole discourse is needed to fill in the request. In this fashion the discourse module reflects the fact that a discourse consists of many sub-discourses centered around individual entities and which are opened and closed by focus shifting or finishing discussion of the current focus.

PAL is similar to the GUS system [Bobrow et al, 1977] because it expects a discourse to provide information about the slots of a frame. GUS permits user initiative although it is unclear what the extent of this initiative is. GUS does not seem to allow for user initiative of the discourse requests. Since PAL expects full user control over all parts of the discourse, PAL needs a complete description of the discourse and its focus. PAL's use of focus also presents a complete theory of the kinds of co-reference problems raised by the GUS system.

6. Co-reference Disambiguation

There are two sub-modules for co-reference interpretation in PAL, the sentential and inter-sentential co-reference modules. The inter-sentential co-reference sub-module chooses co-references for referential terms in the discourse once the focus is identifed. The task of determining co-reference varies depending upon the presence or absence of previous discourse. When there is previous discourse, co-reference interpretation depends largely on the focus. For simple[6] definite noun phrases, PAL assumes either the focus is the direct co-referent of the definite noun phrase or the focus contains a slot that is the co-reference of the definite noun phrase. This assumption needs modification since some definite noun phrases are used to refer outside the context of the discourse. For example, when trying to schedule a meeting, if the speaker says (3), the definite noun phrase co-refers to an entity associated with the meeting under discussion; that association is reflected in the frame slot structure of FRL.

(3) The best place is my office.

However, if the speaker says (4), *the conference room*, i.e. that particular conference room which the speaker has in mind, is not associated with meetings in general, and so the focus does not point out the co-reference.

(4) We ought to meet in the conference room.

However, by searching the focus, the lack of a connection can be noticed, and a reference from the database can then be considered. In this way, the focus acts as an access function, but only for those co-referential terms related to the previous sentences of the discourse.

PAL uses database search with growing contexts of reference to choose reference for other kinds of noun phrases which refer to entities outside the discourse. Growing a context is accomplished using the immediate set of frames from the first sentence and recursively creating larger sets from the slot values of those frames until the frame with the name in question is found. The context growing mechanism reduces

5. A frame is not taken as the meaning, in the classical semantic sense, for (2); PAL makes no claims about this sense of meaning.

6. A simple definite noun phrase is a definite noun phrase containing no relative clauses. At present PAL interprets only such noun phrases.

search from a more global search strategy, and helps control potential ambiguities that exist due to multiple possible references in the database. This same method could be used for definite noun phrases that refer outside the discourse.

Use of the focus is actually somewhat more complex since the definite noun phrase may be a co-reference to the potential focus of the discourse. Should a definite noun phrase co-refer to the potential focus, the discourse module pushes the current focus to a focus stack and takes the potential focus as the new focus. The pushed focus is available for later use in the discourse. The current inter-sentential sub-module does not interpret definite noun phrases used generically. The focus can be used for these cases as well (see Sidner, [forthcoming]), but the details of this process are not included in the current version of PAL.

The inter-sentential co-reference sub-module also determines the co-reference of personal pronouns. For the pronouns of first person plural (*we, us*), two choices can be made. First the sub-module can choose the focus as the direct co-referent of the anaphor. Second the sub-module can choose a set of co-references from a particular slot of the focus. That slot must contain co-references including the speaker of the discourse. For *he/she*, and its object forms, the focus is chosen as a direct co-reference. Using the focus as co-referent explains the anaphoric co-reference in D8 of *his* to Bruce and rather than Mike. When the focus is not the co-referent, a co-referent stipulated by the co-reference rules of the sentential co-reference sub-module, discussed below is used. Finally if neither is acceptable, entities associated with the focus are checked for co-reference. This sub-module predicts misuse of *he/she* pronouns if no co-references are found from this process or if more than one results from the last step in the process.

The interpretation of co-reference for *he/she* pronouns needs to be expanded to include consideration of potential focus since in D8 below, *his* co-refers to Bruce and not to Mike.

> D8: I want to have a meeting this week. $\underline{\text{Bruce}_i}$ will be the guest lecturer. Mike wants to read $\underline{\text{his}_i}$ report first.

It appears that the focus and potential focus ought to be checked for co-reference to such pronouns before sentential co-reference rules are used. However, further experimentation with such cases is needed to confirm this aspect of co-reference.

For the co-reference of *it*, the inter-sentential co-reference sub-module chooses a co-referent either from the focus, the potential focus or from predictions from sentential co-reference rules, which are discussed below. This choice strategy is not entirely adequate because recency appears to play a role in the co-reference choices for *it*. Recency rules are discussed in Sidner [forthcoming], and could be included in a future version of PAL. The inter-sentential co-reference sub-module uses the semantic constraints placed on the pronoun by the verb in a few instances; this portion of PAL could be expanded greatly. Co-reference rules for *they* work similarly to those for *it* with consideration that the speaker cannot be included in the co-reference set.

When no previous discourse exists, PAL's sentential co-reference sub-module uses the co-reference rules of Lasnik [1976] to choose co-references. The rule is stated as follows: If a noun phrase, NP_1, precedes another noun phrase, NP_2, and NP_2 is not a pronoun, and further if the minimal cyclic node dominating NP_1 also dominates NP_2, then NP_2 and NP_2 are disjoint in reference. The expression "disjoint in reference" is taken to mean have no references in common, thereby blocking the co-reference of *Bob* and *Tom* to *they* in (5).

(5) They assume that Bob will talk to Tom.

By using Lasnik's rule, disjoint references of a noun phrase in a sentence can be chosen, as well as a list of acceptable co-references for the noun phrase. This information is recorded in the frame presenting the noun phrase. As pointed out by Reinhart [1976], Lasnik's rule fails to predict the disjoint references in sentences like (6) and (7) below, but these cases are not problematic given inter-sentential co-reference rules because other rules will predict the co-reference for the pronouns first.

(6) Near $\underline{\text{Dan}}$, $\underline{\text{he}}$ saw a snake.
(7) For $\underline{\text{Ben's}}$ wife, $\underline{\text{he}}$ would give his life.

In addition to the use of a co-reference rule, the sentential sub-module determines the referents of proper names. Using the collection of frames which make up the discourse, a frame containing the correct first (and if given, last) name can be found. Should the immediate discourse fail to produce the name referent, a larger context can be grown from the slot values and from the slot defaults of the frame representing the focus. The same context growing mechanism used for definite noun phrases is used. By this process of context growing, ambiguous uses of names like *John* can be avoided. *John* will refer to that person most closely associated with the discourse. If more than one frame for the name John is found, the context growing process predicts that the speaker has used the name ambiguously. Context growing has been effective in a limited number of cases tested so far, although a database with more potential ambiguities would further test this sub-module.

7. Extensions

The current PAL can be expanded in many directions. Some of the necessary developments of its co-reference capabilities have already been discussed. Significantly, these capabilities do not require extensive new theoretical appparatus; the focus of discourse and structure of FRL can sustain the needed improvements. In discourse interpretation PAL must be extended to interpret discourses which define new people, places, events, actions and like objects as well as to interpret preferences of users and purposes for various activities. These extensions not only will make PAL a more useful system, but also they encompass a set of tasks useful for other interactive programming domains. Experimentation on the discourse module of PAL is need to incorporate these new capabilities.

8. Acknowledgements

The author wishes to thank Gretchen Brown, Ira Goldstein and Steve Rosenberg for their comments and suggestions on drafts of this paper.

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 Defense under the Office of Naval Research under Contract Number N00014-75-C-0643.

9. References

Bobrow, D., R. Kaplan, M. Kay, D. Norman, H. Thompson, T. Winograd, [1977] *GUS, A Frame-Driven Dialogue System,* Artificial Intelligence, Volume 8, Number 2, April.

Bullwinkle, C. [1976] *The Semantic Component of PAL: the Personal Assistant Language Understanding Program,* MIT AI Laboratory Working Paper, March.

Bullwinkle, C. [1977] *Levels of Compexity in Discourse for Anaphora Disambiguation and Speech Act Interpretation,* Proceedings of the Fifth International Joint Conference in Artificial Intelligence, August 1977; also M.I.T. A.I. Lab Memo 413.

Charniak, E. [1972] *Toward a Model Of Children's Story Comprehension,* M.I.T. A.I. Lab TR-266.

Deutsch, B. [1975] *Establishing Context In Task-Oriented Dialogues,* Proceedings of the 13 Annual Meeting of ACL, AJCL Microfiche 35.

Goldstein, I. P. and R. B. Roberts. [1977] *NUDGE, A Knowledge-based Scheduling Program,* M.I.T. A.I. Lab memo 405.

Grosz, Barbara [1977] *The Representation and Use of Focus in Dialogue Understanding.* Stanford Research Institute Technical Note 151, Menlo Park, California

Kripke, Saul A. [1972] *Naming and Necessity.* in Semantics of Natural Language, Davidson and Harman (eds.) Reidel Publishing Co., Boston.

Lasnik, Howard. [1976] *Remarks on Co-reference,* Linguistic Analysis, Volume 2, Number 1.

Marcus, Mitchell [1978] *Progress Report on the Parser and Semantics of PAL,* M.I.T. A.I. Lab memo forthcoming.

Reinhart, Tanya [1976] *The Syntactic Domain of Anaphora,* unpublished Ph.D. dissertation, Department of Foreign Literature and Linguistics, M.I.T.

Rieger, Charles J. [1974] *Conceptual Memory: A Theory and Computer Program for Processing the Meaning Content of Natural Language Utterances.* Stanford Artificial Intelligence Lab Memo AIM-233.

Sidner, C. [forthcoming] *A Computational Model of Co-reference Comprehension in English.* Ph.D. dissertation, M.I.T.

Winograd, Terry [1971] *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language.* M.I.T. dissertation.

Participating in Dialogues: Understanding via Plan Deduction

James F. Allen and C. Raymond Perrault

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

Abstract

This paper describes a system designed to participate in purposeful dialogues: dialogues where the participants are conversing to achieve some specific task. The system is intended to help its users attain specific goals in certain typical situations. The language behavior observed in actual dialogues includes the frequent use of sentence fragments, and sentences that cannot be interpreted literally. We have concentrated on the problems raised by these phenomena.

1. Introduction

It has become commonplace in computational linguistic circles to talk of the purposeful nature of human communication. Language is used to achieve certain effects on specific listeners. These effects typically involve modifying the hearer's goals and beliefs about the world. Furthermore, linguistic acts are often performed as steps to achieve non-linguistic acts. For example, in a train station, one may ask when a train leaves in order to be able to board the train.

Cohen's program OSCAR <Perrault and Cohen 1977, Cohen 1978> is an application of these ideas to language generation. It maintains a model of the beliefs and goals of its user and, given a goal, produces a plan to achieve that goal. This plan may involve requests to be made of the user and assertions of information which he is thought to need. The OSCAR program does no plan recognition. How it can be extended to do so is the subject of this paper.

In this framework, a major part of understanding an utterance involves discovering what particular goals the speaker is attempting to achieve. Identifying these goals provides a natural way to handle sentence fragments such as noun phrase utterances, sentences that should not be interpreted literally, and sentences whose purpose is to acknowledge, correct or otherwise clarify previous utterances. In many cases, the syntax of a sentence and the meaning of its words do not determine the speaker's intentions, the hearer must also use his knowledge of the speaker's beliefs and goals.

To provide a clear application of our work we have concentrated on purposeful (or task-oriented) dialogues. These are dialogues in which the participants are co-operating to achieve some specific goal or task.

Let us look at some examples that demonstrate a need for identifying speaker intentions. These are based on transcripts of actual dialogues collected at an information booth in a train station <Horrigan 1977>. The dialogues are between the clerk ('C') in the information booth and a passenger ('P') at the station. The passenger typically wants to meet arriving trains or leave on departing ones. The clerk always wants to help further the passenger's goals.

1.1 A Sentence Fragment

In the train station setting, as in many stereotypical situations, much communication is accomplished by sentence fragments. For example, in one case a dialogue was opened with:

P: The 3:15 train to Windsor? (1)

No syntactic methods can construct a full sentence from this fragment, nor does the meaning of the actual words indicate what is required of the clerk. To interpret this utterance, the clerk must establish what the speaker's goals are. Only by discovering how the observed utterance fits into some expected plan, can he determine what kind of answer is desired. Thus, if the passenger's goal were to board the train to Windsor, a reasonable answer is 'Gate 10'.

---

* This research was supported in part by the National Research Council of Canada.

1.2 An Indirect Speech Act

Many utterances cannot be interpreted at face value. For instance,

P: Do you know when the Windsor train leaves? (2)

An answer of 'yes' to the above is usually inappropriate. However, there are situations where such a sentence is intended literally. Only by considering the speaker's intentions, (i.e. in the form of his plans) can one decide whether a literal or indirect reading is meant. Systems that handle such utterances by rules based solely on the form of the sentence <e.g. Lehnert 1977> fail in this respect because they can only interpret utterances uniformly irrespective of their context.

1.3 A Clarification Dialogue

Many utterances in dialogue are intended to monitor the success/failure of the communication process itself. The clerk generates such utterances when needed, and recognizes when they are introduced by the passenger. A short example of a clerk initiated clarification subdialogue is:

P: When is the Windsor train? (3)
C: To Windsor?
P: Yes
C: 3:15

The initial utterance is not fully understood by the clerk. In particular, he cannot determine if the passenger wants to know something about a departing train or an arriving train. The response 'To Windsor?' is a request for a clarification of the passenger's goals. His answer indicates that he is interested in the departing train. The clerk's response '3:15' is the reply to the passenger's original question.

1.4 An Instance of Helping

In many cases, a response does not only answer the question asked, but provides additional useful information. Such responses cannot be generated unless the clerk realized what the passenger's goals are.

P: When is the train to Montreal? (4)
S: 3:15 at gate 7.

Although the information concerning the departure location was not asked for in P's query, the clerk, realizing that P wants to board the train, offers it to provide what he feels is a useful response.

1.5 The Paper

This paper deals only with the pragmatic aspects of communication: the recognition of intention. We do not mean to suggest that methods based on syntax and semantics are worthless, but we do claim that there are many situations where they are not powerful enough in isolation.

Section 2 introduces the concepts fundamental to our approach, and section 3 provides an overview of the plan deduction process. In section 4 we will reconsider the examples above. Finally we will conclude with some implications of our approach.

2. Fundamental Concepts

Central to our methodology is the concept of a plan and its associated operations: planning, plan deduction and plan execution. We will consider these informally in the next section. It is also necessary to be able to effectively represent the beliefs and goals of various agents, including the system itself. This we cover in section 2.2.

2.1 Planning Terminology

A State is a set of formulas describing some aspect of the world.

As in STRIPS <Fikes & Nilsson 1971>, the operators that change the states can be grouped into families represented by operator schemas, which can be viewed as parameterized procedure definitions. An operator schema consists of a name, a set of parameters and a set of labelled formulas in the following classes:

Preconditions: Conditions that should be true in order for the execution of the procedure to succeed.

Effects: Conditions that should be true after the execution of the procedure.

Body: A set of action names with their parameters that describe the execution of the procedure.

An Operator Instance or simply an Action is a expression constructed from the name of an operator definition with an instantiated parameter list. An action may be executed by executing the body of its definition using the instantiated parameters. Primitive actions have no bodies, their execution is specified by a procedure in the host programming language.

We will define a plan as a directed graph in which the nodes are actions and states. There are two arcs: the enable arc links a state S to an action A provided that S implies the preconditions of A are true, and the effect arc links an action A to a state S provided that S implies the effects of A are true. Given

these, an Atomic Plan is a graph consisting of one path from an action A1 to an action An using enable and effect arcs.

Observing that the bodies of actions are in fact atomic plans we add another node-type 'plan' and an arc body that links an action A to a plan P provided that P is the body of A. A Plan is then defined as an atomic plan in which some actions may be associated to their bodies by body links. (Actions in those bodies may in turn be related to their bodies, etc.)

We say a plan transforms a state S1 into a state Sn if S1 implies the preconditions of the first action in the plan, and Sn implies the effects of the last action in the plan.

A Speech Act is an action that has as parameters a speaker, a hearer, and a propositional content, and whose execution involves the production of an utterance. The preconditions and effects of speech acts are specified in terms of the beliefs and wants of the speaker and hearer.

An example of a speech act is INFORM. A precondition for INFORM is that the speaker believes the propositional content is true, an effect is that the hearer believes the speaker believes the propositional content is true.

There are various operations that manipulate plans. Given an initial state S0 and a goal state S1, planning is the process that constructs a plan that transforms S0 into S1. Given an observed action by an agent A1, plan deduction by an agent A0 is the formation of the plan that A0 believes A1 is executing. Finally, a plan may be executed by executing its constituent actions in the sequence specified by the plan.

Planning and plan deduction occur as the execution of the plan and deduce actions respectively. As a consequence, plans may include steps to deduce other agent's plans, and also steps to plan to attain new goals.

2.2 Believe and Want

This model requires that we can represent the beliefs and wants of different agents. We will not consider the representation in detail here as it has been done elsewhere <Cohen 1978>, but a few characteristics require mention.

The system ('S') must be able to maintain distinct beliefs for different agents and support arbitrary levels of nesting. (eg. S believes, S believes that P believes, S believes that P believes that S believes, ...) S must be able to represent that P knows information that S itself doesn't know. For instance, it must be able to distinguish between

- 'S believes P knows when the train arrives'
- 'S believes P believes that the train does arrive'
- 'S believes P believes the train arrives at 3:15'

The first and third beliefs imply that S believes P could answer the question 'When does the train arrive?'. The third belief also indicates that S knows what the answer would be.

The collection of beliefs of one agent is termed that agent's belief space. Wants are treated in a similar fashion to beliefs. The collection of an agent's wants, i.e. his want space, contains the current plan that he is believed to be executing.

Such a model including beliefs and wants has been implemented and used in the planning system by Cohen <Cohen 1978>. In this system the only propositional attitudes are 'believe' and 'want'; 'know' is introduced by definition.

2.3 Overview of the System

The system executes the plan in its own want space. In particular, it has a plan to help the passenger. One of the actions in this plan is to deduce the passenger's plan; another involves further planning to help him.

Let us look at the helping process in more detail. The system initially has a set of expected goals that it believes the passengers will want to achieve. Associated with these goals may be partially expanded plan fragments outlining how the goals are usually attained. These are the plan expectations.

Understanding an utterance consists of deducing the passenger's plan by seeing what expected plans could include the observed speech act. Since the system is deducing the passenger's plan, the knowledge base it will use in evaluating the plan alternatives will be what it believes the passenger believes. The constructed plan will be part of what it believes the passenger wants.

Once the plan is deduced, the system simulates its execution to find necessary steps that cannot be achieved. These are the obstacles in his plan. Helping the passenger entails overcoming these obstacles. To do this, the system makes the obstacles into its own goals and initiates planning. The execution of this new plan will usually involve conversation on the system's part.

3. The Plan Deduction Process

The plan deduction process starts with a set of 'observed' actions and a set of expectations, and attempts to construct the plan that the user is executing. In this section we describe how it receives its input, we survey the types of inferences it makes and then discuss the process itself. We conclude with a section on why we feel plan deduction is feasible.

3.1 Input

The observed actions arise from the input utterances. The plan deduction process expects to receive a hypothesis about the speech act and its propositional content, reflecting the literal meaning of the utterance. The hypothesis is based on the mood of the utterance: a declarative sentence indicates an INFORM, an imperative indicates a REQUEST and an interrogative indicates a REQUEST to INFORM.

The plan deduction process is in fact powerful enough not to require fully explicit input. The propositional content must provide a structure for the utterance, but specific detail may be omitted. For example, the hypothesis about the propositional content of 'The Montreal train?' would include an unknown predicate involving a train; the train being involved in some unknown predicate also involving Montreal. An unknown predicate should be viewed as a variable ranging over predicates. The specified arguments of the predicate constrain the range of the variable. For example, the utterance 'When is the train?' would produce an unknown predicate involving a train and a time. In the train setting, such a predicate would be ambiguous between arrival time and departure time.

Ideally, the syntactic and semantic components should provide the plan inference component a description of the restrictions that can be imposed on the missing predicates. This can be done by enumeration or by propositions stating the restrictions. Considerable work remains to be done to determine how these methods are best used.

3.2 The Inferences

Before discussing the plan deduction process itself, we would like to survey the types of inferences it will be able to make. Deduction inferences can be divided into two classes. The bottom up inferences, which start at the observed actions and try to infer a plan, and the top down inferences, which start with a plan expectation and try to expand it into more detail.

The top down inferences are basically a simulation of the speaker's planning process. For example, if the speaker's goal was X, how might he achieve X? A typical top down inference is:

"Effect-action inference"
If X is a goal state, find an act A that has effect X, and infer A as part of the speakers plan.

Bottom up inferences are inverses of the planning inferences. For example:

"Action-effect inference"
If A is an action in the speaker's plan, then infer he wants to achieve one of A's effects.

"Action-body inference"
If A is an action in the speakers plan, then infer he is performing action B, where A is part of the body of the definition of B.

"Know-action inference"
If the speaker wants to know the value of some relation R, then he wants to execute an action involving the value of R.

It must be stressed that inferences only suggest possible candidates for elaborating our plan fragments. Not all such candidates are reasonable at any one time. For instance, if an inference suggests a new goal for the speaker that is already believed true, then it is unlikely that this goal is part of the speaker's present plan. The assumption here is that if the speaker believed the underlying goals of his utterance were already achieved, he would not need to speak. This emphasizes a distinction between our system and some of the script based systems <Schank and Abelson 1975>. Although the set of expectations is script type knowledge, the plans constructed for the speaker depend heavily on the current (and changing) model of him.

3.3 The Plan Deduction Process

Plan deduction is basically a search process through many alternative plan fragments. Roughly, each alternative is a hypothesis about the user's want space and consists of the observed actions paired with one of the plan expectations. Associated with each alternative is a set of tasks that refine the alternative by adding new actions, states and relations, by binding variables, or occasionally by splitting it into new alternatives. We will refer to the plan fragments inferred from the observed actions as the observed part of the alternative, and to the rest of the alternative as the expected part.

Alternatives and their tasks are rated and the tasks compete for execution on an

agenda. The highest rated task is always executed.

A few of the factors that affect the rating of an alternative are listed below.

i) The goals in an alternative should be believed likely. This heuristic is usually applied in reverse. For instance, goals that are already believed to hold are not likely.

ii) The preconditions of actions that the agent is executing should not be contradicted. In particular, if an action A that the agent is executing is used to infer another action B via the action-body inference, the agent is considered to be executing B, hence B's preconditions should also be true.

iii) The observed and expected parts of an alternative should contain similar objects and relations. The underlying assumption is that objects are typically wanted for their normal uses (Rieger's function inference <Rieger 1974>). So if an utterance mentions a particular object, the plan expectations that could involve that object are favoured.

The violation of any of these factors reduces the rating of the alternative, but does not necessarily eliminate it from contention. In fact there will be correct plans that fail in some of these respects. If an alternative is accepted in which some preconditions are contradicted, this will lead to a discrepancy between what S believes and what S believes U believes. Such conflicts will often be resolved using further dialogue.

Typical tasks perform the following:

a) Given an action in the observed part of an alternative, apply the bottom-up inferences to the action. If there are mutually exclusive inferences possible, the alternative may be split.

b) Identify the binding of an action parameter in an alternative. The parameter may be associated with a definite description in the observed utterance, or may simply be introduced in the plan construction.

c) Terminate the plan deduction process successfully, by accepting one of the alternatives as the deduced plan. It is not necessary to expand every detail in an alternative. The alternative will be acceptable if it contains an explicit link between its observed and expected parts, and if there are no other competing alternatives that are similarily rated. In cases where two mutually exclusive alternatives have similar ratings, it may be necessary to initiate a planning subtask whose purpose is to engage the user in a clarification subdialogue.

The plan deduction process may terminate unsuccessfully in a number of ways. It may, because of difficulties in the deduction and the presence of other well rated tasks on the agenda, have its ratings reduced to the point where it is no longer competitive and 'drops off' the agenda. Alternatively, it may remain well rated because it is a crucial step in an important plan, but use up all the resources it is allowed. Finally, in extreme cases, there may be no reasonable inferences left to perform.

3.4 Why Should This Work?

There are a number of reasons why we feel this process is feasible. In this section we will discuss the structure of the domain, an abstraction capability in actions, and the assumption of co-operation.

The domain of conversation in our system is well specified, i.e. the possible topics are those defined by the plans we can construct. Once a plan is established as the one being executed, it defines a notion of coherence on the dialogue; dependencies between utterances are reflected by their relation to the plan. The plan also provides strong predictive power as to what to expect next. Although the range of topics in the train domain is limited, the variety of conversation, in intention and form, is considerable. Such domains provide ideal testbeds for a wide range of linguistic problems.

Another factor reducing the combinatorics is that the bodies of actions provide us with an abstraction capability. To elaborate, let us consider an action A. Its body can be viewed as specifying its execution requirements to a level of greater detail. The sequence of actions in the body may introduce new precondition requirements on the execution of A. This is similar to the abstraction capabilities in NOAH <Sacerdoti 1975>. We may create a plan involving A without initially considering the 'details' required in its body. Once we have a plan involving an action at an 'abstract' level, we may expand to a level of greater detail by considering its body. Initially, in the deduction task, we are interested in finding some link between an observed action and one of a set of abstract expectations. It is not necessary to discover the fully expanded plan of the speaker, a small segment of his plan, providing an intersection, is sufficient. When we infer bottom up from one action to another via its body, we may omit considering much detail. As a result, our deduced plans normally include many levels of abstraction, but are not very broad at any one level. Once an intersection is found, and competing alternatives are eliminated, we then can

afford to expand out details of the deduced plan as necessary.

Most important is our primary assumption that the participants are co-operating with each other. In particular, we expect the speaker will form his utterances in such a way that there is little possibility of misunderstanding him. For example, if the speaker believes we have strong expectations about what he will say, he may communicate with minimal indication of his intention. This occurs when the speaker is answering a question, or when the situation restricts the possibilities, as in the train station. If the speaker believes we have little or no expectations, he will tend to mention his goals more explicitly.

As a consequence of this we note that we are not attempting arbitrary Plan Recognition, for the actions we observe are intended to facilitate recognition.

This supports two powerful heuristics:

i) The similarity of objects and relations between the expected and observed plans provides a strong indication of which alternatives are promising. If the objects or relations were intended to be used in a non-expected way, the speaker would be obliged to explicitly mention that intention.

ii) The inferences that are obvious (i.e. among other things, cause little fanout) are preferred. So inferences that are expensive to make will tend to be neglected.

Finally, we should observe that we are not committed to succeeding at all costs, for plan deduction failure does not imply dialogue failure. More important than being able to succeed every time is the ability to recover and continue the dialogue in a coherent manner, say by asking for clarification.

4. A Brief Look at Some Examples

This section reconsiders the problems of section 1. It is convenient to look at the last example first.

4.1. An Example of Helping

This example provides no difficulty to the plan deduction process, but demonstrates how the entire system operates to provide a response.

P: When is the train to Montreal?
S: 3:15 at gate 7.*

P's utterance is viewed as a REQUEST that S INFORM P of the departure time of the train to Montreal. S deduces that P wants the effects of the action, i.e. that P wants to know the departure time, and from this that P wants to be able to board the train to Montreal. Once P's plan is deduced S accepts P's goals as its own. In particular S now wants P to board the train and S wants to inform P of the departure time. However, inspecting the plan, S also finds that P needs to know the departure location. If S believes P does not presently know the location it may be helpful by providing this extra information. So the system's response is based on what it believes the obstacles in P's plan are.

In the following examples we deal only with the plan deduction process.

4.2. A Sentence Fragment

This example is a simplified simulation of the plan deduction process in execution. We consider only the best rated plan alternative, thus making the example a reasonable size. We also ignore ratings and assume the agenda is a FIFO list.

P: The 3:15 train to Windsor?

The observed action is a REQUEST by the passenger that the system INFORM him of some unspecified predicate involving a train. Furthermore, the train is associated with the time 3:15 and is related to the city Windsor in a manner consistent with the preposition 'to'. The plan deduction task produces the three following subtasks to initiate processing.

The Agenda:
i) Identify the formal object representing the train.
ii) Review the expectations and the utterance for similarities.
iii) Apply inferences from the observed action.

---

* note: the system does not generate English output, it specifies the content of the response and generates output in a rigid pre-defined form.

Identify:

The identification of objects requires specialized knowledge. For this example we must know that trains have associated arrival times, departure times, sources and destinations and that destinations are often flagged with the preposition 'to'.

The train is recognized to have a destination Windsor, and to have either an arrival or a departure time of 3:15. Its source is not known at this point. This description does not provide sufficient information to pinpoint the referent in the data base. The subtask completes without identifying the referent.

Review:

The expectations that involve trains are 'passenger travel from Toronto to x' and 'passenger meet train at Toronto from x', these are selected as possible alternatives. But only the former is compatible with the train's destination of Windsor; the train in the 'meet' alternative must have destination Toronto. In the 'travel' alternative, we assume that the trains in the observed and expected parts are the same. Merging the descriptions together we infer that the train has source Toronto and destination Windsor. This alternative also involves the departure time of the train which is consistent with the specified time 3:15. Two new subtasks are created: an 'identify train' task, since more is now known about the train, and an 'expand the travel expectation into more detail' task, which would perform further inferencing.

The Agenda:

i) Infer from the observed action.
ii) Identify train in travel expectation.
iii) Expand the travel expectation into more detail.

Infer:

The observed action is 'passenger REQUEST that the system INFORM him of something concerning a train'. The most promising inference from this action is that it is part of the body of a QUERY action to obtain information. This action is added to the alternative, and an infer task is created involving it.

Identify train:

The train is specified sufficiently by its source, destination and departure time to be associated with a particular train in the data base. The train is successfully identified.

Expand the travel expectation:

One of the steps in the travel plan is the action of boarding the train. To do this, the agent must know its departure time and location. The system believes the passenger knows the departure time (3:15) but not the departure location. Simulating the passenger's planning, it produces the action 'passenger QUERY someone about the departure location' as a way of achieving this knowledge. However, this action matches (i.e. is compatible with) the QUERY action in the observed part. They are merged to form the action 'passenger QUERY system about the departure location of the train'. Since there is now an explicit link between the expected and observed parts, a high priority task is created to accept this alternative.

Accept:

The alternative is accepted because there are no well-rated competitors. This task terminates the plan deduction process successfully. The system now easily finds an obstacle in the plan: the passenger needs to know the departure location. The obstacle can be overcome simply by a system INFORM.

In the next examples we will only point out new and important details. The last example will look at a case where there are well-rated competiting alternatives that cannot easily be eliminated.

4.3. An Indirect Speech Act

P: Do you know when the Windsor train arrives?

This utterance has two different interpretations. The literal interpretation indicates that the passenger wants to know whether the system knows the arrival time. The indirect interpretation indicates that the passenger wants to know the arrival time himself.

The input to the plan deduction process will correspond to the literal reading. The inference path to the 'meet train' expectation is as follows. The effect of the literal reading is that the passenger knows whether the system knows the arrival time. The system knowing the arrival time is a precondition to the action that the system INFORM the passenger of the time, which has the effect that the passenger knows the time, which is a necessary step in the 'meet train' plan. So the plan can be deduced, using either top-down or bottom-up inferences.

Underlying this chain of inferences is our assumption that both S and P know that S is helping, i.e. that S is continually

trying to infer P's plans, locate obstacles and overcome them. S knows that P expects this helpful behaviour and so can assume that P intended the result of any plan inference, obstacle detection or plan construction that S can make on the basis of knowledge that S believes P believes.

The question remains of how far S is to assume P intends these inferences to be pressed. One case at least is clear: whenever S believes P believes that the effects of the literal utterance are already true, S's inference process must continue. This is the case with this example. If the effects of the literal utterance are not believed to be true, S could still choose the indirect goal but this may lead S to ascribe extraneous intentions to P.

From an implementation point of view it is undesirable that the system have to work through the inferences for "standard" indirect forms such as "Do you know ..." and "Can you tell me ...". The indirect forms here should be considered immediately at the start of inferencing (cf. Brown 1978).

4.4 A Clarification Dialogue

P: When is the Windsor train?
S: To Windsor?
P: Yes
S: 3:15

The passenger's first utterance is processed similarily to the first example, however, both the 'travel to x' and 'meet train arriving from x' expectations remain possible. Let us assume there is no contextual reason to favour one alternative over the other. The goal of the plan deduction process is that the system knows the passenger's plan. In this case it is not achievable on the basis of the utterance supplied. A subtask of the plan deduction task recognizes this and initiates a subtask to plan to achieve the goal (system BELIEVE passenger WANTS 'the travel plan') OR (system BELIEVES passenger WANTS 'the meet plan').

The planning for this goal produces a plan that involves asking the passenger if he wants one of the alternatives, and the receiving the answer. The execution of this plan produces a system query corresponding to 'To Windsor?' and then recognizes the passenger's response 'yes'. The dialogue may continue at this level until the goal is achieved. In this case the first question/response pair achieves the goal. Once the passenger's goal is known the system can continue its original deduction task. The only reasonable interface with the 'travel to Windsor' expectation is the step involving knowing the departure time of the train. This is accepted as the passenger's obstacle. The system then plans to overcome the obstacle by informing him of the time, the execution of this plan produces a response corresponding to '3:15'.

5. Implications of the Approach

5.1 Dialogue Failures

We have seen in section 4.4 how a subdialogue was produced by the system in order to successfully complete his deduction of the passenger's plan. We now briefly consider a class of cases where the passenger initiates a subdialogue. These subdialogues indicate that a step in the system's plan did not successfully execute. e.g. After the interchange

P: When is the train?
C: It leaves at 3:15.

typical expected continuations are
'Thanks', '3:15.' or 'And where?'

Responses that indicate failure of the plan fall into three classes. These are all detected using what the system believes is the passenger's plan ('PP').

i) the passenger wants a goal that is already achieved in PP.
e.g. P: When is it?

ii) the passenger denies a goal already achieved in pp.
e.g. P: I don't believe it.

iii) the passenger wants a goal that is not present in PP, but that does fit another alternative expectation.
e.g. P: No! I want to meet the train!

When a failure is detected in the passenger's plan, the system should relate that failure point to an action in its own plan. Recovery entails resuming planning to achieve the effects of this action.

5.2 Linguistic Implications

There are other linguistic problems to which this approach seems to provide some solution. For example, consider

P: I want to go to Windsor. When does it leave?

This is interesting for the it has no previous referent. However, as speakers of English, we have no trouble identifying the referent as the train to Windsor. Our system handles this easily. The first utterance sets up the 'travel to Windsor' as the passenger's plan. The second then indicates that the passenger wants to know when something leaves, and this fits readily into our expected plan.

Finally, consider the problem of noun-noun modification. There is virtually no syntactic information provided in English as to the relationship between the two nouns. For instance, the phrase 'the Montreal train' could refer to the train to Montreal, the train from Montreal, the train owned by Montreal or perhaps the train operated by a team from Montreal in a train race! The point is, given appropriate context, the relationship could be just about anything. We hypothesize that the relationship should be relevant to the plan that the speaker is executing. Therefore, it should be revealed as the utterance is incorporated into its correct plan.

6. Implementation and System Status

The system is implemented in SPITBOL under TSO on an IBM 370/165. Our representation of knowledge is based on an object centred semantic network similar in approach to that of <Bobrow and Winograd 1975> and <Levesque 1977>. The belief and want representation is that described by Cohen <1978> and utilizes partitioned semantic nets as described in <Hendrix 1975>. Plan and actions are also represented in the net and may be executed using a network interpreter similar to that in <Norman and Rumelhart 1975>. An interactive network definition package allows for the quick definition of a domain into net form using a user-controlled syntax and allowing for the automatic inheritance of properties.

The system accepts English input which is parsed by a standard ATN parser <Woods 1970> in isolation from the rest of the system. This isolation is not particularily desirable, but such a scheme is sufficient for initial testing purposes. The parsed form is then transformed into an utterance hypothesis of the form described previously.

The system output is presently specified in the user-defined form used for the definition of new actions and states in the system. For example, the output corresponding to the English assertion 'the train leaves at 3' might be 'SYSTEM INFORM USER THAT TRAIN LEAVE AT 3'

The present implementation includes a network manipulation package, a planning system with the maintenance of the user model <Cohen 1978>, an ATN interpreter <Borgida 1975>, an interactive network definition package and the set of plan deduction subtasks. The system has handled the sentence fragment and the indirect speech act examples. The clarification dialogue capability is designed and currently being tested.

7. Concluding Remarks

Although all of our examples have dealt with one domain of discourse, the train station world, much of the system is domain independent. The domain is reflected in the set of actions and states defined, plus the set of initial plan expectations. Some specific heuristics are attached to the definitions of objects in the domain. For example, the heuristics used to identify trains. The planning, plan execution and plan deduction processes are all independent of the domain of discourse.

We have seen that viewing language as intentional behaviour provides solutions to a wide range of linguistic problems at the sentence level. Furthermore, the ability of the system to treat linguistic and non-linguistic processes uniformly allows it to handle dialogues concerning both the actual domain and the conversation process itself with the same techniques.

Although plan deduction has been attempted in the past, namely in attempts to recognize algorithms from code, it has never been applied to language. It is interesting to observe that its application to a seemingly more complex problem, i.e. language, actually should be an easier task. The fact that language is designed to facilitate the transmission of goals and plans has never before been exploited.

Bibliography

Bobrow, D. & Winograd, T., "An Overview of KRL, a Knowledge Representation Language",TR CSL-76-4, Xerox Palo Alto Research Center, 1976.

Borgida, A., "Topics in the Understanding of English Sentences by Computer", TR78, Dept. of Computer Science, Univ. of Toronto, 1975.

Brown, G.P.,"An Approach to Processing Task-Oriented Dialogue", ms, MIT, 1978.

Bruce, B. C., "Generation as a Social Action', in TINLAP, Schank and Nash-Webber (eds), 1975.

Cohen, P.R., "On Knowing What to Say: Planning Speech Acts", doctoral thesis, University of Toronto, 1978.

Fikes, R.E., Nilsson, N.J., "STRIPS: A new approach to the application of theorem proving", Artificial Intelligence 2, 1971.

Grosz, B.J., "The Representation and Use of Focus in Natural Language Dialogues", 5IJCAI, 1977.

Hendrix, G., "Expanding the Utility of Semantic Networks through Partitioning", 4IJCAI, 1975.

Horrigan, M.K., "Modelling Simple Dialogs, TR108, Dept. of Computer Science, University of Toronto, 1977.

Lehnert, W., "The Process of Question Answering", TR88, Dept. of Computer Science, Yale University, 1977.

Levesque, H.J., "A Procedural Approach to Semantic networks", TR105, Dept. of Computer Science, University of Toronto, 1977.

Norman, D.A. and Rumelhart, D.E., Explorations in Cognition, W.H. Freeman and Co., San Francisco, 1976.

Perrault, C.R. and Cohen, P.R., "Planning Speech Acts", AI Memo 77-1, Department of Computer Science, Univ. of Toronto, 1977.

Rieger, C., "Conceptual Memory: ...", doctoral thesis, Stanford, 1974.

Sacerdoti, E., "The Non-linear Nature of Plans", 4IJCAI, 1975.

Sadock, J.; Towards a Linguistic Theory of Speech Acts, New York, Academic Press, 1975.

Schank, R. and Abelson, R., "Scripts, Plans and Knowledge", 4IJCAI, 1975.

Searle, J.R., Speech Acts, Cambridge University Press, 1969.

Searle, J.R., "Indirect Speech Acts" in Syntax and Semantics, Vol. 3: Speech Acts, Cole and Morgan (eds), Academic Press, 1975.

Woods, W., "Transition Network Grammars for Natural Language Analysis", CACM Vol 13, No. 10, 1970.

Analyzing Conversation

Gordon I. McCalla

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada[1]

Abstract

Presented is a brief description of an approach to the modelling of conversation. It is suggested that to succeed at this endeavour, the problem must be tackled principally as a problem in pragmatics rather than as one in language analysis alone. Several pragmatic aspects of conversation are delineated and it is shown that the attempt to account for them raises a number of general problems in the representation of knowledge. A scheme designed to solve many of these problems is sketched and a typical conversation in a small scenario is analyzed in terms of this scheme.

1. Overview

In this paper I will discuss research into the problems of modelling natural language conversation. The work is founded on several perceived trends in the study of natural language. The first trend is a change in the focus of research attention from the phonetic and syntactic levels of linguistic analysis to the semantic and pragmatic levels. This not only is occurring in linguistics (e.g. recent interest in semantics (summarized in Leech (1974) for example) or Fillmore's (1975) interest in "frames"), but also in artificial intelligence (e.g. work on scripts by Schank and Abelson (1975); Bruce's (1975) social action paradigms; Bullwinkle's (1977), Grosz's (1977), and Cohen's (1978) work on conversation). The second trend is a blurring of the boundaries separating the various linguistic levels (for example Lakoff's (1971) criticisms of Chomsky; also the development of case frame theories of language (e.g. Fillmore (1968), Taylor and Rosenberg (1975))), a trend which tends to suggest that information from whatever level should be used when appropriate. The third trend is the consideration of language from a performance, rather than a competence, viewpoint (e.g. Grice (1957),(1968); much recent work in artificial intelligence, such as the script based models of Schank and Abelson (1975), or the frame model of Charniak (1977)).

These three trends are not diverging; rather, they seem to be coming together into a single viewpoint: language should be studied as it is used with semantic and pragmatic information driving the more purely surface aspects. However, this shouldn't preclude knowledge from whatever level being applied when relevant. Winograd (1976, 1977) argues persuasively for a similar approach to the study of natural language. These are the reasons why conversation, a domain in which language is used as naturally as possible and a domain in which semantic and pragmatic considerations are of utmost importance has been chosen for study.

There are many issues which must be resolved when trying to model conversation from such a pragmatics centred perspective. Of particular importance are

(i) whether world and linguistic knowledge can be effectively combined, and in particular whether language can be viewed as an activity like any other;

(ii) how the goals of a conversant affect what he says and how he understands;

(iii) how the knowledge a conversant has about the other conversants affect what he says and how he understands; and

(iv) how a conversant is able to focus on the relevant aspects and ignore the irrelevant aspects of any conversational scenario.

Throughout this research, the main concern has been to develop a general approach to resolving these four issues, sometimes at the expense of a detailed analysis of certain phenomena. Thus, for example, there are no penetrating insights into the problems of accounting for linguistic surface phenomena (especially the myriad difficulties of generating surface strings); nor into the problem of reference; nor into the problem of dealing with massively unexpected utterances or other surprises; etc. Suggestions are made as to _where_ such problems will arise, but there is little attempt to show _how_ to actually handle them.

The paper is organized as follows. In section 2 the basic approach to modelling conversation is described. In section 3 a microworld (called the "concert scenario") in which to study conversation is proposed. In section 4 an example is given of how a conversation which might occur in the concert scenario could be modelled using the approach of section 2. Finally, in

---

[1]This paper is based on Ph.D. research carried out in the Department of Computer Science, University of British Columbia, Vancouver, B.C. More details can be found in McCalla (1978).

Author's current address:
Department of Computational Science
University of Saskatchewan
Saskatoon, Saskatchewan

section 5 conclusions are drawn as to the contributions and shortcomings of this research.

2. The Approach

The viewpoint of this research is that linguistic capabilities are subgoals of non-linguistic capabilities. Thus, the easiest way to describe this approach to language is in terms of "level of goal", from higher level non-linguistic goals through lower level goals that are called in to understand particular parts of an utterance. Since goals can call in subgoals arbitrarily, it is sometimes rather hard to classify them precisely; they do seem, however, to roughly fall into four main levels: the non-linguistic level, the script level, the speech act level, and the language level.

(i) non-linguistic goals: Goals at this level are primarily concerned with undertaking significant plans of action such as, for example, attending a concert, buying a ticket to the concert, buying a drink at intermission, etc. Not primarily concerned with language, they do, however, know enough to call in linguistic subgoals when appropriate (e.g. to talk to the ticket seller or bartender). Perhaps as importantly, much of what is said is interpreted or produced in the context created by this level.

(ii) scripts: Scripts (akin to those of Schank and Abelson (1975)) are subgoals of non-linguistic goals (or sometimes of higher level scripts) called in to actually direct a conversation (e.g. the script to direct a conversation to buy a ticket). They are responsible for keeping track of the utterances of all parties to a conversation, for determining the sequence of speaking, for recognizing the beginnings and endings of a conversation, for using script expectations to aid the interpretation and production of utterances, and for meshing these expectations with the actual utterances produced. Scripts have available to them models of the conversants for use in performing their varied tasks.

(iii) speech acts: Speech acts (e.g. inquire, respond, inform) represent ideas expressible in a single verbal action by a lone speaker (the name has been chosen because of the similarity of goals at this level to the speech acts espoused by Austin (1962) and Searle (1969), and used more recently in AI systems such as that of Cohen (1978) or Bullwinkle (1977)). Called in by scripts, speech acts are responsible for interpreting or producing actual utterances, for checking that an utterance is not in conflict with the special requirements of a speech act of its type, and for making sure the utterance doesn't violate anything known about the conversant (available from a conversant model). Speech acts sometimes deal directly with surface linguistic strings, but more often call in language level goals to buffer them from the "real world".

(iv) language level goals: The speciality of this level of analysis is language itself. The tasks of goals at this level are to transform surface language into internal concepts (interpretation) or vice versa (generation). Such tasks involve appropriately grouping words (syntax), performing checks that the groups are consonant with known information discovered in the knowledge base or in the currently relevant context (semantics), and occasionally doing other tests perhaps involving such pragmatic considerations as looking at the conversant model or doing rather sophisticated inferences. Such sophisticated processing is the exception, however. Most often language level goals are concerned with parsing in a more traditional sense. I will not detail this level any further here. See McCalla (1978) for a fuller discussion of the language level or see Taylor and Rosenberg (1975) for a similar approach to parsing.

To actually model conversation in these terms, a large number of problems which are mainly problems in the representation of knowledge must first be tackled. That is, a representation scheme is needed that ideally would allow large amounts of information of various kinds (procedural or declarative, "real world" or linguistic, ...) to co-exist together, enable easy access to this information, incorporate a context or focussing mechanism, not collapse when presented with an unexpected situation, and satisfy both combinatorial and complexity considerations.

A representation scheme which attempts to come to terms with a number of these problems has been developed. It has been given precision by defining it in terms of a number of LISP functions (collectively named |LISP, pronounced "bar-lisp") which are, to date, not fully implemented, but whose semantics are specified fairly rigorously. A LISP-style notation has also been devised (some examples of which will be seen in the forthcoming description of a conversation to buy a ticket (see section 4)).

The representation scheme can be briefly summarized as follows:

(i) Knowledge is stored in objects which are closed to one another and can communicate only by passing messages (to borrow Hewitt's (1973) term).

(ii) The most interesting kind of object, called a pattern expression (|PEXPR), very roughly corresponds to a frame (Minsky (1974)) or schemata (Havens (1978)). Most domain knowledge is represented in such objects as patterns (in the PLANNER / CONNIVER tradition (Hewitt (1972), McDermott and Sussman (1974))). Since these patterns can be static (often encoding semantic

network-style "links") or can contain certain "active" macro elements (essentially "instructions" to be activated during message passing), procedural or declarative information can be represented.

(iii) Messages to a |PEXPR are also patterns that are handled by matching them against patterns in the |PEXPR and returning the first pattern that matches. The matcher is totally symmetric and will handle static elements or macro elements in either pattern of a match. This allows message patterns and |PEXPR patterns to be equally sophisticated. The central role for pattern matching here is consonant with its importance in much of AI (e.g. in PLANNER and CONNIVER, or in KRL (Bobrow and Winograd (1977))).

(iv) If a message pattern cannot be matched in a |PEXPR, failure to match processing (associated with the object whose name is the first element of the message pattern) can take place. This can range from trying to "inherit" the pattern along certain hierarchical "links" emanating from the |PEXPR (see Levesque and Mylopoulos (1978) for a discussion of the subtleties involved in doing such inheritance) through performing arbitrary inferences.

(v) A by-product of message passing is the creation of an activation record to which temporary variables and other local effects of the message passing are restricted. This activation record is called an execution instance and is a pattern expression like any other (and hence able to be accessed in identical fashion to other |PEXPRs).

(vi) As it handles a message, a pattern expression may need to communicate with another pattern expression. Whole chains of messages can be generated this way, with the consequent creation of whole chains of execution instances. They form a dynamic environment (akin to that of ALGOL or LISP) called the execution environment which turns out to be a very useful focussing and context mechanism as well as allowing the discovery of current goals.

(vii) Execution instances are not automatically removed after the message they were created in conjunction with has been answered. Instead, they stay around and old chains of them are consequently preserved. Such venerable execution environments can be accessed if the details of what went on in the past are needed. They thus produce a sort of episodic memory (to use the Tulving (1972) or Schank (1974) term).

3. The Concert Scenario

Discussed so far has been a goal-oriented approach to conversation and a representation scheme in which to attempt implementation of this approach. The actions of a conversation model built to simulate certain aspects of a particular scenario will now be described.

The model is assumed to be an active participant carrying out a (simulated) plan to attend a symphony concert. To carry out the plan, it can, among its many other tasks, engage in several conversations, including one to buy a ticket to the concert, another to buy a drink at intermission, one with a friend, etc. The "concert scenario" has been chosen because it is flexible and complex enough to provide a real test for the model and to illustrate most of the representation and language issues, yet it is of very finite dimensions. Moreover, various kinds of dialogue can occur including task-oriented (see Deutsch (1974)), non-task-oriented, formal, informal, etc.; non-linguistic goals occur and interact with the linguistic goals; and, finally, having a single scenario allows a small amount of information to be used in several, possibly quite different, settings.

As was mentioned above, the model engages in several conversations in the concert scenario. The task-oriented dialogue undertaken to buy a ticket to the concert will now be examined.

4. The Ticket Buying Conversation

Each piece of knowledge needed by the model in its task of buying a ticket to the concert is encoded (using the representation scheme outlined previously) as a pattern expression (|PEXPR). There are two main kinds of |PEXPR of relevance to this domain:

(i) primary pattern expressions to carry out parts of the main plan of attending a concert including going to the concert, buying a ticket, taking part in a conversation during the ticket purchase, and so on.

(ii) secondary pattern expressions, such as those representing models of the conversant, the agenda of the concert, and the like, that are sources of information for the primary |PEXPRs but aren't really part of the mainstream plan. The distinction between (i) and (ii) can be seen as the difference between active objects calling in other active objects to accomplish subgoals and static objects standing by to provide certain pieces of "foregrounded" knowledge when asked to do so by the active objects.

Figure 1 illustrates the primary |PEXPRs and their goal / subgoal dependencies.

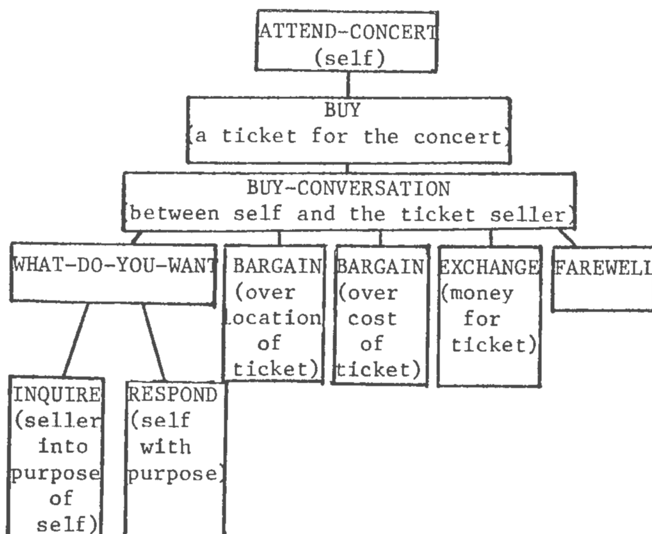


Figure 1-Primary Goals in Buying a Ticket

The model is assumed to have a high level goal of attending a concert (ATTEND-CONCERT) and in the process of doing so it calls in (among other goals) the subgoal of BUYing a ticket to the concert. This requires the model to engage in a conversation to buy the ticket (directed by the BUY-CONVERSATION script). This conversation has five phases: a greeting phase (directed by a script called WHAT-DO-YOU-WANT which expects the conversant to utter a speech act that INQUIREs into the purpose of the model and directs the model to undertake a RESPOND speech act outlining this purpose), a BARGAIN phase over the location of the ticket in the theatre, another BARGAIN phase over the cost of the ticket, an EXCHANGE of the agreed upon money for the agreed upon ticket, and, finally, an appropriate set of FAREWELLs.

The model will now be briefly described as it interprets and produces a couple of utterances in this scenario. Since the various goals are all encoded as |PEXPRs, when one goal calls in another, it means that it is sending a message to the other goal to EXECUTE some action, or to INTERPRET or GENERATE an utterance, or to EXPECT something to happen. The model starts with the high level goal of attending a symphony concert, so is EXECUTEing the |PEXPR ATTEND-CONCERT.

(i) ATTEND-CONCERT: a non-linguistic goal that directs the model's efforts to attend this particular concert. Conceptually, at least, it has been built by some plan-construction objects at the request of even higher level goals when they decided that attendance at the concert would be a good idea. It must achieve the many subgoals necessary to accomplish this goal. The major one of importance here is buying a ticket.

(ii) BUY: the model's non-linguistic |PEXPR to direct the buying of something. In this case ATTEND-CONCERT suggests buying a ticket to the concert. BUY is a simplified attempt to model what goes on in buying something. It must direct the model to the place of purchase, must recognize the particular seller of the item, must access |PEXPRs which represent various bargaining positions of the two parties to the buying, and finally must engage in a conversation to effect the purchase.

(iii) BUY-CONVERSATION: is the script that controls a conversation to buy something, in this case a ticket from the ticket seller. The script is outlined in Figure 2, and although the details aren't too important, the main script knowledge is encoded in the !(SCRIPT-SEQUENCE ---) element of the EXECUTE pattern.[2] The EXECUTE pattern of this pattern expression predicts the conversational sequence of events, from the greetings which open the conversation, through bargaining over the location of the ticket, bargaining over the price, exchanging the agreed upon goods for the agreed upon amount of money, and finally the goodbyes which terminate the script. Each of these predictions takes the form of EXECUTEing a sub-script to direct things. The first is rather poetically named WHAT-DO-YOU-WANT.

---

[2]For those interested in the hoary details, the process is explained here. When BUY wants to enter into conversation, it will send an EXECUTE message of the form

```
(EXECUTE BUY-CONVERSATION
 SELF TICKET-SELLER1 TICKET-FOR-CONCERT1
 ?CONV-RESULT)
```

to BUY-CONVERSATION. This message pattern will be matched against the EXECUTE pattern of BUY-CONVERSATION, with SELF, TICKET-SELLER1, and TICKET-FOR-CONCERT1 being bound as values of BUYER, SELLER, and ITEM respectively (as implied by the "?" macro); and with the !(SCRIPT-SEQUENCE ---) element being executed (as implied by the "!" macro) step by step much as is a PROG in LISP. When !(SCRIPT-SEQUENCE ---) is done, the value |RETURNed will be bound as value of CONV-RESULT in the message pattern, and the entire pattern will be returned to BUY.

```
<|PDEF BUY-CONVERSATION
  (SUPERSET BUY-CONVERSATION
    SOCIAL-TRANSACTION-CONVERSATION)
  (EXECUTE BUY-CONVERSATION
   ?BUYER ?SELLER ?ITEM
   !(SCRIPT-SEQUENCE ()
    STEP1
     (WHAT-DO-YOU-WANT
      (EXECUTE WHAT-DO-YOU-WANT
       !SELLER !BUYER ?WWCONV))
     (!BUYER
      (WANT !BUYER (EXCHANGE
          ?BUYER-HAS ?BUYER-WANTS)))
     (!SELLER
      (WANT !SELLER (EXCHANGE
          ?SELLER-HAS ?SELLER-WANTS)))
    STEP2
     (BARGAIN
      (EXECUTE BARGAIN !SELLER !BUYER
       !SELLER-HAS !BUYER-WANTS !ITEM ?B1))
    STEP3
     (BARGAIN
      (EXECUTE BARGAIN !BUYER !SELLER
       !BUYER-HAS !SELLER-WANTS !ITEM ?B2))
    STEP4
     (!ITEM (COST !ITEM ?AMOUNT))
     (EXCHANGE
      (EXECUTE EXCHANGE !BUYER !SELLER
       !AMOUNT !ITEM EXCH1))
    STEP5
     (FAREWELL
      (EXECUTE FAREWELL !BUYER !SELLER ?C))
     (|RETURN (|CURRENT))))    >
```

Figure 2 - The BUY-CONVERSATION Script

(iv) WHAT-DO-YOU-WANT: a script that knows about the kind of language that accompanies a query into the desires of somebody; EXECUTEd in this case by the BUY-CONVERSATION |PEXPR to handle the first couple of utterances in the conversation to buy a ticket. It expects the ticket seller to inquire into the purposes of the model; and expects the model to respond appropriately to this inquiry.

(v) INQUIRE: a speech act which will either understand or produce an "inquire" utterance, depending on whether the model is listening or speaking. In the current example, the WHAT-DO-YOU-WANT pattern expression expects the ticket seller to utter an inquiry into the purpose of the model, so INQUIRE is activated to understand such an utterance. Achieving this interpretation requires the |PEXPR to look into the input buffer for words which have actually been uttered. Discovering the word "yes", it checks to see if there is a speech act associated with the word "yes" which could be construed as an inquiry. Finding that there is (YES2), INQUIRE replaces itself with YES2 which continues the processing (since the actual input should take precedence over any expectation).

(vi) YES2: represents the meaning of "yes" that corresponds to an inquiry (rather than the meaning "affirmative answer"). It is called in to continue the understanding of the input. Knowing that YES2 is an inquiry into the current purpose of the model (which is available in the execution environment), it is able to achieve a proper interpretation of the utterance. Most speech acts, however, would not be so lucky, and would need to call in the language level to reduce the actual utterance to internal concepts which could be more readily understood. YES2, at any rate, is satisfied and returns to its calling |PEXPR.

WHAT-DO-YOU-WANT regains control; sees that the first utterance is just about as expected (if it weren't, WHAT-DO-YOU-WANT would have had to explain what went wrong); ties it into the conversation to date, being kept track of in the script (this tying-in could be a very complex process but is quite trivial in the current model). Before handling the next script utterance, WHAT-DO-YOU-WANT must first decide if something in the conversation to date is demanding priority over its script expectations, and if so, what to do about it. In this case, since the conversation to date has been more or less identical with the script's expectations, there is no such conflicting demand. Thus, WHAT-DO-YOU-WANT moves on to the second script utterance, the production of the model's response to the ticket seller's inquiry.

(vii) RESPOND: a speech act that contains the model's ideas about responding to a query, including how to understand or produce a response. Since WHAT-DO-YOU-WANT orders RESPOND to produce an utterance stating the purpose of the model, it does so (printing an appropriate surface string such as "I'd like a ticket to the concert.") Clearly a number of other objects would have to be called on here to decide such things as how much semantic information will get across the purpose of the model to the ticket seller (which involves, at least, looking at the beliefs of the conversant, of the model itself, and into the execution environment); how to phrase the eventual output; what words to choose; how to order them; and so on. These problems aren't treated to any great extent: I'm at present content with the knowledge that the model knows generally what to say at this point and don't really care if it says it well.

Once RESPOND is done, it returns again to WHAT-DO-YOU-WANT which must check that the utterance produced is appropriate, tie it in to the conversation to date, and then proceed to the next script utterance. But, since the script is now complete, WHAT-DO-YOU-WANT returns to the BUY-CONVERSATION script which, if satisfied with its behaviour, ties the WHAT-DO-YOU-WANT utterances into its conception of the conversation to date. It then proceeds to the bargaining scripts, the exchange phase (an only

partially linguistic goal), and the farewells which end the conversation. BUY-CONVERSATION then returns to BUY, which, when finished, goes back to ATTEND-CONCERT to continue with the plan of attending the concert. Among other things, more conversations could be undertaken before the ATTEND-CONCERT |PEXPR is satisfied, and these could be handled in much the same way as the ticket buying conversation.

This has been a brief look at some of the many action-packed primary pattern expressions in the concert scenario. But, these |PEXPRs need to access lots of data contained in other, secondary pattern expressions. Since secondary |PEXPRs act mainly as receptacles for knowledge, they are quite passive in their behaviour. Here, the important connections to other |PEXPRs are not the dynamic messages passed amongst primary pattern expressions, but are rather the more static semantic network type links of their patterns.

For any particular set of primary |PEXPRs, a certain collection of secondary |PEXPRs is needed, constituting in a sense, the foregrounded information for the primaries. The relevance of each secondary |PEXPR is discovered at some stage by a primary |PEXPR, which can record this fact by asserting a "pointer"[3] to the secondary in its execution instance. This pointer is then available (up the execution environment) to all subgoals of the primary |PEXPR when they want to access knowledge in the secondary |PEXPR.

When a secondary |PEXPR is asked about its views on some particular subject, it is sent a message just as would be done for any |PEXPR. It thus becomes (for the moment) a part of the plan and hence temporarily a sort of primary |PEXPR. But, the kinds of questions it is asked are so trivial (usually involving a simple match for a piece of data), that the secondary cannot really be considered to be executing a subgoal in the same sense as, say, BUY does for ATTEND-CONCERT.

So, lets look at some of the secondary pattern expressions that have proven useful in the concert scenario. The first set to be needed are CONCERT1, CONCERT, MASSEY-HALL, and AGENDA-CONCERT1 (see Figure 3).

```
<|PDEF CONCERT1
 (INSTANCE-OF CONCERT1 CONCERT)
 (LOCATION CONCERT1 MASSEY-HALL)
 (AGENDA CONCERT1 AGENDA-CONCERT1)    >

<|PDEF CONCERT
 (SUPERSET CONCERT EVENT)
 (ENTRANCE-REQUIREMENT ▼CONCERT
    ▼TICKET-FOR-CONCERT)   >

<|PDEF MASSEY-HALL
 (INSTANCE-OF MASSEY-HALL THEATRE)
 (TICKET-BOOTH MASSEY-HALL WICKET-MH)
 (LOBBY MASSEY-HALL LOBBY-MH)
 (BAR MASSEY-HALL BAR-MH)
 (AUDITORIUM MASSEY-HALL AUDITORIUM-MH)
 (SEATS MASSEY-HALL SEATS-MH)    >

<|PDEF AGENDA-CONCERT1
 (INSTANCE-OF AGENDA-CONCERT1
               AGENDA-CONCERT)
 (ORCHESTRA CONCERT1 TORONTO-SYMPHONY)
 (CONDUCTOR ORCHESTRA CONCERT1 DAVIS)
 (FIRST-HALF CONCERT1 JUPITER-SYMPHONY)
 (SECOND-HALF CONCERT1 EMPEROR-CONCERTO)
 (SOLOIST SECOND-HALF CONCERT1 BRENDL)    >
```

<u>Figure 3 - Concert Knowledge</u>

These pattern expressions contain all the model's knowledge about this particular concert, including the agenda, entrance requirement[4], the soloists, the location of the concert, etc. In particular, CONCERT1 is passed as part of the message that activates ATTEND-CONCERT. The other three |PEXPRs are pointed to from CONCERT1 and all are used extensively in further sub-|PEXPRs (especially to extract the entrance requirement to the concert).

Another secondary pattern expression, the TICKET-FOR-CONCERT1 |PEXPR (a new instance of the generic TICKET-FOR-CONCERT - see Figure 4) is generated by ATTEND-CONCERT when it is about to BUY the ticket.

---

[3]A pointer is actually just a pattern of the form

(pointer-name A B)

asserted in |PEXPR A and pointing to |PEXPR B.

[4]The ENTRANCE-REQUIREMENT pattern of CONCERT can be interpreted as saying that the entrance requirement for a particular instance of a concert is a particular instance of a ticket for the concert (this is implied by the "▼" macros).

```
<|PDEF TICKET-FOR-CONCERT1
 (INSTANCE-OF TICKET-FOR-CONCERT1
     TICKET-FOR-CONCERT)    >

<|PDEF TICKET-FOR-CONCERT
 (SUPERSET TICKET-FOR-CONCERT TICKET)
 (LOCATION ↓TICKET-FOR-CONCERT WICKET-MH)
 (REPN ↓TICKET-FOR-CONCERT
        =(X (SUBPART X SEATS-MH)))
 (COST ↓TICKET-FOR-CONCERT
 !(|PROG2
   (!TICKET-FOR-CONCERT
       (REPN !TICKET-FOR-CONCERT ?AREA))
   (|SELECTQ AREA
       (SEATS-MH-CENTRE '↓DOLLARS-10)
       (SEATS-MH-RCENTRE '↓DOLLARS-5)
       (SEATS-MH-LCENTRE '↓DOLLARS-5)
       (SEATS-MH-BALCONY '↓DOLLARS-3)
           NIL)))     >

<|PDEF TICKET
 (SUPERSET TICKET ENTRANCE-REQUIREMENT)
 (SELLER ↓TICKET ↓TICKET-SELLER)    >
```

Figure 4 - Ticket Knowledge

The TICKET-FOR-CONCERT1 pattern expression can inherit from TICKET-FOR-CONCERT knowledge that the physical location of the ticket is in the ticket wicket at Massey Hall, that the desired location represented by the ticket should be a certain place in the theatre[5], the projected cost for such a location, and so forth. Later, once actual characteristics (location, cost, etc.) of the ticket have been determined (by the BARGAIN sub-goal of BUY-CONVERSATION), they can be added to TICKET-FOR-CONCERT1.

Perhaps the most important secondary pattern expression is TICKET-SELLER1, first created as a new instance of the generic TICKET-SELLER (see Figure 5) by the BUY |PEXPR as part of its expectations about who will be selling the ticket.

[5]This information is contained in the REPN pattern of TICKET-FOR-CONCERT which can be interpreted as saying that a particular concert ticket will represent a location which is a subpart of the seats in Massey Hall (as implied by the "=" macro).

```
<|PDEF TICKET-SELLER1
  (INSTANCE-OF TICKET-SELLER1
                 TICKET-SELLER)    >

<|PDEF TICKET-SELLER
  (SUPERSET TICKET-SELLER SELLER)
  (SELL ↓TICKET-SELLER ↓TICKET)
  (WANT ↓TICKET-SELLER
   (EXCHANGE
    TICKET-SELLER-HAS-BARGAINING-POSN
    TICKET-SELLER-WANTS-BARGAINING-POSN)) >
```

Figure 5 - Ticket Seller Knowledge

The TICKET-SELLER1 pattern expression constitutes the model's model of the conversant. It initially contains only the knowledge that it is a TICKET-SELLER, but via its INSTANCE-OF pointer it is able to inherit much information. In particular the immediately superior TICKET-SELLER contains general information about ticket sellers: that they sell tickets, that they're willing to exchange tickets for an appropriate amount of money, that they are sellers, and so on. This information is of much use later on when various bargaining positions must be discovered. A truly complete TICKET-SELLER would contain information about speaking habits, probable locations, potential scripts, etc. As was the case with TICKET-FOR-CONCERT1, as time goes on, more and more characteristics of TICKET-SELLER1 can be added to give it an ever more accurate view of this particular conversant.

5. Conclusion

This example, although far too brief to be really convincing, does illustrate a number of interesting points. Issue (i), the intermixing of all kinds of knowledge, has been attempted - linguistic knowledge (in scripts, speech acts, and, although not shown, at the language level) is mixed with "real world" knowledge (ticket information, concert knowledge, ...); procedural knowledge (e.g. the EXECUTE patterns of BUY-CONVERSATION which accomplish subgoals) rubs shoulders with declarative encodings (e.g. macro-less patterns, in secondary |PEXPRs especially); linguistic and non-linguistic goals can call one another (e.g. the non-linguistic BUY EXECUTEs the BUY-CONVERSATION script which EXECUTEs, in turn, non-linguistic EXCHANGE).

Given that the whole organization is goal-oriented, issue (ii), the effect of the model's goals on its behaviour, is obviously of central import. The most striking instance: to understand and respond to "Yes?" the model must look at its current goals (available in the execution environment). Issue (iii), using conversant models, has been illustrated as well. Knowledge about ticket sellers helps the model at various

points, especially during the two BARGAIN phases of BUY-CONVERSATION.

Finally, in response to issue (iv), a context mechanism has been delineated, namely the execution environment. Not only does this focus the model's attention on only the relevant super-goals at any stage, but it also is the central core off of which hang pointers to secondary |PEXPRs. In a very real sense the further away a piece of information is from this core the less relevant it is in a given situation.

Obviously, much has been left unsaid (both good and bad) about this approach in general and about the representation scheme and the concert model in particular. Much left unsaid has also been left undone. An immediate priority is getting the representation ideas fully implemented and the concert dialogues running. Other priorities include analyzing in much more detail the problems presented by surface language; examining more flexible dialogues, especially those necessitating a much bigger role for "bottom-up" feedback; and increasing the information content of the conversant models to account for more subtle aspects of a conversant's influence on a conversation. Work by Cohen (1978) and Allen and Perrault (1978) will be of use here. Other extensions and improvements could be suggested. Hopefully, however, the framework provided here will prove sufficiently robust that such extensions and improvements can be readily incorporated without undue upheaval.

Acknowledgements

I would like to thank the many people who have asked questions, made suggestions, provided insights, or otherwise influenced me in the elaboration of the ideas presented here. I would like to single out especially Richard Rosenberg, Alan Mackworth, Ray Reiter, Rachel Gelbart, Jim Davidson, Bill Havens, Peter Rowat, and Brian Funt from UBC; Hector Levesque and John Mylopoulos from U. of Toronto; and Nick Cercone from Simon Fraser University. I would also like to acknowledge the National Research Council of Canada for supporting the more recent portions of this research.

Bibliography

Allen and Perrault (1978). J. Allen, C. R. Perrault, "Participating in Dialogues: Understanding via Plan Deduction", Proc. Second CSCSI / SCEIO Conference, Toronto, Ontario, July 1978.

Austin (1962). J. L. Austin, How to do Things with Words, Oxford University Press, Oxford, England, 1962.

Bobrow and Wegbreit (1973). D. G. Bobrow, B. Wegbreit, "A Model and Stack Implementation of Multiple Environments", CACM, 16, 10, October 1973, pp. 591-602.

Bobrow and Winograd (1977). D. G. Bobrow, T. Winograd, and KRL Research Group, "Experience with KRL-0: One Cycle of a Knowledge Representation Language", Proc. IJCAI5, Cambridge, Mass., 1977.

Bruce (1975). B. Bruce, Belief Systems and Language Understanding, BBN Rep. #2973, Bolt, Beranek, and Newman, Inc., Cambridge, Mass., 1975.

Bullwinkle (1977). C. Bullwinkle, "Levels of Complexity in Discourse for Anaphora Disambiguation and Speech Act Interpretation", Proc. IJCAI5, Cambridge, Mass., 1977.

Charniak (1977). E. Charniak, "Ms. Malaprop, A Language Comprehension Program", Proc. IJCAI5, Cambridge, Mass., 1977.

Cohen (1978). P. Cohen, On Knowing What to Say: Planning Speech Acts, Ph.D. Thesis, Dept. of Computer Science, U. of Toronto, 1978.

Deutsch (1974). B. Deutsch, "The Structure of Task Oriented Dialogues", IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, April 1974.

Fillmore (1968). C. Fillmore, "The Case for Case", in E. Bach and R. Harms (eds.), Universals in Linguistic Theory, Holt, Rinehart, and Winston, New York, 1968.

Fillmore (1975). C. Fillmore, "An Alternative to Checklist Theories of Meaning", Proc. First Annual Meeting of the Berkeley Linguistics Society, Berkeley, Calif., Feb. 1975.

Grice (1957). H. P. Grice, "Meaning", Philosophical Review, July 1957.

Grice (1968). H. P. Grice, Logic and Conversation, Wm. James Lecture, Unpublished Mimeo, Berkeley, Calif., 1968.

Grosz (1977). B. J. Grosz, "The Representation and Use of Focus in a System for Understanding Dialogs", Proc. IJCAI5, Cambridge, Mass., 1977.

Havens (1978). W. S. Havens, A Procedural Model of Recognition for Machine Perception, Ph.D. Thesis, Dept. of Computer Science, UBC, Vancouver, B.C., 1978.

Hewitt (1972). C. Hewitt, Description and

Theoretical Analysis (Using Schemata) of PLANNER, MIT AI Memo 251, Cambridge, Mass., April 1972.

Hewitt (1973). C. Hewitt, P. Bishop, R. Steiger, "A Universal Modular Actor Formalism for Artificial Intelligence", Proc. IJCAI3, Stanford, Calif., Aug. 1973.

Leech (1974). G. Leech, Semantics, Penquin Books, England, 1974.

Levesque and Mylopoulos (1978). H. Levesque, J. Mylopoulos, A Procedural Approach to Semantic Networks, AI-Memo 78-1, Dept. of Computer Science, U. of Toronto, Feb. 1978.

McCalla (1978). G. McCalla, An Approach to the Organization of Knowledge for the Modelling of Conversation, Technical Report, Dept. of Computer Science, UBC, Vancouver, B.C., 1978.

McDermott and Sussman (1974). D. V. McDermott, G. J. Sussman, The CONNIVER Reference Manual, MIT AI Memo 259a, Cambridge, Mass., 1974.

Minsky (1974). M. Minsky, A Framework for Representing Knowledge, MIT AI Memo 306, Cambridge, Mass., June 1974.

Schank (1974). R. C. Schank, Is There a Semantic Memory?, Report #3, Istituto per gli Studi Semantici e Cognitivi, Castagnola, Switzerland, 1974.

Schank and Abelson (1975). R. C. Schank, R. P. Abelson, "Scripts, Plans, and Knowledge", Proc. IJCAI4, Tbilisi, USSR, Sept. 1975.

Searle (1969). J. R. Searle, Speech Acts, Cambridge Univ. Press, Cambridge, England, 1969.

Taylor and Rosenberg (1975). B. H. Taylor, R. S. Rosenberg, "A Case Driven Parser for Natural Language", AJCL, Microfiche 31, 1975.

Tulving (1972). E. Tulving, "Episodic and Semantic Memory", in E. Tulving and W. Donaldson (eds.), Organization of Memory, Academic Press, New York, 1972.

Winograd (1976). T. Winograd, "Towards a Procedural Understanding of Semantics", Revue Internationale de Philosphie, 1976.

Winograd (1977). T. Winograd, A Framework for Understanding Discourse, SAIL Memo, AI Lab., Stanford University, Stanford, California, 1977.

Hypothesis Guided Induction: Jumping to Conclusions

Eugene C. Freuder
Department of Mathematics and Computer Science
University of New Hampshire
Durham, New Hampshire 03824

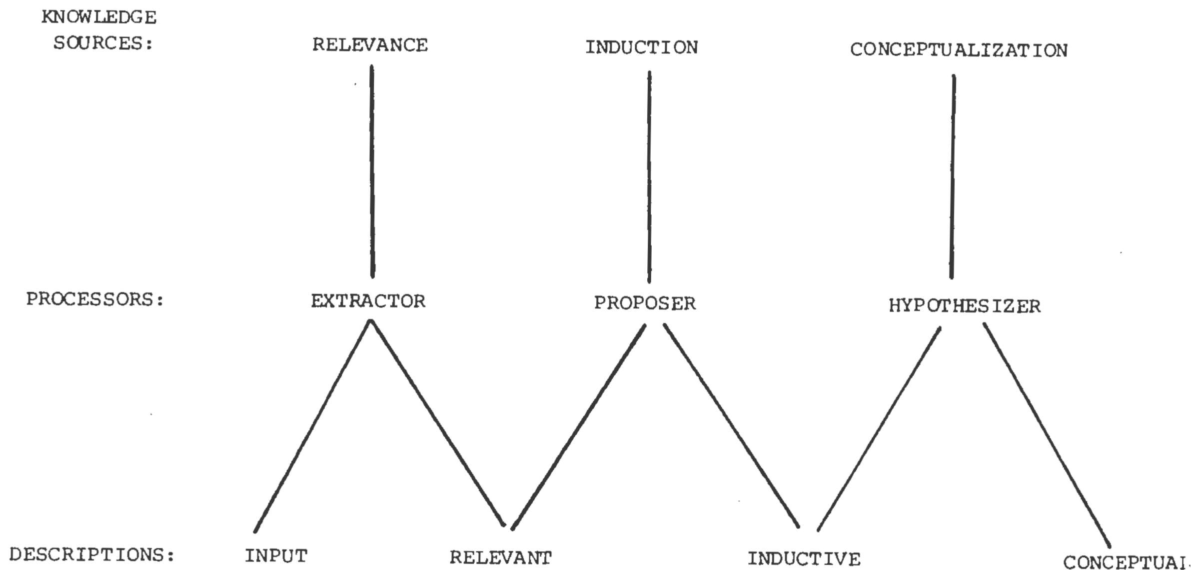
Inductive learning involves the formation of a new concept from examples and counterexamples. A classical approach to inductive learning in AI [Winston 1975] and psychology [Bruner, Goodnow and Austin 1956] utilizes the first example as the initial concept description, and generalizes that description based on successive examples and counterexamples. I suggest an alternative approach which uses the first example to produce a general hypothesis for the concept, and constrains or alters that hypothesis as required by further examples and counterexamples.

The former approach might be termed "conservative" and thus the latter "rash". A conservative approach gradually broadens concepts, based on current examples; a rash approach jumps immediately to general conclusions, based in part on knowledge distilled from previous experience. In some contexts I believe the rash approach will prove to be more efficient for machines and a better model of human behavior. Compromises between these two extremes may also have their utility.

I present a simplified model of the rash extreme and suggest a number of directions for further work. The implementation of the model is in progress; the program being implemented is called DUCK. I expect to test the program initially on Winston's domain of structural descriptions [Winston 1975] and the learning of poker hands, used as an example by Vere [Vere 1977].

The basic structure of the model is shown in figure 1. The desciption of a sample--example or counterexample--is processed by the EXTRACTOR and the PROPOSER. The initial input description, especially if taken from a "sensory" device like a TV camera, will have a good deal of extraneous information. The EXTRACTOR picks out the data deemed relevant to the learning task, producing a second description of the sample. The PROPOSER looks at this description for properties or patterns that are of inductive interest. These form the final description of the sample.

Figure 1



The HYPOTHESIZER chooses from the final description of the first example an hypothesis for the concept to be learned. The HYPOTHESIZER uses later samples to modify the concept hypothesis, when it fails to account for the samples.

Each processing module has an associated knowledge source. The processors are to function largely to "compile" the knowledge, making the expertise easily extensible in a given domain, or transferable to another domain, by extending, or replacing, the knowledge sources. The hope is that detailed knowledge appropriate to the learning task will increase the efficiency of the learning process, while a hierarchy of knowledge modules will permit DUCK to focus on the appropriate heuristics.

Consider Winston's canonical example of learning an arch. The EXTRACTOR would produce assertions about support relations and such. It would ignore explicitly, as Winston's program does implicitly, features like color, irrelevant to the structural concepts being learned. The PROPOSER would look for coincidences or extremes. The coincidence of two supports for the same block is about the only thing that stands out as of possible inductive interest. The HYPOTHESIZER takes this feature as the initial concept hypothesis.

Given a counterexample where the supporting blocks touch, the PROPOSER would pick out "touching" as an extreme value of the spatial proximity relationship. (One of the problems of learning by counterexample involves picking out the relevant differences, when the sample is not neatly confined to a single difference.) As "not touching" is not "strong" enough to function as a concept hypothesis on its own, it would be added to the initial hypothesis. The result is a good basic concept of an arch.

The learning model described here is "knowledge-based". There has been a recent surge of interest in knowledge-based learning, as evidenced by the papers at the latest IJCAI. Fox and Reddy discuss "knowledge guided learning" [Fox and Reddy 1977], Davis emphasizes "meta-level knowledge" [Davis 1977]. Goldstein and Grimson "annotate" production systems [Goldstein and Grimson 1977], while Vere supplies "background knowledge" [Vere 1977]. Michalski reports an algorithm for the determination of a "relevant" description set [Michalski 1977]. Lenat's knowledge-based heuristic search for new concepts is concerned with "interestingness" [Lenat 1977]. I find a particular relationship to the work of Soloway and Riseman [Soloway and Riseman 1977]. Earlier relevant knowledge-based efforts include the concept formation of Meta-DENDRAL [Buchanan, Feigenbaum and Lederberg 1971], and the program construction of Sussman's HACKER [Sussman 1975].

However, I think this present work is distinctive in its focus on a rash strategy, specializing an initial hypothesis, as opposed to generalizing the initial sample.

I present several speculations regarding extensions of this work on induction.

1. Intermediate models compromise the conservative and rash extremes.

How conservative or rash we are in our induction depends on context. If I am being shown the procedure for defusing a bomb, I will assume every movement must be copied precisely. When demonstrating more common tasks, we often take pains to point out deviations from the norm that will otherwise be assumed: "you have to push down while pulling this drawer open".

A compromise implementation of the conservative and rash approaches would be prepared to make hypotheses at an "appropriate" level of generality, and then to either generalize or specialize as required. Another compromise approach would involve carrying forward several alternative hypotheses in parallel.

2. Questions can implement hypothesis driven induction.

We often question the teacher, sometimes by providing samples of our own: "Is this a frammus, then?" We may seek to verify or clarify part of our hypothesis: "If it's blue, will it still be a frammus?" If we are carrying alternatives forward, questions can help distinguish among them.

3. Previously learned related concepts can help form hypotheses.

Related concepts serve as additional counterexamples. If we are learning a partition of a set of phenomena, an hypothesis for one which fails to rule out another is unacceptable. Related concepts can help choose appropriate properties from the relevant or input descriptions to specialize the concept hypothesis. Since most horses are big, size strikes us important in defining a Shetland pony.

4. Knowledge sources can be organized hierarchically, and themselves learned inductively.

Knowledge sources can be utilized in combination and organized hierarchically. Knowlege about sequences, for example, might appear under mathematical interest knowledge, along with knowledge of even numbers. There is no a priori reason why 2, 4, 6, 8, 10 should not be a valuable hand in poker. However, more specialized

knowledge, induced from experience with other card games, would not expect even numbers to play an interesting role in forming poker hand concepts.

This knowledge can be modified inductively, even while being used to learn other concepts. The system can learn, for example, that the color of card pips is not usually important; or when forced to return to the input description level repeatedly for a feature, it can decide to modify the inductive interest knowledge to look for that feature. When an advanced version of DUCK first enters a new domain it may have to be conservative. As it learns concepts in that domain it also learns learning strategies for that domain, embodying them in the knowledge sources. The more specific learning strategies justify more general, rash inductive behavior. When first learning card games, the system may not know that card backs are unimportant. However, the system learns to remove card backs from the properties of inductive interest. I think this reflects human learning behavior. This is one reason an experienced cook can learn a complex dish more easily than a novice learns a simple one. Lenat [Lenat 1977] has recently studied heuristics for determining "interest", and is currently working on a system which can learn these heuristics in turn.

5. Context influences our learning strategy.

Knowlege or expectations about the concepts to be learned can influence hypothesis formation. This may be fairly specific, constituting an outline or frame [Minsky 1975] to be filled in, or general: "I expect color to be important." We may also utilize knowledge of the learning process: "Why is he telling me that?"

6. The rash model can be hedged by "probabalistic induction".

Attaching probabilities to our hypotheses interpolates Winston's "must be", "must not be" extremes.

7. "Functional induction" can generate hypotheses.

If we know the purpose of tables, we know the top must support objects, then we may infer it important that the legs support the top.

8. "Deductive induction" can justify "leaping to conclusions".

Suppose we see a "Grunji fruit" for the first time, observing that it is purple. The next day someone asks us "what color are Grunji fruits"? We reply "purple". Induction from a single example, how rash! Yet if we see an exotic foreign car for the first time, observing it to be blue, we will not infer that all such cars are blue, quite the contrary. In Winston's terms, the color pointer would be moved from "blue" to "any color" in the car concept, while becoming a "must be" pointer for the fruit, after only the first example.

In fact, the rash inductions here are probably simple deductions. We have previously learned that "all fruits with the same name have the same color" (with several qualifications, I grant). Adding "one Grunji fruit is purple" then naturally implies that all Grunji fruits are purple.

References

Bruner, Goodnow and Austin. _A Study of Thinking_. Wiley. 1956.

Buchanan, Feigenbaum and Lederberg. A heuristic programming study of theory formation in science. IJCAI-71. 1971.

Davis. Interactive transfer of expertise: acquistion of new inference rules. IJCAI-77. 1977.

Fox and Reddy. Knowledge guided learning of structural descriptions. IJCAI-77. 1977.

Goldstein and Grimson. Annotated production systems: a model for skill acquisition. IJCAI-77. 1977.

Lenat. Automated theory formation in mathematics. IJCAI-77. 1977.

Michalski. A system of programs for computer-aided induction: a summary. IJCAI-77. 1977.

Minsky. A framework for representing knowledge. in _The Psychology of Computer Vision_, Winston, ed. McGraw-Hill. New York. 1975.

Soloway and Riseman. Levels of pattern description in learning. IJCAI-77. 1977.

Sussman. _A Computer Model of Skill Acquisition_. Elsevier. New York. 1975.

Vere. Induction of relational productions in the presence of background information. IJCAI-77. 1977.

Winston. Learning structural descriptions from examples. in _The Psychology of Computer Vision_, Winston, ed. McGraw-Hill. New York. 1975.

EXPLORATION IN VISUAL-MOTOR SPACES

by
Z.Pylyshyn, E.W.Elcock, M.Marmor and P.Sander
The University of Western Ontario
London, Canada

1.0 Introduction

This paper should be viewed as a provisional progress report on a wide ranging exploration of what might be called visual-motor reasoning. We began with an interest in the phenomenon of reasoning with the aid of diagrams, as in doing geometry. As we puzzled over the question of why diagrams appear to be of immense help in such reasoning, we became convinced that in order to understand this phenomenon we had to take into account both the architecture of the human cognitive system and its input-output transducers. In other words we concluded that it is not some inherent property of diagrams which makes them so useful, but rather it is the way this form of representation interacts with the particular set of primitive operations and resource-limited trade-offs characteristic of human cognition. Thus our original interest led us to consider a much broader set of questions than we had anticipated. It led us to ask what perceptual functions could be considered primitive; how spatial relations might be represented; how the intention to draw a figure meeting a certain description interacts with the available motor primitives (efferent commands) as well as with proprioceptive and visual inputs (the afferent signals). The latter, in turn, involves a consideration of the problem of perceptual-motor coordination.

While it is possible that not all these various aspects are equally relevant to the phenomenon of interest we felt that we could not justify leaving any of them out in developing the research program. For example, some of what we know about a diagram comes from the original description; some from the planned sequence of drawing operations; some from what is visually noticed in the course of executing this sequence and still others from scanning the diagram in the course of answering questions about it. If we did not consider the drawing and scanning components, the semantic representation of the diagram which would be built--and hence the relative complexity of various deductions--would not be the same. Furthermore the way we draw the diagram, the way in which we scan it, the features which we notice and the timing of the noticings and intermediate inferences are intimately related to the properties we attribute to each of these functions and to the way in which they interact. Hence it seems unlikely that we can account for the relative complexity of different diagram-aided inferences unless we attempt at least a schematic design of all relevant aspects of an entire perceptual-motor system underlying this kind of reasoning.

To have a concrete task to focus on we took one of the goals of this project to be the design of a system which can answer simple questions (put to it in some restricted format) about diagrams by actually going through the process of drawing and examining them directly, as well as by making inferences over its semantic representation of them. For example after drawing prescribed lines and figures and joining specified points the system should be able to determine which lines intersect and whether certain of the lines form specified figures or are contained inside other figures or are in certain relations to other lines and points.

We have approached this goal by first setting out what we consider to be some critical design decisions. These decisions follow the principle of attempting whenever possible to remain faithful to the known facts about human cognitive mechanisms. More accurately, we have tried to avoid making assumptions about available mechanisms which are implausible as psychological mechanisms, though they may be universal in general programming languages. In this respect we follow the lead of Allen Newell in his modelling of certain cognitive phenomena (Newell, 1973). We have adopted this strategy not solely because of an interest in modelling human cognitive processes but also because, as expressed earlier, we believe that reasoning-by-drawing has the properties it does precisely because of the nature of the human computational architecture. An additional guiding principle has been that whenever the relevant psychological facts are either unavailable or not unequivocal we attempt to use the least powerful (and therefore least presumptive) plausible mechanism capable of carrying out the required function.

An example of a consequence of adopting these principles is that we have avoided what might appear to be easy numerical solutions to some of the problems encountered. For example we do not maintain a computational equivalent of a Cartesian model of Euclidian space, nor the kind

of quantized model which could be provided using array structures as implemented in many programming languages. While there are usually many reasons for excluding such models, one straightforward one is that there is ample evidence that arithmetic operations (which map numerical expressions onto numerical expressions) are not primitive operations in the human architecture. Hence magnitude manipulation processes must be modelled by some other means (c.f. Pylyshyn, 1978). Incidentally the decision to shun the use of arithmetic applies only to the part of the system which models cognitive or symbolic processes. The parts of the system representing the retina and patterns of light on its two-dimensional surface or the intensity of signals going to and from limbs (on the non-symbolic side of the transducers) are modelled using numerical methods when appropriate. These are models of physical, not cognitive, events and so are immune from our strictures.

In the remainder of this report we will sketch the principle design decisions we have made concerning the visual component; the representation and inference component; and the motor system and coordination component which interact to generate the drawings.

2.0 The Visual Component

We are not attempting anything so ambitious as a general vision system. Nonetheless we do intend that the assumptions we make about the visual component be realistic and general, however incomplete. By working in the domain of two-dimensional drawings (oriented at right angles to the line of sight) we are able to bypass many difficult problems. By confining our domain to that of plane geometry we are able to restrict the low-level visual processes to ones which deliver only a small number of aggregate types, such as points, lines and junctions, rather than the much richer variety that might otherwise be needed (c.f. Barrow and Tenenbaum, in press). On the other hand problems of low-level vision are present and merge into problems of recognition, interpretation and inference as they do in general vision. Furthermore we do face the problem of building up a semantic representation from multiple sources of data, including data derived from visually scanning the diagram and detecting features as yet unknown to the system--i.e. not yet in the semantic representation.

2.1 The Retina

Visual information is assumed to come in as a grey-level retinal matrix. The size of the retina was dictated by the following considerations. If information on the retina was to be randomly accessible or processable in parallel, then making the retina the full size of the diagram would make it possible to bypass some important problems of representation. Since the "raw data" would always be available to interrogation by the perceptual primitives (see 2.6) the problem of selective encoding would not arise (though inference is still a problem). Alternatively if one did not assume parallel access by primitives then one would need to scan the retinal matrix somehow (cell-by-cell or perhaps by groups of cells) in order to build up a semantic representation. This however, is equivalent to having a smaller retina: it simply moves the diagram into the head and the retina in still farther!

Choosing a very small retina size relative to the size of the diagram (in the extreme it might be a single pixel) is to blur a distinction between perception and inference or problem solving that we wish to preserve. Indeed in our view it is the very existence of this distinction which makes reasoning through diagrams different from reasoning in general. With a small retina, information has to be put together serially, as a blind man with a cane must piece together a scene. If we assume that there are visual primitives which operate in a parallel manner over the retina then a choice of retina size represents a choice of how spatially local the primitive features of the system will be. This in turn partly determines the vocabulary over which the semantic representation is constructed, as well as the distribution of computational effort over the two categories of perception and deduction.

In our system, retinal size is a parameter. However we expect that the appropriate retinal size is one which rarely includes as much as a whole geometrical figure or as little as a single vertex with only a small fraction of the attached line segment in view. Evidence from the perception of anomalous figures (c.f. Hochberg, 1968) suggests that this is roughly of the right order for human perception of line drawings (though obviously we can make such drawings any size and view them at any distance. The evidence suggests, however, that when we do this our visual interpretation may change.)

We have not attempted to take into account peripheral vision (at least in the initial design) for a number of reasons. (1) Although the psychophysics of the visual field has been extensively studied, relatively little is known about the information processing function performed by peripheral vision; (2) the phenomena we are studying would be manifest, with respect to those aspects we are interested in, even if diagrams were viewed through tunnel vision; (3) the attention distribution mechanism to be described in 2.3 makes it possible to model almost any sort of peripheral processing notions by providing for the imposition of attention allocation restrictions favouring the centre of the retina (or on some other more complex basis).

2.2 Primitive Aggregation

The most elementary visual process is the aggregation of optical features on the retina into clusters. Such aggregation is assumed to be a primitive data-driven context-free non-resource limited process. It may be viewed as the inherent action of the optical transducer located at the retina. Exactly what type of aggregation functions are computed, whether they are limited in their spatial extent, whether they respond to optical continuity (region driven) or discontinuity (edge driven) or some other intrinsic property, whether their influence can propagate to neighbouring

regions, and whether they can cascade to produce more complex aggregated patterns in an automatic data-driven manner, are all fascinating open research questions. A number of promising proposals are available on such questions, including Marr's (in press) work on the primal sketch, Zucker's (1978) relaxation labelling scheme and Barrow and Tenenbaum's (in press) proposal for intrinsic property extraction.

Once again by confining our domain to drawings in plane geometry we can greatly simplify our system while still conforming to the basic insights reached in the research on low-level vision. We assume a basic unlimited-resource aggregation process for such retinal objects as points (including isolated points, endpoints, and intersection or function points), lines (with two, one or no endpoints visible), and a few combination aggregates (such as X's, T's, V's and perhaps even more complex but undescribed objects like "figure"--but these have not yet been explored). Each aggregate has one of a small number of type-flags associated with it. Every time a movement of the retina occurs the aggregation process is reactivated. And as each aggregate is formed it provides a potential interrupt of the next phase, which might be thought of as attention-capturing and which is not resource-unlimited.

2.3 Allocation of Attention

When an aggregate is formed it can be given a token of a resource-limited referencing mechanism, which can be thought of as a unit of attention, called a FINST (which, for historical reasons, stands for "instantion finger"). We expect that the allocation of FINSTs will be on some priority schedule based on such considerations as the type and location of the aggregate and perhaps also on characteristics of the higher level process active at the time, but little has been done on this scheduling aspect to date. When an aggregate dissappears (i.e. moves off the retina) its FINST is automatically returned to the pool. Retinal objects with FINSTs are referenced (or named) objects and hence can be referred to by the system. They may, for example, be identified as instances of objects known to the system and hence they may become bound to semantic or memory nodes (or, more accurately, to existentially quantified variables in assertions). They may, furthermore, be bound to arguments of various primitive perceptual functions and predicates as will be illustrated below. FINSTs, therefore, provide for a limited subset of retinal objects to be treated in a unitary manner and referred to by the system. Other important properties of FINSTs are discussed below.

2.4 Referential Continuity Over Retinal Translation

With a moving retina we must face the problem of recognizing the continued presence of the same object at different retinal locations. A relevant question is: at what level of perceptual analysis is the identity of retinal objects (except for translation) decided? One way to decide pattern identity might be to construct a description of the pattern at time t which is invariant over certain spatial transformations, say D(t), then construct another such description at a later time t', say D'(t') and compare the two descriptions. If they are identical then call the second pattern by the same name that was used to designate the first pattern. What seems unsatisfactory about this approach is that it requires that an abstract translation-invariant description be constructed prior to deciding that some pattern has moved. But in the case of continuous movement this is surely unreasonable. It seems more plausible in that case that recognizing that a pattern has moved across the retina should be independent of, and possibly even prior to, the further analysis of that pattern. There is evidence (such as discussed by Marr, in press) that perception of a pattern's movement is determined by cues that are more retinally local than the perception of its overall shape.

Our approach has been to hypothesize that maintenance of the identity of an object which has a FINST (i.e. which is a named aggregate) is a computational primitive. In other words we posit that the visual transducer is so constructed that once a FINST has been placed on an aggregate it remains attached to it as the aggregate changes retinal location quasi-continuously (i.e. by moving through all distinct intermediate retinal locations). The way in which this is to be implemented on a standard computer is not considered relevant to the function being modelled, since that function operates on a different machine architecture. For example in the relevant architecture, maintenance of continuity might be accomplished by cellular arrays. For us that function is simply a primitive operation--a building-block for describing the perceptual process.

Another way to view this property is to consider that what we have done is to assign identical names (FINSTs) to primitively aggregated retinal objects when we have a clear basis for concluding that they arise from the same physical objects on the diagram. This principle of labeling retinal objects in terms of properties of their physical source is an important one to follow in vision whenever it is possible to do so from local evidence. It is essential to do so, for example, in order to make use of stereoscopic depth cues since optical objects on the two retinas must be identified as arising from the same physical point before retinal disparity can be determined. Fortunately this too can be done from the local cues, on the basis of assumptions concerning the continuity of surfaces "almost everywhere". Similarly Barrow and Tenenbaum (in press) have argued for labeling retinal features on a basis which reflects the possible physical property of scenes from which these features arose. Our continuity maintenance assumption can be viewed as an instance of this principle. When movement across the retina is continuous it is easy to see how local evidence enables such an identity maintenance function to be primitively realized. When movement is in saccades the situation is more complex, but the brief persistence of the

preceding image together with some limited visual capacity during the rapid saccade might also provide the means to maintain identity primitively. For longer blackouts, however, it could well be that identity must be recognized by the more costly method outlined at the beginning of this section.

2.5 Attending to Higher Level Objects

Recall that a FINST can be placed on any primitively aggregated object and that the FINST remains on that object when the latter changes retinal coordinates continuously. For example, in the current system, as a retinal scan reveals more or different parts of a line continuously, a fixed object reference or FINST is associated with the changing aggregation. More formally and generally, it is the primitive aggregation process itself which determines what constitutes object invariance under such change. Thus what happens at an intersection (X) depends on the properties of the particular aggregation process. (Currently the system individually aggregates uninterrupted segments of lines as well as the entire set of concurrent segments so that the COG module can refer to these component parts individually). Similar principles would apply to curved lines--i.e. it would depend on the aggregation process.

The above illustrates the way in which a FINST can be attached to an aggregated object even when the entire object is not present on the retina. Furthermore, as more of the object appears it keeps the identity assigned to its first part providing only that it continuously becomes absorbed into the earlier object by whatever aggregation process is available. Should cascaded aggregations or region aggregations be permitted, it might be possible to cast such Gestalt phenomena as figure-ground isolation in these terms.

While we have not, for the time being, developed the notions of more complex data-driven aggregations, we have considered alternative ways in which higher level structures might receive FINSTs. This might be thought of as allocating attention to a figure as a whole, without necessarily allocating attention to its individual parts. Such a notion might be a useful way to try to make sense of Gestalt notions of holistic perception of patterns. To understand what it would mean to place a FINST on a higher level object we must examine again what function FINSTs serve in our system.

2.6 Primitive Visual Operations

One of the main reasons for introducing FINSTs is that once we have named objects we can bind these names to other variables such as nodes for particular known objects, nodes for general object types (in definitional structures) and to arguments in primitive visual operations or predicates. For example, the system then has a way of asking whether P is true of <u>this</u> and <u>that</u> since the latter two objects have <u>names</u> and <u>so</u> can be referred to in that manner (recall that these names uniquely refer to objects even when the latter change retinal coordinates and even when all of the potential aggregate is not on the retina).

Visual predicates have types associated with their arguments. The evaluation of a predicate is considered to be a computational primitive (i.e. a single step in the hypothesized architecture) providing that all its arguments are bound to objects of the appropriate type. Thus the evaluation of ONLINE (P : point-type, L : line-type) is primitive providing P is bound to an aggregated object of type "point" and L is bound to an aggregated object of type "line", and of course providing that both these objects have FINSTs on them (i.e. providing they are being "attended to" by name). There is no restriction at present on the permissible locations of P and L providing they both have FINSTS, which means that at least part of them is on the retina. Sample predicates include PARALLEL (L1 : line, L2 : line), PERPENDICULAR (L1 : line, L2 : line), ENDPOINT (P : point, L : line). In addition some of the operations have the effect of retrieving a FINST or even generating one for a retinal object. For example INTERSECT (L1 : line, L2 : line), or JUNCTION (L1 : line, L2 : line) returns either "false" or a FINST corresponding to the intersection or junction point. This may be interpreted to mean that the only way to verify that two lines intersect or meet is by visually noticing their common point.

The notion of visual primitive and that of a FINST are intimately related insofar as the only method by which FINSTs become interpreted and therefore assimilated into the semantics is through a visual primitive, and primitives are only defined over objects with FINSTs (we anticipate that the visual primitives may be represented as both data-driven demon (or if-added) form and process-driven servant (or if-needed) form).

This way of viewing FINSTs opens up the possibility of there being FINSTs on larger complex objects, even though the latter may not have been primitively aggregated but merely interpreted as an integral object (such as a triangle or even a pair of parallel lines) in the semantic representation. Following our characterization of FINSTs above we can think of a FINST being placed on a more complex figure (i.e. on a semantic node corresponding to that figure) as meaning that the figure is being singled out for attention in such a way that certain visual predicates can be primitively evaluated for that figure. One of the most eligible candidates for such a predicate is PARTOF (X : object, Y : object). Thus if there is a FINST on a triangle and another on one of its sides it is primitive to decide that the indicated line is part of the indicated figure. Our current proposal is to restrict predicates such as PARTOF to only take arguments from adjacent levels in a descriptive hierarchy (e.g. figures and subfigures, or lines and points, but not figures and points--the latter requiring a transitive inference and also requiring that intermediate-level objects be attended to as well).

Recall that FINSTs were introduced as tokens of attention in order to account for resource limited trade-offs. In view of this it should be the case that when one is attending to a figure as a whole one needn't also be attending to each of its parts. That is to say, attending to component parts requires additional tokens of attention or FINSTs beyond that placed on the whole figure node. A useful metaphor for thinking about FINSTs and the primitive PARTOF predicate is the following. When a FINST is placed on a node in some hierarchical description, that node and all its first level branches become the same colour. The primitive predicate is simply one which decides whether two objects in the graph have the same colour. In the case of a line having a FINST, we can think of the line having a unique colour. Deciding whether a point is on the line simply requires verifying that the point is the same colour as the line. The purpose of the graph colouring metaphor is to suggest that placing a FINST on a complex composite object causes an intrinsic primitively detectable property (viz. colour in the metaphor) to propagate down to its component parts. We have considered whether regions defined by lines should also be thought of as coloured, but that issue raises a number of deep problems which we have not yet resolved.

3.0 Representation and Inference

3.1 Qualitative and Quantitative Representations

The system's internal (post-transducer) representation of the overall diagram being drawn is, as stated earlier, primarily qualitative. Some coarse quantitative information is retained concerning selected relative distances and angles. This quantitative information is, however, represented in the form of a nominal or category scale (such as small, medium and large) using a symbolic notation which, like the Dewey decimal classification scheme, indicates both magnitude and precision. This form of representation has a number of psychologically interesting properties to recommend it but a discussion of the general issue of magnitude representation is beyond the scope of this report (our proposal is essentially the same as that of Marrand Nishihara, 1977).

Information in the overall representation (which we frequently refer to as the "semantics" because the terms of this data structure designate conceptual or interpreted properties) is stored and processed by the cognitive component of the system (the so-called COG module). Information in this representation arises from two general sources: the instructions to draw and the perceptual input. The first contributes the overall category names (e.g. triangle, square). In addition the drawing plan, assembled from definitions and additional constraints, supplies relations such as PARTOF and property names such as base, hypotenuse, altitude, right angle, midpoint and so on. For ease of communication with the user it also attaches external label names (such as might be written on the diagram) to internal node names (or COG nodes). External labels are not at present used in the recognition process since this would allow the identification of diagram parts by methods other than those we are primarily interested in.

The second source of semantic representation (perceptual input) provides a way to elaborate the semantics through visual discovery rather than through formal (axiomatic) deductive inference. Any visual predicate can provide a means of adding new properties and relationships to the evolving semantic structure.

An additional source of new perceptual information comes from monitoring non-visual sensory input--specifically kinesthetic information. It is possible to obtain magnitude information (distances and angles) from the execution of motor commands. In keeping with our attention-limited assumption, however, such information is not simply recorded whenever motor commands are carried out. The commands themselves are never recorded. What is sensed is the kinesthetic information and that only occurs when such information is being explicitly attended to and encoded. This explicit attending is not done using FINSTs but by executing movements in a special "monitoring" mode which results in the generation of magnitude information. Such information may be given a coarse symbolic encoding and added to the semantics or it may be used directly in a magnitude-comparison predicate. This procedure is necessary in order to accomplish tasks like finding the mid-point of a line segment when the segment is too long to fit on the retina. Though the general idea of dealing with magnitude (or other) information from kinesthesia (or perhaps from some proprioceptive time estimation) by some resource constrained means seems sound enough, we consider the current proposal as only a crude first step requiring much more development.

3.2 Inferences While Scanning and Recognizing

In order to discover new properties of the diagram while scanning it, the system must first identify the parts of the diagram it knows about. This recognition involves making inferences about the identity of objects. Such inferences can usually be made quite simply, given that the system always knows the identity of some of the objects (i.e. the FINSTs on some of the objects are bound to semantic objects or COG nodes), that it has a representation of the diagram which contains nodes for most of the retinal objects (this follows from the fact that most of the semantics are constructed while the figure is being drawn and from the continuity constraint on retinal motion), and that it has visual primitives which provide it with certain relations between the known and unknown retinal objects. In fact the only inferences which are permitted in this context are ones which begin with identified-object COG nodes and propagate over existing relation links using information obtained from visual primitives. We have deliberately confined ourselves to a weak inference scheme since we felt that continuity of perception with retinal movement should not have to depend on complex computations. By restricting the power of the deductive recognizer

to something resembling graph matching we place the additional burden of achieving the required level of performance on the design of an efficient representation and appropriate visual primitives. At present our deduction proceeds as follows.

The first step in identifying a new retinal object is to discover spatial relations between the object and instantiated objects currently on the retina (i.e. objects with FINSTs). The system does this by using "visual predicates". For each relation, its inverse is applied to the already instatiated objects and may produce a number of "candidate" uninstantiated COG nodes to match to the new retinal object. Now the system visually determines whether the new object satisfies the properties of each of the candidate objects. Any candidate COG node which has a property which is violated by the new object is rejected. By this graph-matching process, the number of candidate objects is reduced to (hopefully) one COG node which satisfies all the spatial relations between the new retinal object and the already instantiated retinal objects.

If several possibilities remain after this heuristic has been applied, other information (e.g. scanning direction) is used to try to further reduce the number of possibilities. If more than one possibility still remains for identification of the new retinal object, then instantiation is deferred. Identification of the object may be attempted again whenever the state of knowledge about retinal objects changes, e.g. if some other new retinal object is identified.

If there are no possibilities for identification of the new object, then the COG should be modified to accommodate it. For example, a new COG node is created for an intersection point which is being seen for the first time.

It seems unlikely that this method will be sufficient by itself. For one thing its use of movement information is still too rudimentary to provide ways of identifying objects when the retina moves over empty regions. It also does not seem to set up strong expectations of what will be encountered in scanning and the identification of objects seems somewhat more piecemeal than it might be. However it represents a first step and we felt it was better to begin with a weak heuristic and add additional facilities rather than doing the converse.

4.0 Coordinating the Motor System

Three objects in the system can be moved by issuing commands to the motor system. They are: the retina (or, more specifically, the position of the centre of the retina in relation to the diagram) ; the pen (which may be either down and leaving a trace, or up) ; and a third object, to be described below, called an anchor (we leave open the possibility of adding additional anchors, if they are needed). Both the pen and the anchor are visually distinct. When they are on the retina they can be primitively recognized. In addition the pen has a visually recognizable orientation so that, for example, the PARALLEL predicate can ascertain whether its orientation is parallel to that of an attended line. Because they are part of the motor system the position of these three objects can also be sensed kinesthetically. This dual sensing property will serve as the basis for perceptual motor coordination, as we shall see below.

Motor commands for moving these objects take LOGO-like or local polar coordinate form --i.e. direction and distance. However, we assume that all movements are under the control of a peripheral feedback loop through either the visual or kinesthetic system. Thus all motor commands are really of the form MOVE-UNTIL (<predicate>) or TURN-UNTIL (<predicate>), where the predicate is either a visual or kinesthetic one. To draw a line parallel to, say, the bottom of the paper (we assume figures are drawn on an area whose frame is visible) we can do a TURN-UNTIL command with the predicate being PARALLEL and its arguments bound to the pen FINST and the paper bottom FINST (assuming of course that both are on the current retina). Then we issue a MOVE-UNTIL command with the predicate being one which perhaps evaluates an intersection of the new line with one on the retina or which evaluates the (very coarse) sensing of n kinesthetic units of monitored motion.

The interesting problems arise when we wish to move an object to a currently unseen point or from a currently unseen point to a point on the retina. For example we might want to move the centre of the retina to some point we specify as a COG node (or a description or external label which evaluates to a COG node). Alternatively we may wish to move the pen, currently off the retina, to some object on the retina which has a FINST. We cannot simply issue the command to move to a particular FINST since neither the motor nor the kinesthetic system can interpret a retinal location. Retinal objects are not known to the system in terms of kinesthetic coordinates nor vice versa. In other words we cannot use, say, a TURN-UNTIL command with a visual predicate if the pen is not visible, nor can we say MOVE-UNTIL and specify a visual predicate under these circumstances because that does not specify (nor can it be made to specify) which direction to move the pen in order to bring it onto the retinal field where the predicate could be evaluated. The problem is how to inform the kinesthetic system, which can sense off-retinal locations, of the location of retinal objects and conversely how to inform the visual system of the kinesthetic coordinates cooresponding to retinal objects.

We have already explained that the "obvious" solution of maintaining a global cartesian or matrix representation of the whole diagram is unsatisfactory. While it would eliminate what we call the "cross-modality binding problem" by allowing both visual and kinesthetic systems access to a global diagram-centered frame of reference, this entails assumptions about the processor which we consider

unrealistic. However if this coordination across modalities is to take place it must respect the general resource-limited constraints we imposed on our design.

Our approach to the cross-modality binding problem has been to seek the minimally powerful mechanism, entailing the fewest assumptions about the nature of the underlying cognitive architecture, which could achieve the desired result. It seemed to us that the least we could assume is that there were a few objects which could be bound to both visual and kinesthetic sensors--i.e. whose location could be simultaneously available in both systems. The pen and the anchor (and to some extent the centre of the retina if we think of it as having a distinct visual sign--such as cross-hairs) are such objects. We can both distinguish them when they are on the retina (they automatically receive FINSTs) and sense them kinesthetically. The location of the centre of the retina in kinesthetic space can also be sensed. Thus starting from the pen or anchor on the retina we can move them to a particular FINST on the retina (typically one bound to a COG node) using a visual primitive. Then if we somehow move the retina so that neither the pen nor the anchor is visible any longer we still in principle have a way of determining their position in kinesthetic space and so in principle should be able to move the retina back to them. To capitalize on this fact we propose that it is a property of the motor system that it can be primitively instructed to move any of the three kinesthetically sensed objects (retina, pen, anchor) to the position occupied by either of the other two. Furthermore we propose that locations of each of these three moveable objects remain fixed in space, independent of any movements of the other two, unless they are commanded to move. This then allows us to, say, place the anchor at the centre of the retina, move the retina away to where the pen had earlier been left and then move both the pen and the retina together to the location of the anchor while drawing a line.

While these assumptions entail the existence of feedback systems capable of directing the movement of objects in such a way as to bring their sensed positions into coincidence, this seemed like the least that could be assumed and still make perceptual motor coordination possible. Furthermore they are closely related to a proposal made at the turn of the century by Henri Poincaré (in an essay entitled "Why space has three dimensions"). In discussing the question of how people's intuitions of a unitary three-dimensional physical space could have arisen, given the diverse sources of different information about location provided by our various senses, Pioncaré made the following observation. He noted that certain coincidences of inputs, such as might occur when what we independently conceptualize as a single object is being sensed by several sensory systems, could provide the boundary condition which results in a single coordinated space-frame developing. What we are proposing is that the system's ability to recognize, for a few objects whose movement it controls, that what it senses kinesthetically is also what it detects visually can provide a limited but crucial funnel through which we can effect visual-kinesthetic location binding, and hence the basis for perceptual motor coordination.

This elementary facility appears to be sufficient to allow a variety of drawing plans to be constructed using a "two-fingered" algorithm that leaves an anchor behind on the figure while it moves the retina to a new location. Using this procedure we can compile other useful composite commands. We can even think of such elementary functions as line tracking being carried out by placing the anchor on the line on the side of the retina towards which we wish to track and then moving the retina to it and iterating. Although further research might persuade us that additional motor primitives are plausible the minimal mechanism proposed represents a promising start. It bears a resemblance to the Marr and Nishikara (1977) minimal mechanism proposal for rotating a three dimensional model in their vision system.

5.0 Implementation

The design principles discussed in this paper, particularly those relating to the use of FINSTs in the visual processing, the use of cross-modality bindings in the scanning and drawing mechanisms, and the use of the described uniform recognition mechanism have been embodied in the heuristic implementation written in POP-10 (Davies, 1976) on the DEC-system 10. The details of the implementation together with sample run-time protocols are presented in a U.W.O. report (Pylyshyn et al. 1978).

Acknowledgement

This research was sponsored by the National Research Council operating grant A4092.

References

Barrow, H.G., and Tenenbaum, J.M. Recovering intrinsic scene characteristics from images. M.A. Hanson and E. Riseman (eds.) Computer vision systems, New York: Academic Press (in press).

Davies, D.J.M. POP-10 User's Manuel, Report #25, Dept. of Computer Science, University of Western Ontario, London, Canada, May, 1976.

Hochberg, J. In the mind's eye. In R.L. Haber (ed.), Contemporary theory and research in visual perception. New York: Holt, Rinehart and Winston, 1968, pp.309-331.

Marr, D. Representing visual information. M.A. Hanson and E. Riseman (eds.). Computer vision systems, New York: Academic (in press).

Marr, D. and Nishihara, H.K. Representation and recognition of the spatial organization of three-dimensional shapes. Phil. Trans. Royal Soc. (B), 1977.

Newell, A. Production systems: models of control structures. M.W.G. Chase (ed.), Visual information processing, New York: Academic Press,1973.

Pylyshyn, Z. On the explanatory adequacy of cognitive process models (mimeo), 1978.

Pylyshyn, Z., Elcock, E.W., Marmor, M., Sander, P. A system for perceptual-motor based reasoning. University of Western Ontario, Department of Computer Science, Report No. 42, 1978.

Zucker, S. Relaxation labeling and low-level vision. Proc. second international conference, Canadian Society for the Computational Study of Intelligence (CSCSI/SCEIO), University of Toronto, July, 1978.

USING MULTI-LEVEL SEMANTICS TO UNDERSTAND SKETCHES OF HOUSES AND OTHER POLYHEDRAL OBJECTS

by

Jan A. Mulder

and

Alan K. Mackworth

Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada V6T 1W5

Abstract

HOUSE, a computer program, can interpret sketches of houses and other polyhedral objects. This paper describes the design and current implementation status of HOUSE. The program uses seven levels of representation of the meaning of the sketch. It achieves a consistent interpretation at each level before proceeding to the next level. The interpretations produced on one level are used as cues to invoke models at the next level. The notion of consistency is extended to include both internal and external consistency. Consistent interpretations are arrived at through a uniform network consistency algorithm. The program is presented in the context of the goals of a sketch understanding project. HOUSE is evaluated with respect to its contributions towards satisfying those goals.

1. Motivation

The purpose of this paper is to report on the design of a program, HOUSE, that interprets sketches of polyhedral objects composed of meaningful parts, such as houses. The program, which has recently been implemented, is the latest result of the SEE project, a project set up to explore the interpretation of images designed for person to person communication. The goals of this project are:

i) to develop methods of exploiting the semantics of images designed for communication as typified by sketches,
ii) to explore possible solutions to the chicken and egg problem in perception: sensible segmentation requires interpretation and *vice versa*,
iii) to broaden the scope of vision programs by applying lessons learned in the blocks world to other domains,
iv) to provide an experimental vehicle for studying control structures required to implement schema-based theories of perception,
v) to make available useful interpretation programs for some restricted but important classes of sketches,
vi) to explore the relationship between natural and conventional representations.

2. The cycle of perception and MAPSEE

HOUSE is an offshoot of MAPSEE, a program designed for interpretation of sketches of geographic maps (Mackworth, 1977a). The assumption underlying both programs is that perception is an active process both data-driven and model-driven in character. Mackworth (1977b) has argued that all perceptual processes can be viewed as a cycle consisting of four processes: cue discovery, model invocation, model testing and model elaboration (see Fig. 1). In particular, all vision programs can be usefully characterized by how they embody this cycle. MAPSEE shows that a viable solution to the perceptual chicken and egg problem can be obtained by closing the cycle. In MAPSEE, the cycle is entered in the cue discovery phase, that is, a conservative, tentative segmentation into picture primitives (chains and regions) is done

first. A number of picture fragments are identified as cues in this segmentation. These cues give access to a number of domain dependent models.

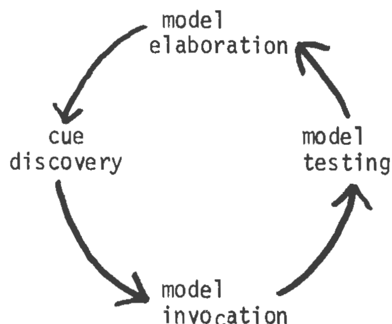


Fig. 1 the cycle of perception.

A model here is an interpretation, a naming, of the parts of the cue. A network consistency algorithm takes these models, together with the primitives in the picture which the models are supposed to interpret, and tests the consistency of the possible interpretations for the different primitives in the picture. The interpretations that survive the consistency tests provide a basis for sensibly refining and extending the segmentation.

3. Why HOUSE?

Clowes (1971) argued that all picture interpretation tasks involve formulating and manipulating descriptions in two distinct domains: the picture domain and the scene domain. Besides simply avoiding the confusion of linguistic category errors (lines and regions exist in the picture domain, edges and surfaces in the scene domain) this approach led to greater precision in the formulation of picture interpretation tasks. In MAPSEE, for example, chains of connected line segments in the cartographic picture domain correspond to rivers, roads, shore lines, coast lines, mountain sides and parts of bridges in the geographic scene domain; regions correspond to seas, lakes and landmasses.

In many tasks, however, the requirements of descriptive adequacy alone dictate that we need more than two distinct domains. Consider, for example, the sketch in Fig. 2a. We need first of all to distinguish the primitive connected points from the straight lines and the regions they appear to define. We need to represent the shape of the edges depicted (convex, concave, crack or occluding), the three-dimensional orientations of the edges and the shapes and orientations of the surfaces depicted (horizontal, vertical,...). Most importantly, we must, in this domain, go beyond three-dimensional geometric structure. We must be able to name surfaces according to their function in this architectural domain (wall, door, window), be able to describe and use their attributes (the walls are vertical) and interrelationships (the window is surrounded by the side wall and coplanar with it), and be able to interpret the whole as a functional entity, a house, as well as a three-dimensional polyhedral object.

In order to make these distinctions it is necessary to fracture the picture and scene domains into seven distinct domains. Since these domains are at least partially ordered with respect to semantic content or abstraction from the original image we shall call them levels. Contrasting this with MAPSEE where image cues invoke scene models we can see that HOUSE requires a cue/model hierarchy. The interpretation strategy in HOUSE is to achieve a consistent interpretation by following a MAPSEE-like cycle of perception at each level before proceeding onto the next.

4. Description

4.1 Levels of Representation

The seven levels of representation in HOUSE are:

1) Sketch level: the picture is represented as an interconnected set of points.
2) Line/region level: straight line representation and region representation.
3) Vertex level: the lines are interrelated by vertices, the region boundaries and shapes are computed.
4) Edge level: lines are interpreted as edges, relating the surfaces connected by the edge. The edge types possible are: convex (+), concave (-), occlude (>), occlude-concave (>̣)

and crack (c).

5) Orientation level: the three-dimensional orientations of both surfaces and edges are represented. This classification is very crude. Possible orientations are: vertical, horizontal or slanted.
6) Surface naming level: the surfaces carry meaningful names. For example, a surface can be ground, ground* (a horizontal surface coplanar with the ground such as a path), roof, window or door-handle. A surface is a side-face or top-face if it is part of a cube or a wedge.
7) Object level: the image is represented as an object. The possible objects in HOUSE are a cube, a wedge and a house.

Fig. 2 shows an image, interpreted as a house at the object level, represented at the seven different levels in the hierarchy.

4.2 Input

HOUSE receives a sketch in the form of a procedure for drawing it, created by the routines that track the stylus on a data tablet. The input is a sequence of plotter commands, a command being Move (pen up) to (x,y) or Draw (pen down) to (x,y) from the current position. Each series of pen down commands forms a chain of connected line elements.

4.3 Multi-levels of processing

The interpretation process strives to represent the image at the highest level possible. This is achieved by systematically bootstrapping up through the seven levels described above. A consistent interpretation of the image has to be achieved at each level before the step into the next level can be made. The cycle of perception serves as a metaphor for the description of the process. The cycle can be found at each level of processing, stepping through its four stages: cue discovery, model invocation, model testing and model elaboration. The objectives at each level of processing are always: 1) to construct a consistent representation and 2) to find the cues that allow bootstrapping into the next level.

4.3.1 Low Level Segmentation

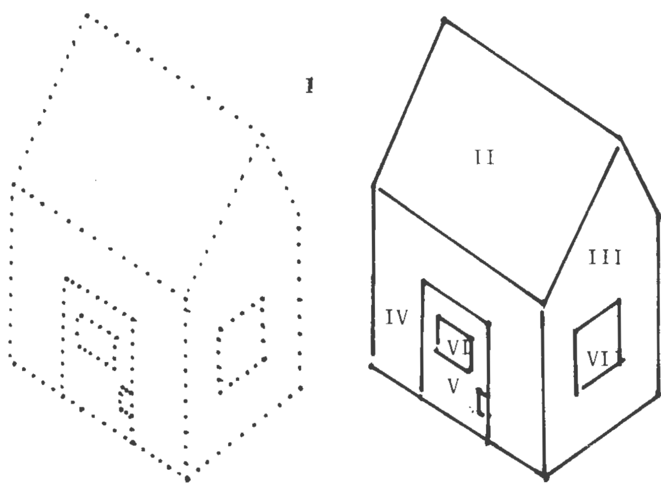
One of the lessons learned from the blocks world is that one needs to maintain a variety of representations each at various levels of detail in order to meet the demands of the interpretation task. These representations are created by means of four different segmentation procedures resulting in point, line, region and vertex representations.

Point formation. The points in the picture are represented in two different ways. First there is a network representation of the set of all points in the picture. Apart from this a coarse array representation is maintained (32x32). Each cell contains the list of points in that area. Quick answers to questions such as "what am I near?" can be given this way.

Line formation. A chain is defined as a set of interconnected points. The coarsest line representation of a chain is the straight line joining its end points. A procedure searches for the point in the chain furthest from that line and uses this point to split the line into two components. The chain is recursively subdivided until there are no free points left.

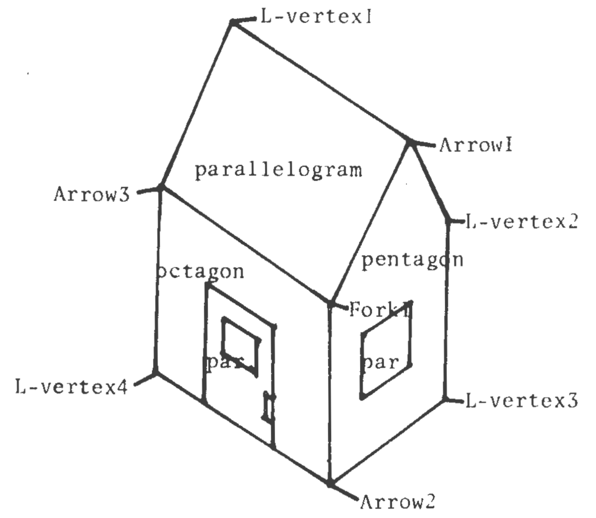
Vertex formation. The vertices used in HOUSE are: Free-ends, Links, L-vertices, Tees, Arrows and Forks (Fig. 3). Each vertex has its own formation procedure. These procedures are efficient in the sense that they use the line representation of each chain just up to the level of detail they require. The procedures are also conservative. For example, a merge of two Free-ends into an L-vertex or Link will occur iff the distance between the ends is very small. Thus, one prevents vertices from being merged that were not intended to be. Conservative segmentation will often miss genuine cues but, crucially, it will not supply false cues (Mackworth, 1977a).

Region formation. A region segmentation is achieved by subdividing the picture into empty patches, a patch being subdivided only if it is not empty.

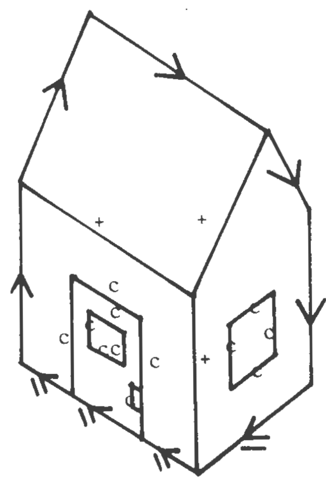


a) Sketch level

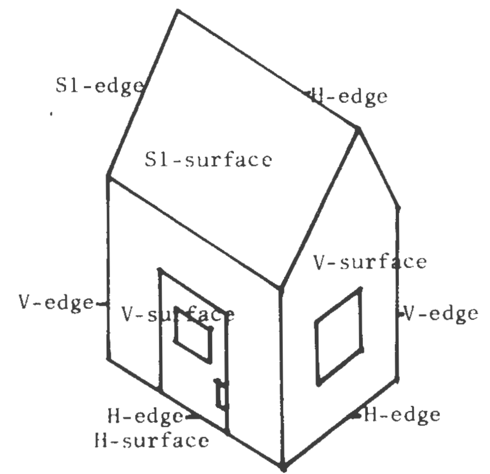
b) line/region level



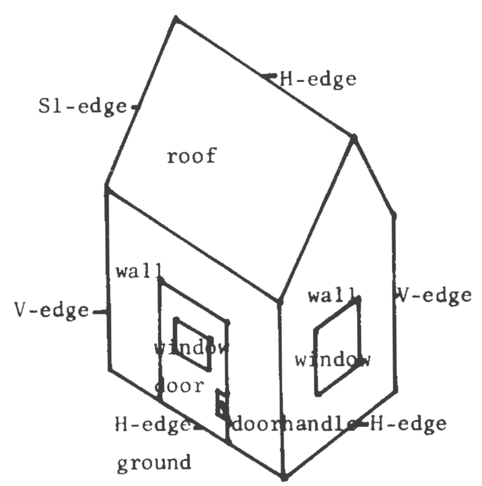
c) vertex level



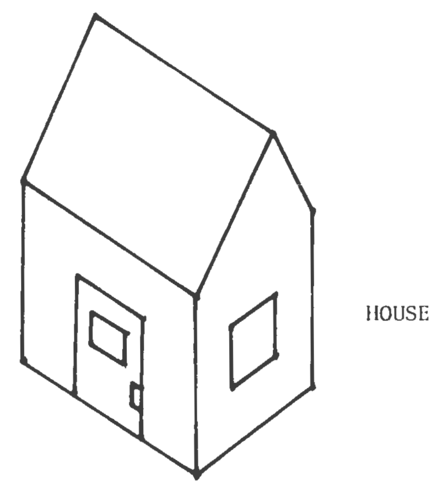
d) Edge level



e) orientation level



f) Surface naming level



g) Object level

Figure 2 Levels of Representation

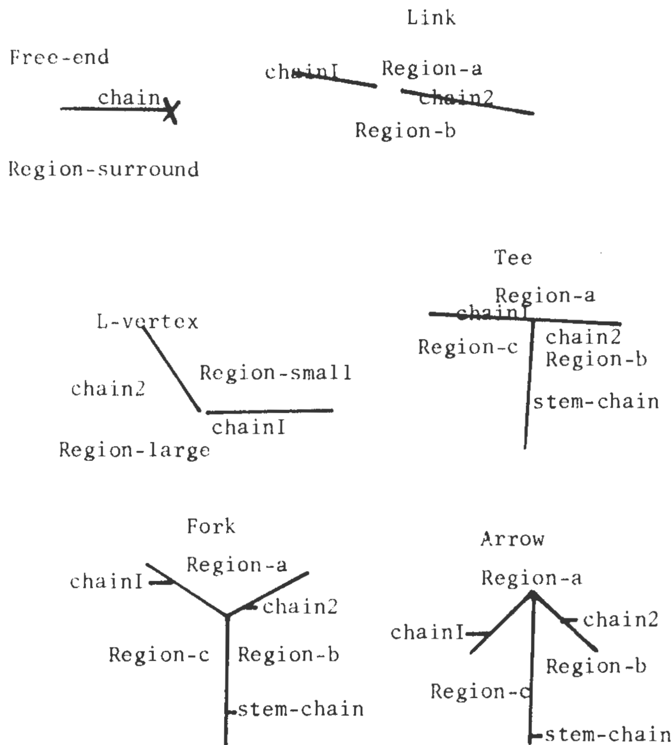


Figure 3. the vertices

Again, for conservative reasons, this process stops at a relatively large patch size.

The low level segmentation process continues through levels 1 to 3. This process cumulates in the formation of the low level cues (the vertices in Fig. 3) that allow bootstrapping into the interpretation cycle of the edge level, level 4. Chains and regions are the primitives in HOUSE. They are constrained by the vertices. Each vertex has a procedure at its disposal by means of which it can find out which regions it constrains. This procedure crawls along the bisector of each vertex line pair. This process is conservatively biased in the same way as the region formation procedure was. It will travel over a distance less than the size of the smallest patch along the bisector. If no region is found a region-ghost is created (Mackworth,1977a). Such a region-ghost stands for the region which has that relationship to the vertex but cannot yet be identified.

4.3.2 Cue interpretation tables

For each level of processing beyond level 3 there exists a set of cues which have procedures attached to them that will allow one or more interpretations for the primitives at that level. Fig. 4 shows the primitives at each level. A few examples of the constraints imposed at each level might be useful.

At the edge level we have used traditional Huffman (1971), Clowes (1971) and Waltz (1972) junction interpretations to interpret the edges.

At the orientation level, we have used extremely crude characterizations of the orientations of surfaces and edges (similar to but much cruder than those suggested by Waltz, 1972) into horizontal, vertical and slanted. The cues here are the edge types. A typical inference is that two surfaces separated by a crack must have the same orientation. A vertical line lying in a surface with a vertical orientation must be a vertical edge. These constraints are essentially compiled versions of the gradient space constraints exploited by POLY (Mackworth, 1973).

At the surface naming level, we use inferences such as, "the ground is horizontal, walls are vertical, roofs can be slanted or horizontal". Relational information such as, "windows share a crack edge with walls or doors and are surrounded by them" is also exploited here.

At the object level, certain cues must be present before the object can be called a house, cube or wedge. Some parts of a house (a putative wall containing a door or a window and connected via a convex edge to a putative roof) must be there before it can be a house. Other parts (e.g. door-handles) are optional, as in Winston's (1975) architectural models.

Edge level

cue type	domain(s)		
	chn		
free-end	convex,concave,occl-conc,occlude,crack		
	chn1	chn2	
l-vertex	occlude occl-conc occlude crack	occlude occlude occl-conc crack	
link	occlude convex concave occl-conc crack	occlude convex concave occl-conc crack	
	stem-chn	chn1	chn2
arrow	convex convex concave concave	occl-conc occlude occl/crack occlude	occl-conc occlude occlude crack
fork	convex concave convex concave	occlude concave convex occl-conc	occlude concave convex occl-conc
tee	convex crack crack	occlude crack occl-conc	occlude crack occl-conc

Orientation level

cue type	domain(s)		
	chn	region-a	region-b
occl-conc	h-edge	v-surface sl-surface	h-surface
	h-edge	v-surface sl-surface	h-surface
concave	h-edge	h-surface	v-surface sl-surface
	v-edge	v-surface	v-surface
	v-edge	v-surface	v-surface
convex	h-edge	sl-surface v-surface	sl-surface h-surface
	h-edge	sl-surface h-surface	sl-surface v-surface
	sl-edge	v-surface	sl-surface
	sl-edge	sl-surface	v-surface
	h/sl-edge v-edge	v-surface	v-surface
crack	h/sl-edge	sl-surface	sl-surface
	h-edge	h-surface	h-surface

Surface naming level

cue type	domain(s)		
	chn	region-a	region-b
occl-conc	h-edge	wall/door roof side-face top-face	ground
concave	h-edge	wall/door roof side-face top-face	ground*
	v-edge	side-face	side-face
	h/sl-edge	wall/roof	roof
convex	h/sl-edge	roof	wall/roof
	v-edge	wall	wall
	h-edge	ground	ground*
crack	h/v-edge sl-edge	window	wall

cue type	region-a	region-b
inside	window	wall/door
surrounds	wall/door	window
common	wall/roof side-face top-face	ground
	wall/door-handle	door
	door	wall/door-handle

cue type	region
v-surface	wall/door/window/door-handle side-face
sl-surface	roof/top-face
h-surface	ground/ground*/roof/top-face

Object level

cue type	domain
	object
common-wall-door-* convex-wall-roof-h-edge	house
convex-wall-roof-h-edge inside-window-wall-*	house
convex-side-face-top-face-h-edge convex-parallelogram-parallelogram-h-edge convex-side-face-side-face-v-edge convex-parallelogram-parallelogram-v-edge	cube
convex-side-face-top-face-h-edge convex-parallelogram-triangle-h-edge convex-side-face-side-face-v-edge convex-parallelogram-parallelogram-v-edge	wedge
convex-side-face-top-face-sl-edge convex-triangle-parallelogram-sl-edge convex-top-face-top-face-h-edge convex-parallelogram-parallelogram-h-edge	wedge

Figure 4 Cue interpretation tables

4.3.3 Network Consistency

At each level of interpretation, certain configurations of primitives, known as cues, invoke models which specify the allowable interpretations for the primitives. To ensure a globally consistent solution we must find an interpretation for each primitive such that each cue has at least one satisfied model. Two different types of consistency are required for each interpretation. An interpretation should be consistent with the internal description of the primitive (internal consistency) and it should be consistent with at least one of the interpretations for related primitives (external consistency).

External consistency is achieved by a network consistency algorithm, NC (Mackworth, 1977a). Input for this algorithm is a list (actually, a queue) of variable/relation pairs. The variables are the primitive chains and regions, the relations are the cue instances constraining the primitives they are paired with. The domain of each primitive is formed by its set of possible _a priori_ interpretations. NC takes the first pair (X,R) from the queue and checks for each value a in the domain of X to see if the other variables also constrained by R have at least one value in their domains that is directly constrained by R. If such a value cannot be found then a is deleted from X. An empty domain for X implies that there is no consistency in interpretations possible. If this is not the case then the queue is replaced by the union of the queue and the set of pairs obtained from all relations other than R that directly constrain X. These two steps are repeated until the queue is empty. If two or more primitives have more than one value left in their domains after this operation, the domain of one of the primitives is split in half and NC is applied recursively. An extensive discussion and elaboration of network consistency algorithms is given in (Mackworth, 1977c). The network consistency algorithm used in HOUSE is the same at all levels of processing beyond level 3.

Internal consistency is provided in the form of filters both in the cue interpretation tables and in the network consistency algorithm. These filters can prevent a model from being validated. For example, a slanted surface cannot have a vertical edge as part of its boundary.

Internal and external consistency are the equivalents of model testing and model elaboration, respectively, in the cycle of perception. The completion of model elaboration either starts the cue discovery at the next higher level or it leads to a resegmentation of the picture. Note that all the interpretations obtained at the present level and lower levels are potential cues for the next higher level.

4.3.4 Resegmentation

Resegmentation can be initiated for many reasons. The door-handle of the house in Fig. 2 is a good example. Because this region is too small, the conservative segmentation will initially overlook it. A region-ghost is created in its place because the region finding procedures crawling up along the bisectors of the angles will all fail to find a region. In the interpretation process, however, a region-ghost is treated as all other primitives. Their domains are provided with a set of initial interpretations, whose consistency is tested by the network consistency algorithm. A region-ghost with an interpretation left in its domain initiates resegmentation, that is, the region segmentation is further refined until the region-ghosts can be located and labelled as regions. Although region resegmentation is used in MAPSEE, it has not been implemented in HOUSE at the time of writing.

4.4 Output

The output of the program consists of a network of interconnected interpretations. An interpretation at each level consists of a list of all the primitives at that level; each primitive is paired with a valid interpretation for the primitive. Each interpretation at a given level is linked to the interpretation at the next lowest level that spawned it and the interpretations at the next highest level that it, in turn, has spawned. The number of such interpretations as

returned by the network consistency algorithm varies from level to level. The general trend is that the number of interpretations at each level increases up to the orientation level and decreases beyond this level. For example, a wedge has 7 interpretations at the edge level, 30 at the orientation level, 11 at the surface naming level and 3 at the object level, two of these representing a wedge, the third one representing its concave interpretation.

5. Discussion

It is not feasible to discuss all the implications of HOUSE in this short paper. For such a discussion the reader is referred to Mulder (1978). HOUSE's treatment of the cycle of perception should speak for itself by now. It is shown in Fig. 5.

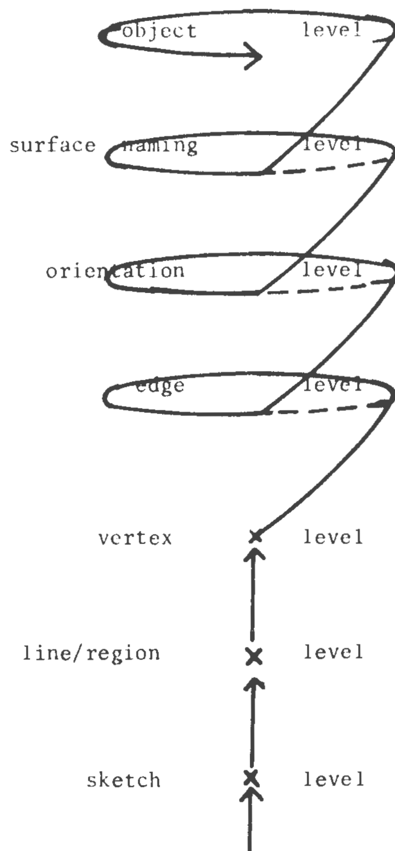


Figure 5. HOUSE's treatment of the cycle

We will limit ourselves here to a short discussion of the contributions of HOUSE to the goals of the project.

i) The exploitation of the semantics of an image in levels buys both modularity (and thus generality) and efficiency. For example, merging the orientation level and the surface naming level would have led to the same end results. However, the total number of combined interpretations to be tested by the network consistency algorithm would be much larger in the merged situation. Ruling out certain combinations at the orientation level prevents similar combinations from being made at the surface naming level.

ii) The chicken and egg problem "segmentation requires interpretation" remains a problem. Both MAPSEE and HOUSE show that a conservative segmentation of the picture, eventually corrected later by means of interpretation, can work. However, it is not difficult to think of examples in which even a conservative segmentation would be incorrect. We are working on solutions of this problem, but none of these has been implemented.

iii) The need for and implementation of multiple representations and levels of details, one of the lessons learned from the blocks world, was mentioned before. For example, the vertex finding algorithms make sophisticated use of multiple representations.

iv) The utility of stratifying the interpretation process does not require one to use a totally bottom-up approach as HOUSE currently does. Within the framework we have presented it should be easy to see that one can start processing at a higher level before finishing at a lower level. With the cycle running concurrently on many levels at once the constraints embodied in the cue interpretation table could propagate vertical consistency (Zucker, 1977) down the levels as well as up.
Schema-based theories of perception require a control structure different from HOUSE's present control structure. Schemata are active processing elements which can be activated in either a

top-down way or a bottom-up way (Bobrow & Norman, 1975). The models in MAPSEE and HOUSE are predicates treated as data structures by the network consistency algorithm. Schemata on the other hand take over control themselves, controlling both internal and external consistency. A next stage of the project may be to implement models that can take over control in order to test internal consistency. Having schemata that control external consistency as well would mean that we have to alter the network consistency algorithm. At this point, the project will probably diverge, one side focussing on the development of a general control structure for the interpretation of sketches, the other focussing on a schemata controlled interpretation. A substantial contribution to schemata-controlled interpretation has been made by Havens (1978a). Havens has designed and implemented a programming language called MAYA (Havens, 1978b) which is a multi-processing dialect of LISP that provides structures for representing schemata, and control structures for coordinating and integrating bottom-up and top-down schema instantiation.

v) HOUSE is a first step towards realistic architectural sketch understanding.

vi) One might expect that interpretations in the block world are more constrained by euclidian than by conventional representations as contrasted with the domain of geographic maps where conventional signs play a strong role. Data collected in an experiment done by Mulder and Mackworth (1978) however show, that slant estimates of cube surfaces by human subjects are controlled by three different schemata. Only one of these is purely geometrical; the other two display an intriguing mixture of geometry and convention.

6. Related Work

We have already discussed the commonality of the ideas behind the two programs and the influence of previous image understanding research. Levels of processing have also been proposed by Zucker (1977) for processes in a low level vision system. Network consistency algorithms and their development from binary arc consistency algorithms (Waltz, 1972) are described in (Mackworth, 1977c). Related algorithms have been proposed by Gaschnig (1974), Barrow and Tenenbaum (1976), Freuder (1976) and Rosenfeld, Hummel and Zucker (1976).

7. Acknowledgments

This research is supported by grants from the National Research Council of Canada to the second author and by a scholarship from the Cultural Exchange Section of the Canada Council to the first author. We are grateful to Bill Havens and Randy Goebel for discussion and comments on an earlier draft of this paper.

8. References

Barrow H.G. and Tenenbaum J.M. (1976), "MSYS : a system for reasoning about scenes", Tech. Note 121, A.I. Center, Stanford Res. Inst., Menlo Park, CA.

Bobrow D.G. & Norman D.A. (1975), "Some principles of memory schemata", in Bobrow, D.G. & Collins, A, (Eds.) Representation and Understanding, Academic Press 1975.

Clowes M.B. (1971), "On seeing things", Artificial Intelligence 2, 1, 79-112.

Freuder E.C. (1976), "Synthesizing constraint expressions", A.I. Memo 370, M.I.T., Cambridge, Mass.

Gaschnig J. (1974), "A constraint satisfaction method for inference making", Proc. 12th Ann. Allerton Conf. on Circuit Theory, U. of Ill., Urbana-Champaign, Ill., pp. 866-874.

Havens W.S. (1978a), "Recognition as a model for machine perception", Proc. of the 2nd Nat. Conf. of the Can. Soc. for Computational Studies of Intelligence (this volume).

Havens W.S. (1978b), "A procedural model for machine perception", Ph.D. Thesis, TR78-3, U. of British Columbia, February 1978.

Huffman D.A. (1971), "Impossible objects as nonsense sentences", Machine Intelligence 6, B. Meltzer and D. Michie, (Eds.), Edin. Univ. Press, Edinburgh, pp. 295-323.

Mackworth A.K. (1973), "Interpreting pictures of polyhedral scenes", Artificial Intelligence 4, 2, 121-137.

Mackworth A.K. (1977a), "On reading sketch maps", Proc. Of IJCAI-77, M.I.T. ,Cambridge, MA., pp. 598-606.

Mackworth A.K. (1977b), "Vision research strategy": black magic,metaphors,miniworlds and maps", in Computer Vision Systems, E. Riseman and A. Hansen (Eds.), Academic Press (in press).

Mackworth A.K. (1977c), "Consistency in networks of relations", Artificial Intelligence 8, 1, 99-118

Mulder J.A. (1978), M.Sc. thesis, Univ. of British Columbia (in preparation).

Mulder J.A. & Mackworth A.K. (1978), in preparation.

Rosenfeld A., Hummel R.A. and Zucker S.W. (1976), "Scene labelling by relaxation operations", IEEE Trans. On Systems, Man and Cybernetics, SMC-6, 420-433.

Waltz D. (1972), "Generating semantic descriptions from drawings of scenes with shadows", MAC AI-TR-271, M.I.T., Cambridge, MA.

Winston P.H. (1975), "Learning structural descriptions from examples", in Winston, P.H. (Ed.), The Psychology of Computer Vision, McGraw Hill, 1975, pp. 157-209.

Zucker S.W. (1977) "Towards consistent descriptions in vision systems", Proc. Of IJCAI-77, M.I.T., Cambridge, MA., p. 709.

A PROCEDURAL MODEL OF RECOGNITION FOR MACHINE PERCEPTION

William S. Havens
Department of Computer Science
University of British Columbia
Vancouver, Canada V6T-1W5

ABSTRACT

Aspects of a schema-based theory of machine perception are discussed. Perception is characterized as an active recognition task that employs schemata as a knowledge representation to compose new descriptions of observed experience. A procedural model of recognition for machine perception is presented. The model integrates top-down, hypothesis-driven search with bottom-up, data-driven search in schemata networks. Heuristic procedural methods are associated with specific schemata to guide their recognition. Multiple methods may be applied concurrently in both top-down and bottom-up search modes. The implementation of the recognition model as an A.I. programming language called Maya is described. Maya is a multiprocessing LISP dialect that provides data structures for representing schemata and control structures for integrating top-down and bottom-up processing in schemata networks.

1. Introduction

The creation of intelligent automata has been a compelling dream of mankind for millenia. Only in the last few years, however, with the invention of the von Neumann digital computer, has the realization of this dream been a serious possibility. Unfortunately, our high expectations have been maddeningly difficult to realize. In particular, we do not yet have an adequate theory of machine perception. However, as Mackworth (1977b) points out, elements of such a theory are emerging. Towards this end the research efforts of many have focused on the development of schemata as a viable machine representation of knowledge. This representation has appeared in various incarnations including frames (Minsky, 1975) (Winograd, 1975), scripts (Schank and Abelson, 1975), and schemata (Rumelhart and Ortony, 1976) (Bobrow and Norman, 1975). A second major effort has been the refinement of search mechanisms for schema-based perception (Kuipers, 1975) (Freuder, 1976).

This paper presents aspects of a schema-based theory of machine perception. The work is motivated by the belief that perception can be characterized as a recognition process that composes new descriptions of observed experience in terms of stored stereotypical knowledge of the world. A similar view has been advocated by Bobrow and Winograd (1977). A theory of machine perception is seen as having two major parts: a formalism for the representation of knowledge and a model of the processes and control structures required to perform search and deduction on that representation. This work focuses on the development of a procedural model of recognition for schema-based representations.

2. A Procedural Model

In this model, a schema is a modular representation of everything known about some stereotypical concept, object, event, or situation (Minsky, 1975). That knowledge can be manifest in three forms. First, each schema contains factual knowledge about the concept that the schema represents, and that knowledge may be realized declaratively, procedurally, or as some combination of data and attached procedure. At first, such facts are expectations

in an uninstantiated stereotype schema. These expectations are systematically replaced by specific values in the schema instance as they are recognized.

Second, each schema may contain procedural knowledge to guide the recognition process for that schema's stereotype. Such active heuristic knowledge is called a method. Instead of relying on general search methods, domain-specific methods can be associated with particular schemata to exploit special techniques that are particularly effective for the recognition of that schema. Methods may be employed in both top-down and bottom-up search modes.

And third, schemata form relations with other schemata thereby creating heirarchial network structures. This process allows complex concepts to be represented via the composition of more primitive schemata, as composition hierarchies. For a thorough discussion of schemata as a representation, see Winograd (1975).

The types of recognition processes that can operate on schema-based knowledge representations are constrained by perception itself. Perception cannot passively reflect sensation, but must be an active search process motivated by plans, expectations, and desires. Our sensory experience of the world is often ambiguous and illusory. Likewise, our knowledge of the world by which we interpret sensory experience is incomplete and often erroneous. Yet, perception must operate in this uncertain environment. The search process must tolerate indeterminancy by exploiting context and allowing multiple partial interpretations to be hypothesized and their confirmation attempted concurrently over time.

As well, machine perception must be both an active process guided by hypothesis and expectation and a passive process driven by events and sensory observations (Freuder, 1976). Observations behave as cues which both stimulate the formation of hypotheses and the activation of active heuristic knowledge associated with the specific hypotheses. Such hypothesis-specific knowledge is used to direct the recognition process by making observations, creating new expectations, and attempting to satisfy those expectations.

The expectations associated with stereotype schemata play an important role in the recognition process. They are dynamic properties of each schema that change as the uninstantiated instance proceeds toward being fully specified. At each point in this process, the schema's expectations represent what additional information is required to complete its recognition. From a different perspective, they represent the schema's knowledge of the world: what it expects to occur next or be found next from observation. Expectations provide the predictive power of models for their associated schemata.

Expectations may be represented by simple default values to be replaced by observed values when they are discovered, or they may be represented by complex patterns with attached procedural methods. These attached methods may use either goal-directed or data-driven search mechanisms (Rumelhart and Ortony, 1976). Goal-directed methods are designed to perform a top-down search of a schema's composition hierarchy in order to satisfy the methods' associated expectations. Data-driven methods, however, are designed to perform a bottom-up search of a composition heirarchy based on the satisfaction of their associated expectations.

For large knowledge bases, the recognition process cannot utilize purely top-down goal-directed techniques. Our knowledge of the world is far too complex to rely solely on goal-directed mechanisms. Machine perception must employ an integration of both goal-directed and data-driven search. Goal-directed search provides active guidance based on domain-specific knowledge of the hypothesis being attempted, whereas data-driven search uses the observation of cues to select likely hypotheses. See Havens (1976a).

Goal-directed search for schemata is realized by employing a top-down search of a schema's composition hierarchy. Top-down methods attempt to recognize instances of their schema's stereotype by making observations from sensory input and by calling on the efforts of the sub-schemata as subgoals (Kuipers, 1975).

Data-driven search in schemata requires the bottom-up search of a schema's composition hierarchy. In the model discussed here, a schema's method may be activated either from a higher schema as a <u>subgoal</u>, or from a lower schema as a <u>supergoal</u>. If a schema's method has been explicitly called as as a subgoal, then it must return a success or failure to its caller. However, if a schema has been activated as a supergoal, then it does not have an explicit caller. Instead, it must compute which higher schemata in the composition hierarchy its completed schema instance could plausibly be part of. This computation is facilitated by requiring each schema to contain inverse composition relations (usually called "part-of") with all plausible higher schemata.

Bottom-up, data-driven search in schemata depends on multiple supergoals being active simultaneously. Since it is desired that the recognition of schemata be conducted by procedural methods, these bottom-up methods must be allowed to apply their heuristic techniques concurrently. In bottom-up search, therefore, supergoal methods are realized as concurrent processes.

By allowing a schema to be activated from the bottom-up, the acknowledged deficiencies of goal-directed search in schemata can be avoided. A schema need not be hypothesized as a subgoal on blind expectation. Instead a likely schema is selected as a supergoal based on the recognition of supportive evidence from the input data. Once a schema has been selected as a supergoal, it may continue the search for a fully specified instance by using top-down techniques until one or more of the schema's expectations prove difficult to achieve. Since the supergoal exists as a process, it may then suspend itself bound to patterns representing those expectations until further supportive evidence is discovered.

When such matching evidence is found renewing the probability of the schema's success, the suspended supergoal is resumed. Supergoals iterate through a cycle, called an <u>expectation/matching cycle</u>, of being resumed by the discovery of matching evidence, computing a new set of expectations about their evolving schema instances, and then suspending themselves and possibly other methods to those expectations. Since multiple methods may be attached to multiple expectations, these expectations represent diverse possible directions for a schema's script. The eventual choice of a search path is not made by blind hypothesis but is data-driven, chosen by the discovery of evidence matching a particular expectation. The method associa-

ted with that expectation is then activated to continue its schema's recognition. The branches in a schema's non-deterministic script are therefore chosen deterministically by discovery of information supporting that path.

Machine perception must also be a recursive process. The recognition of complex concepts cannot be sustained using only primitive low-level cues. Instead, the recognition of concepts can be used recursively as internal cues in the perception of more abstract concepts. In the model discussed here, cues are both primitive features computed from the external world and abstract internal features represented as schema instances. When a method satisfies all its schema's expectations for a fully specified concept, that instance becomes an internal high-level cue. By allowing cues to be arbitrarily complex concepts, a context-sensitive cycle of perception (Mackworth, 1977b) is realized. Starting at the sensory data level, primitive cues present in the input are used to drive the hypothesis and recognition of low-level concepts. These features then behave as higher level cues that stimulate the hypothesis of more abstract interpretations. When a concept has been recognized, the completed instance uses knowledge of what higher schemata in any composition hierarchies it might plausibly be part of. It then attempts to match the expectations of those schemata. This process is called _completion_ in the recognition model.

3. _Maya_

The recognition model described above has been implemented as an experimental programming language called _Maya_ (Havens, 1976b). Maya is a LISP dialect designed to facilitate programming in schema-based systems. The language extends the data types defined in LISP, provides a multiprocessing interpreter, and defines control structure primitives for integrating top-down and bottom-up search in schemata networks.

Maya separates the LISP notion of an atom into two separate data types. In LISP, each atom is used both as a variable and as the name of a set of properties, usually implemented as property lists. In Maya, however, variables are differentiated from the names associated with property lists because it is desirable to distinguish between the value of a variable and the name of a data object. Variables are represented syntactically, as in LISP, by their atoms but names of objects are represented by their atoms prefixed by a colon.

The most important extensions to LISP's data types are the inclusion of objects and items. Objects subsume LISP property lists, the OBLIST, and can be used to represent schemata frames, and semantic networks. A schema or frame can be thought of as a collection of named slots. A node of a semantic network may be considered to be a set of named relations. Objects can conveniently represent both of these structures. Each object is composed of a set of associations between atomic names and arbitrary forms. A name is said to be _defined_ by its binding in some object. For example, the property list of the atom PRINT is an object which defines an association between the name SUBR and the system print function. Likewise, since the OBLIST is an association of LISP names to their definitions, it is considered the _global object_ in Maya.

Objects are created by the data primitive

OBJECT which has the following form:

(OBJECT <type><name$_1$><defn$_1$><name$_2$><defn$_2$>)

OBJECT creates a new object of user-specified type <type>, consisting of a set of associations between each atomic name <name$_i$> and its local definition <defn$_i$>.

Whereas objects associate atomic names with their definitions, items associate variable names with their local values. Items are sets of local variable instances. In Maya, generators such as the pattern matcher and the top-down and bottom-up control structure primitives always return items as values. These items contain the set of local variable bindings computed within the function.

In LISP, the interpreter's stack contains only function invocations and variable bindings. Maya's stack, however, also includes objects and items. Pushing an object onto the stack provides an incremental context mechanism. Since Maya is defined about a deep-binding scheme, objects pushed onto the stack alter the interpreter's "view" of the OBLIST. The object nearest the top of the stack is called the <u>enclosing object</u> and represents the schema or semantic network node in which the system is "operating." On the other hand, pushing an item onto the stack in effect creates a new instance of each variable contained in the item. Each variable's binding remains visible until another item containing the variable's name is pushed on the stack or the item is popped from the stack.

Objects and items are pushed onto the stack by a function called SEND, which takes a sequence of forms to evaluate:

(SEND <A$_1$> <A$_2$> <A$_3$> ... <A$_n$>).

SEND evaluates each form <A$_i$> in a left-to-right order. If the value returned from the evaluation of some <A$_i$> is either an object or an item, it is pushed onto the stack. Then <A$_i$+1> is evaluated from within this new context or new variable environment respectively. The sequence, <A$_1$> through <A$_n$>, can be viewed as a search procedure through a network of objects representing schemata. If the evaluation of some <A$_i$> yields an object, by pushing it onto the stack, the interpreter in effect "goes to" that object for the evaluation of the rest of the sequence, <A$_i$+1>, ..., <A$_n$>.

The three Maya data primitives GET*, PUT*, and REM* are analogous to their LISP counterparts GET, PUTPROP, and REMPROP respectively except that they operate on the enclosing object rather than the property list of a specified atom. They can be used to access, create, modify, and destroy schemata data structures. Their relationship to the more familiar LISP primitives is illustrated by the equivalence of the two following expressions:

(PUTPROP <atom> <name> <defn>)

(SEND :<atom> (PUT* <name> <defn>))

Maya provides a pattern matching system similar to that of CONNIVER's (McDermott & Sussman, 1973). Patterns consist of n-tuples of constants and pattern variables. Patterns can be matched against a datum consisting of another tuple or an associative index of tuples and their values called a <u>tuplebase</u>. Maya permits objects to contain arbitrary tuplebases. To represent a schema's expectations and attached top-down and bottom-up methods, patterns are bound to procedures and processes in the tuplebase of the object representing that schema.

Every pattern-matching and control function in Maya returns as value an item representing the binding of pattern variables assigned during the

match. The returned item also contains a reactivation tag that permits the pattern match to be resumed for an alternate choice. An item in Maya therefore has properties similar to the possibilities list of CONNIVER. It represents the set of all successful matches given a particular pattern and datum.

Maya defines control mechanisms for employing top-down and bottom-up methods within particular schemata. Both types of methods are realized by combining the notions of generators from CONNIVER and QLAMBDA expressions from QLISP (Reboh and Sacerdoti, 1973). QLAMBDAs are invoked by matching their pattern argument against a pattern datum. If the match succeeds, the pattern variables assigned during the match are used as the actual arguments to the QLAMBDA expression, and the body of the expression is evaluated. QLAMBDAs return items as values containing a reactivation tag for the generator. QLAMBDAs return control to their caller when the last form in their body is evaluated or when they suspend their execution. In this latter case, the reactivation tag is set to the form just past the suspension in the QLAMBDA body.

Procedural methods may be associated with schemata through various means. The simplest technique is to define a function local to the object representing some schema. In Maya, the function DEFUN always adds new function definitions to the enclosing object. At top-level, the enclosing object is the OBLIST, so new definitions are added globally as in LISP. However, DEFUN can define a new function to a schema by first making that schema the enclosing object. For example,

(SEND <schema> (DEFUN <foo> ...))

adds a local definition of the function <foo> to <schema>. This local function definition can then be evaluated via:

(SEND <schema> (<foo> ...)).

Top down methods in Maya are normally realized as QLAMBDA agenerators bound to patterns in tuplebases. Since tuplebases can be associated with specific schemata, these patterns represent the schema's expectations and the attached QLAMBDA expressions are considered top-down methods for satisfying those expectations. Maya defines two control primitives for pattern-directed invocation of top-down methods.

DIST generates items in depth-first order. In a tuplebase of QLAMBDA methods, <db>,

(DIST <pattern> <db>)

will invoke a method associated with a pattern argument that matches <pattern>. DIST returns as value the item returned from the QLAMBDA plus a new reactivation tag. If that tag is re-evaluated, DIST will recall that same QLAMBDA generator repeatedly until it fails to return a next item. DIST will then attempt to match the pattern of another QLAMBDA method in the tuplebase.

In contrast, BIST generates items in breadth-first order.

(BIST <pattern> <db>)

calls once each generator in the tuplebase <db> that matches the given pattern <pattern>. Only after it has called all possible matching QLAMBDAs will it recall each suspended generator for the second time, then each for the third time, and so forth.

The ability to apply top-down methods in either depth-first or breadth-first order is important. Generators are supposed to return possible items in order of likelihood. Yet the depth-first

only mechanisms of CONNIVER and other languages must completely exhaust the possibilities of one generator, no matter how unlikely, before considering the possible items of another generator in the database.

Bottom-up methods have been described as supergoals in the recognition model. Supergoals are implemented in Maya as processes which may be associated with specific schemata. Processes may be explicitly created and invoked by the Maya primitive PROCESS. The expression,

(PROCESS <schema> <ql> <pattern>),

attempts to match <pattern> against the pattern argument of the QLAMBDA expression <ql>. If the match succeeds, a new process is created and begun. The procedure body of the process is taken to be the body of the QLAMBDA expression. An association is made between the new process and the specified schema <schema>. If NIL is specified, then no association is made.

Once begun, the process will continue executing until either its procedure body is exhausted or until it suspends its execution. Control then returns to the call of PROCESS in the invoking process which returns as value an item from the match of <pattern> to <ql>. Thus, this item may be used to return values back from the terminating process.

As with top-down subgoal methods, bottom-up methods may be bound to patterns in tuplebases and invoked by matching those patterns. Again, these tuplebases represent the unsatisfied expectations of a partially recognized schema instance. When another fully-specified schema instance matches an expectation of a bottom-up method, the associated process is resumed as a supergoal. The process may then continue the recognition of its schema as a function of the now satisfied expectation. Processes are invoked from tuplebases via the primitive RESUME which has the following form:

(RESUME <pattern> <db>).

The pattern <pattern> is matched against a tuplebase of processes <db>. If the match is successful, the associated process is resumed from the point of its last suspension. When the resumed process suspends itself again or terminates, control will be returned to this call to RESUME which returns as value an item containing the pattern variable assignments made during the pattern match plus a reactivation tag. This tag may be used later to resume any other suspended processes in <db> that matches <pattern>.

Once a schema's supergoal has been resumed as a process, the supergoal continues the recognition of its schema. First, it incorporates the new information provided in the pattern match into the schema instance's evolving description. The supergoal may now continue the recognition of its schema using one of the three modes of search. Based on the added information provided by the newly satisfied expectation, the supergoal may attempt to complete the schema instance using top-down, goal-directed methods. On the other hand, if the likelihood of success is not high, then the supergoal can compute a new set of expectations for the schema based on the new information. It then suspends itself and possibly additional methods to these new expectations. The third possibility is that the supergoal suspends other methods to the new expectations but continues its own execution. By so doing it can now direct the discovery of information that may match its own schema's expectations. The choice is completely heuristic. If it yields

control to the process that invoked it, then that process will guide the bottom-up recognition process. When information is discovered, matching any of the supergoal schema's expectations, then the attached process will be resumed. On the other hand, if it retains control, then it can use domain-specific knowledge about its schemata to guide the bottom-up search process. In effect, supergoal search provides an "extra degree of freedom" not possible in subgoal search.

A supergoal suspends itself by calling the SUSPEND function which has the following form:

(SUSPEND <pattern> <db>).

The current process is suspended to the pattern <pattern> in the tuplebase <db>. Control returns to the process which invoked the current process. If the suspended process is later resumed, SUSPEND returns an item representing the match to <pattern>.

Supergoals conduct the recognition of their schemata in parallel with other schemata by going through an iterative expectation/matching cycle of computing expectations for the schema instance, suspending themselves and other processes to those expectations, and then being resumed by lower supergoals that match those expectations. This cycle begins when a schema is first proposed as a likely hypothesis and terminates when a complete schema instance has been recognized. Recognized instances then become abstract cues in the recognition of higher schemata thereby realizing a recursive cue/model hierarchy.

In Maya, when a supergoal concludes that its schema is fully instantiated, it attempts to match the attributes of this instance against the expectations of those higher schemata of which the schema could plausibly be part. Completion is realized using the primitive COMPLETE which has the following form:

(COMPLETE <pattern> <db>).

COMPLETE attempts to match the pattern <pattern> against the patterns in the tuplebase <db>. If a match is successful, all currently active processes associated with the same schema as the current process are suspended to a reactivation tag. For recognition tasks, it is assumed that all processes associated with the same schema instance and all their subprocesses are concerned with the recognition of that schema. Since the schema's recognition is complete, they are all suspended.

The process bound to the datum in <db> that matched <pattern> is then resumed. Included in the item returned to the resumed process is the reactivation tag. If that process terminates or suspends, all the suspended processes and sub-processes are re-instated, and COMPLETE returns an item representing the match of <pattern> plus a new reactivation for subsequent matches in the tuplebase <db>.

In the recognition model, a completing schema instance can match the expectations of multiple higher supergoals thereby realizing a mechanism for handling non-determinism in bottom-up recognition. By evaluating the reactivation tag, every schema having expectations matching <pattern> can be resumed as a supergoal. Indeed, completion is seen as a bottom-up generator of supergoals compared to the top-down generators of subgoals defined in CONNIVER.

4. Conclusion

This paper has outlined some procedural aspects of a theory of machine perception. Perception was characterized as an active recognition task that uses a schema-based knowledge representa-

tion to compose new descriptions of observed objects, situations, and events. It was argued that in order to cope with the complexity of everyday experience, the recognition process must be guided by active heuristic procedural methods. Such methods, associated with particular schemata, direct the recognition of instances of those schemata.

The perceptual process must also utilize an integration of top-down and bottom-up search methods. Top-down techniques can be used to efficiently confirm the schematic aspects of recognition. Bottom-up techniques can be used to generate hypotheses, discover cues, and handle anomaly and ambiguity.

The desirability of employing a recursive cue/model hierarchy for perception was also pointed out. By allowing completing schemata to act as internal cues in the invocation of higher schemata, a mechanism is defined for realizing a context-sensitive and recursive cycle of perception.

Finally, the implementation of the recognition model as a programming language called Maya was described. Data structure extensions to LISP for implementing schemata were presented and control structures for realizing integrated top-down and bottom-up search in schemata networks were defined. Currently, Maya is an operational language system running within MTS LISP. A number of application programs have been written in Maya. Future plans include incorporating Maya into the new multiprocess INTERLISP system under development at UBC and then applying the recognition model via Maya to the schema-based interpretation of Landsat images.

References

BOBROW, D. G. & NORMAN, D. A. (1975) Some Principles of Memory Schemata, in D. G. Bobrow & A. Collins (eds.), Representation and Understanding, Academic Press, New York

BOBROW, D.G. & WINOGRAD, T. (1977) An Overview of KRL: A Knowledge Representation Language, Cognitive Science, Vol. 1, #1, Jan. 1977

FAHLMAN, S.E. (1975) Thesis Progress Report: A System for Representing and Using Real-World Knowledge, AIM-331, A.I. Lab, MIT, Cambridge, MA

FREUDER, E.C. (1976) A Computer System for Visual Recognition Using Active Knowledge, Ph.D. Thesis, AI-TR-345, MIT A.I. Lab, Cambridge, MA

HAVENS, W.S. (1976a) Can Frames Solve the Chicken and Egg Problem?, Proc. First CSCI/SCEIO, UBC, Vancouver, Canada, August 1976

HAVENS, W.S. (1976b) Maya Language Reference Manual, TM-13, Dept. of Comp. Sci., University of British Columbia, Vancouver, Canada

KUIPERS, B.J. (1975) A Frame for Frames: Representing Knowledge for aecognition, in Representation and Understanding, D.G. Bobrow & A. Collins (eds.) Academic Press, NY

MACKWORTH, A.K. (1977a) On Reading Sketch Maps, TR-77-2, Dept. of Comp. Sci., University of British Columbia, Vancouver, Canada, also Proc. IJCAI-77, MIT, Cambridge, MA, August 1977, p. 598

MACKWORTH, A.K. (1977b) Vision Research Strategy: Black Magic, Metaphors, Mechanisms, Miniworlds, and Maps, Proc. Workshop on Comp. Vision Systems, June 1977, Univ. of Massachusetts, Amherst, MA

MINSKY, M. (1975) A Framework for Representing Knowledge, in The Psychology of Computer Vision, P. Winston (ed.), McGraw-Hill, NY

REBOH, R. & SACERDOTI, E. (1973) A Preliminary QLISP Manual, Stanford A.I. Lab, Techn. Note #81, August 1973

RUMELHART, D.E. & ORTONY, A. (1976) The Representation of Knowledge in Memory, TR-55, Center for Human Info. Processing, Dept. of Psychology, Univ. of California at San Diego, La Jolla, CA

SCHANK, R. & ABELSON, R. (1975) Scripts, Plans and Knowledge, Proc. IJCAI4, Tbilisi, Georgia, USSR, September 1975, pp. 151-157

WINOGRAD, T. (1975) Frame Representations and the Procedural-Declarative Controversy, in Representation and Understanding, D.G. Bobrow & A. Collins (eds.), Academic Press, New York

Approaches to Object Selection for General Problem Solvers

Phil London
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract - I describe first steps toward a unified theory of object selection. Object selection is the problem solving process whereby objects are chosen to participate in a synthesized plan. Two distinct object selection techniques and their relationship to a dependency-based modelling mechanism are discussed.

1. Introduction

Selection of strategies and objects is a fundamental activity of general problem solving. Strategy selection involves choosing the most appropriate strategy for achieving a goal in a particular problem solving environment. This is a frequently investigated topic, and techniques have been developed ranging from heuristic search [N1] to "intelligent" selection of strategies by evaluation of the planning environment using procedures [S1] or discrimination nets [R1], [RL1].

Frequently, strategies chosen as appropriate in a given problem solving context require that certain objects participate in their execution. Thus, object selection is an important part of a problem solver's skills, providing the ability, for example, to decide on which peg to place the top disk in the Tower-of-Hanoi, or which hammer to select during a carpentry task.

Because of the obstacles encountered in developing good strategy selection techniques and methods to deal with the pervasive problem of subgoal interactions (see [RL1], [W1], [S1], [M1], for instance), object selection has been often overlooked. Nevertheless, object selection is an integral part of the overall problem solving enterprise. The purpose of this paper is to explore some of the issues involved in constructing a unified theory for object selection.

2. Models

Flexible object selection techniques, as well as strategy selection techniques, require a flexible modelling mechanism. The "world model" is a fundamental component of any problem solving system. The object selection techniques described later rely on a dependency-based modelling mechanism [L1].

Briefly, the dependency-based model makes use of a dependency net, an explicit representation of the justifications for beliefs. In this way, not only is the problem solver's current model of the environment represented, but so are the reasons for supporting belief in that environment. The graph structure of the dependency net allows the efficient determination of effects and causes of alterations to the model. A simple example of a dependency net is depicted in Figure 1, representing the situation in Figure 2.[1]

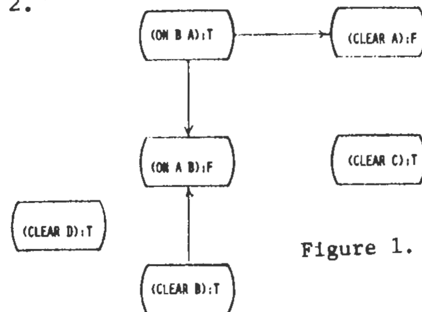
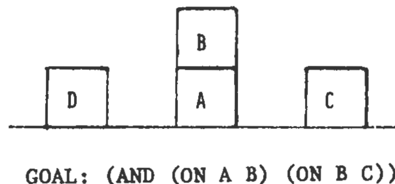


Figure 1.



GOAL: (AND (ON A B) (ON B C))

Figure 2.

[1] See [L1] for more details on the structure and semantics of dependency nets.

A dependency-based model affords several advantages to object selection tasks. A particular advantage is the dependency net's ability to represent explicitly the effects of assumptions. If an assumption (for instance, an assumption of which peg would be best for the top disk in the Tower-of-Hanoi) leads to an excessively high cost solution, responsibility for this difficulty can be efficiently traced to the assumption at fault.[2]

3. Object Selection

The techniques I have developed to deal with the object selection problem are best described via an example. Consider again the simple blocks world situation in Figure 2.[3] If the goals are solved in the order specified (first (ON A B) and then (ON B C)), the problem solver will not realize that B should have been put on C until after A is on B. To achieve (ON A B), both (CLEARTOP A) and (CLEARTOP B) must be true. To achieve (CLEARTOP A), B must be placed out of the way (anywhere but on top of A). A non-optimal solution to this problem of not knowing where to put B would be to include in the CLEARTOP strategy, explicit instructions to place the block on top of the one desired to be cleared at some "freespace" site on the table. This accomplishes (CLEARTOP A), but the possibilty remains that freespace could have been selected as a more optimal location (namely, by putting B on top of C).

A more desirable solution to this problem is achieved by providing a general purpose description mechanism in which are specified features of the object to be selected. Then, the problem solver can make the selection at a time it deems appropriate. For example, a description of freespace for the CLEARTOP strategy might look like

```
(*DESCRIPTOR -freespace
    [OR <AND (CLASS -freespace BLOCK)
             (CLEARTOP -freespace)>
        (REGION-OF -freespace TABLE) ]
    [TABLE-REGION-1]  ).
```

This *descriptor* characterizes freespace as either a block whose top is clear or as a region of the table. Furthermore, if no reasonable binding can be discovered during the planning process, a particular region of the table (TABLE-REGION-1) is chosen as the default.

I have investigated two distinct techniques for manipulating descriptors in general problem solvers. These two object selection techniques are called *deferred descriptor binding* and *assumption propagation*. Deferred binding involves delaying the determination of an appropriate referent for a descriptor until the choice of the object has become reasonably secure. Assumption propagation is a form of backtracking in which a suitable referent for the descriptor is conjectured, and facilities for retracting that choice, if necessary, are provided.

3.1. Deferred Descriptor Binding

The deferred binding technique for object selection is based on the use of a symbolic description of an object in much the same manner as a direct reference to the object. Thus, when the CLEARTOP strategy is invoked in the example above, rather than a commitment to a particular object satisfying the freespace descriptor, the descriptor itself can be referenced in assertions that would normally reference the object. When the action (PUTON B -freespace) is modelled, rather than having effects (CLEARTOP A) and (ON B TABLE-REGION-1), its effects will be (CLEARTOP A) and (ON B -freespace). When the deferred binding technique is being applied, descriptors are similar to the "formal objects" used by Sussman [S2] and Sacerdoti [S1]. Descriptors are treated as variables whose bindings are delayed until one can be found which satisfies the descriptor's features.

The power of deferred descriptor binding can be seen in the example when a solution to the second goal (ON B C) is attempted. Since (1) the world model contains the assertion (ON B -freespace), (2) the current goal can be made true by binding -freespace to C, and (3) the object C satisfies the description specified in the freespace descriptor, the object C becomes a prime candidate for binding to the variable -freespace. There are no other suitable candidates and, therefore, the object C is substituted for all references to -freespace in both the plan and the model. Thus, the original specification of the clearing action (PUTON B -freespace) is modified to (PUTON B C). Both original goals are now

---

[2] See [SS1] for one application of dependency nets to backtracking on incorrect assumptions.

[3] This example is cited as a difficulty for certain problem reduction problem solvers by Waldinger [W1]. In particular, problem solvers which do not replan in response to subgoal protection violations (see e.g. [W1], [W2], [RL1]) generate redundant actions in solving this problem.

achieved and the plan is complete.[4]

3.2. Assumption Propagation

The assumption propagation technique for object selection is a form of backtracking. Assumption propagation involves generating a collection of alternative bindings for a descriptor, and selecting one as a conjecture. If that conjecture results in difficulties for further problem solving, the conjecture can be retracted. The name "assumption propagation" was chosen because the effects of a conjecture are propagated through the dependency-based model much as are other modifications to the problem solving environment.

The dependency-based model allows easy detection of instances in which a selected binding for a descriptor subverts the overall planning effort. The cases for which bad binding assumptions should be retracted include:

1. The subgoal currently being considered by the problem solver is false because of a particular assumption.
2. The effects of an assumption propagate to a protected subgoal, making it untrue.[5]

One advantage of assumption propagation as contained in a dependency modelling framework is that it allows the ultimate sources of problems of the types described above to be determined. The dependency-based model also allows wrong conjectures to be retracted efficiently (see [L1]). Assumption propagation, in particular, the methods for seeking the source of a difficulty and recovering from an incorrect conjecture, uses an algorithm similar to the dependency-directed backtracking technique developed by Stallman and Sussman [SS1].[6]

To illustrate assumption propagation, once again consider the situation in Figure 2. When solving the subgoal (CLEARTOP A), the assumption propagation technique would generate a list of alternative bindings for the freespace descriptor that would include blocks D and C and the table:

(BINDING -freespace (D C TABLE)).

Let us now assume that the system arbitrarily chooses 'D' as the first binding to consider. This assumption would be incorporated into the original dependency net depicted in Figure 1 as illustrated in Figure 3. (The "slashed-through" links represent no longer valid relations.)

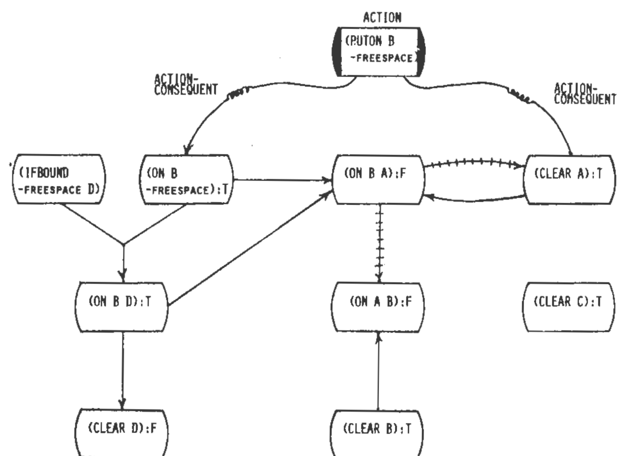


Figure 3.

When the goal (ON B C) is considered, the problem solver will encounter supporting relations for (ON B C):FALSE as depicted in Figure 4. One of the reasons for (ON B C):FALSE is that D was chosen as a binding for freespace. By recovering only along this line of support to the point at which freespace was bound, a new choice of a binding can be made without disrupting the entire model. Eventually, 'C' will be selected, the effect of which will be to make the goal (ON B C) true (and, therefore, will result in an optimal plan).

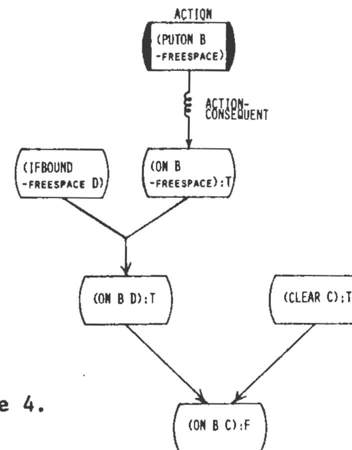


Figure 4.

4. Selection Techniques: Enhancements

So far, I have described, via an example, two distinct methods for performing object selection tasks. These object selection techniques, deferred descriptor binding and assumption propagation, represent skills which must

---

[4] This approach to deferred binding is quite similar to the approach applied to resolving formal object references by Sacerdoti [S1]. In Section 4.1., I describe some enhancements to deferred binding which represent advances over the formal object method.

[5] For discussions of subgoal protection, see [RL1], [W1], and [S2].

[6] A detailed formulation of the dependency-directed backtracking algorithm appears in [D1].

be brought to bear on general problem solving tasks. I now briefly discuss the flexibility with which these techniques can be endowed by describing a few enhancements to the basic algorithms.

4.1. Deferred Descriptor Binding

Deferred descriptor binding provides a problem solver with the ability to treat symbolic descriptions of an object as it would treat the object itself. In this way, the decision as to the best possible binding for a descriptor can be deferred until a reasonably secure choice can be made. Occaisionally, though, what turns out eventually to be the best choice is somewhat counterintuitive! Consider a Tower-of-Hanoi problem stated as:

```
(AND (ON DSK-BIG PEG3)
     (ON DSK-MIDDLE PEG3)
     (ON DSK-SMALL PEG3)).
```

When solving the first goal, the problem solver might be inclined to make what seems to be (from the local evidence available) an obvious choice of PEG2 for a freespace site on which to put DSK-SMALL while clearing DSK-BIG. PEG2 might seem best at first because choosing PEG3 would immediately disenable the action (PUTON DSK-BIG PEG3) which provides the solution to the first goal.

By embedding deferred descriptor binding in a dependency-based model, mistakes like this can be undone by simply invoking the recovery mechanism for assumption propagation. The formal object techniques of Sussman and Sacerdoti require binding deferment until only one possible candidate binding remains, but this reduces the flexibility of the deferment technique. There are instances where it is advantageous to postpone binding only until there are a few possible remaining candidates.

Alternatively, a good deal of flexibility can be gained by augmenting the feature list in a descriptor dynamically during the planning process. This can serve two purposes. First, augmenting the descriptor can help reduce the number of alternative bindings a descriptor might have. Again, using Tower-of-Hanoi as an example, consider the freespace descriptor for the CLEARTOP strategy:

```
(*DESCRIPTOR -freespace
    (AND (CLASS -freespace PEG)
         (LEGAL -dsk -freespace)
         (NEQUAL -freespace
                (*DESCRIPTOR -peg
                     (ON -dsk -peg))) )).
```

Here, freespace is a peg on which it is legal to place the disk which must be moved, while excluding the peg it is already on. If the problem solver is considering what to do with the middle disk, then a freespace descriptor has been instantiated for the top disk. It would be useful to augment the middle disk's freespace descriptor to read:

```
(*DESCRIPTOR -freespace
    (AND (CLASS -freespace PEG)
         (LEGAL -dsk -freespace)
         (NEQUAL -freespace
                (*DESCRIPTOR -peg
                     (ON -dsk -peg)))
         <NEQUAL -freespace D27>  )),
```

where D27 is the instantiated freespace descriptor associated with DSK-SMALL. Thus, freespace for DSK-MIDDLE is described as not only a legal peg not equal to the one DSK-MIDDLE is on, but also as a peg not equal to the one eventually selected as freespace for DSK-SMALL. When a decision is made on a freespace site for one disk, the other now falls into line immediately.

The second way in which descriptor augmentation benefits problem solving is that goal satisfaction can sometimes be simplified at the cost of more complicated object selection. That is, it is possible, in some instances, to assume a goal is true and augment the features of descriptors occurring in the goal statement. Thus, any object later chosen as satisfying the descriptor also satisfies the goal.

4.2. Assumption Propagation

Assumption propagation is a backtracking technique in which conjectures are made about suitable bindings for descriptors. By embedding the assumptions in the dependency-based model, a backtracking technique similar to dependency-directed backtracking [SS1] can be employed. Dependency-directed backtracking makes use of the dependency record to determine quickly the source of a contradiction by tracing to the incorrect assumption that led to it. To apply such a technique in general problem solving, general purpose methods are required which can diagnose the appropriate times to invoke backtracking.[7]

General purpose methods for diagnosing the appropriateness of invoking backtracking make use of the dependency-based model. They include cases in which the source of a subgoal protection violation [RL1] or a subgoal

---

[7] Compare to ARS [SS1], which is an expert problem solving system. ARS uses domain-specific knowledge to determine that a contradiction has been derived and that is is appropriate to reject an assumption.

being false can be traced through the dependency net to a binding assumption for a descriptor. Identification of such general methods for detecting cases in which it is appropriate to retract an object selection conjecture is an important step in generalizing dependency-directed backtracking.

Determining an instance in which backtracking for object selection is appropriate is not as straightforward as determining the appropriateness of backtracking in other problem solving tasks; here, backtracking is not invoked on a failure, but on a lack of success! Because assumption propagation for a particular choice of descriptor binding leads to a false subgoal or a protection violation does not necessarily imply that another possible binding for the responsible descriptor will improve the situation.

Therefore, one enhancement for assumption propagation, in cases where there are only a small number of alternative choices, is to carry along separate assumptions for descriptor bindings in parallel. Parallel assumption propagation allows comparisons among competing descriptor bindings to be made. The dependency net model provides the capabilities for parallel assumption propagation, but, for efficiency, should be applied judiciously. The technique does demonstrate advantages over sequential assumption propagation in cases where, for instance, binary decisions are involved. Since carrying along two distinct bindings for a single variable often results in contradictory beliefs for truth values of some assertions in the model, parallel and sequential assumption propagation can be applied cooperatively with the sequential technique taking over where the parallel approach leads to a contradiction.

5. Conclusion

The techniques I have described represent steps toward a unified theory of object selection and an implementation effort is well underway. I have presented object selection as an important task to be performed by any general problem solver. The techniques presented here display the flexibility and generality required of a formalism on which to base a theory of object selection. The differentiation of object selection into distinct techniques provides a first step toward such a theory.

6. References

[D1] Doyle, Jon. "A Glimpse of Truth Maintenance." MIT AI Memo 461, February 1978.

[L1] London, Phil. "A Dependency-Based Modelling Mechanism for Problem Solving." University of Maryland Computer Science TR 589, Nov. 1977.

[M1] McDermott, Drew V. "Vocabularies for Problem Solver State Descriptions." Proc. IJCAI5 Cambridge, Mass. August 1977.

[N1] Nilsson, Nils J. Problem Solving Methods in Artificial Intelligence. McGraw-Hill, 1971.

[R1] Rieger, Chuck. "One System for Two Tasks: A Commonsense Algorithm Memory that Solves Problems and Comprehends Language." Artificial Intelligence, January, 1976.

[RL1] Rieger, Chuck and Phil London. "Subgoal Protection and Unravelling During Plan Synthesis." Proc. IJCAI5, Cambridge, Mass., August 1977.

[S1] Sacerdoti, Earl D. "A Structure for Plans and Behavior." SRI AI TN #109, August 1975.

[SS1] Stallman, Richard M. and Gerald Jay Sussman. "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer Aided Circuit Analysis." Artificial Intelligence, Vol. 9, No. 2, (October 1977), pp. 135-196.

[S2] Sussman, Gerald J. "A Computer Model of Skill Acquisition." MIT AI-TR 297, Aug. 1973.

[W1] Waldinger, Richard. "Achieving Several Goals Simultaneously." SRI AI TN #107, July, 1975.

[W2] Warren, David. "WARPLAN: A System for Generating Plans." Memo No. 76, Department of Computational Logic, University of Edinburgh, June 1974.

Acknowledgements - I wish to thank Chuck Rieger, Milt Grinberg, Rich Wood, Steve Small, and Alan Thompson for many interesting discussions and help in preparing this document. I also wish to thank the National Aeronautics and Space Administration for support of this research under Grant NSG-7253.

EXPERIMENTAL CASE STUDIES OF BACKTRACK VS. WALTZ-TYPE VS. NEW ALGORITHMS FOR SATISFICING ASSIGNMENT PROBLEMS*

John Gaschnig
Artificial Intelligence Center
SRI International, Inc.
333 Ravenswood Dr.
Menlo Park, CA 94025

SUMMARY

Mackworth [1977] claims that Waltz-type network consistency algorithms are "clearly better than automatic backtracking", but he cites no experimental data comparing tne performances of the algorithms under identical conditions. Here we report the results of a set of performance measurement experiments comparing these two algorithms with two other algorithms, BKMARK (Gaschnig [1977]) and a new algorithm BKJUMP. Each of the algorithms is valid for a broadly and precisely defined class of satisficing assignment problems (SAPs) that includes numerous familiar problems. The results span four functionallyequivalent algorithms, three performance measures, two solutioncriteria, and four sample sets of SAPs, and the results represent more than 13,000 distinct algorithm executions. The four sample sets of SAPs include two sets of "N-Queens" problems (for N up to 50) and two quite different types of randomly generated problems. We describe the simple set-theoretic mathematical model underlying our experimental approach.

The results show algorithm BKMARK the most efficient of the four algorithms in all cases, and the Waltz-type algorithm least efficient in almost all cases. To give an indication of speed, BKMARK finds solutions to the 50-Queens problem (search space size $\sim 10^{84}$) at the rate of one per 9 cpu-seconds on a PDP/KL-10. One "random SAPs" experiment compares the performance of the algorithms in solving N-Queens problems to the corresponding performances in solving a set of "random-N-Queens" problems, which are identical in size and "degree of constraint" (L) to the N-Queens problems. The other "random SAPs" experiment compares algorithm performances on a set of problems identical in size but varying in degree of constraint. This shows the dependence of performance on L, all other things equal. These data show in particular that the Waltz-type algorithm does not better the others on highly constrained problems.

These first controlled comparisons of the four algorithms illustrate the value of voluminous hard data in resolving speculations and in uncovering previously unsuspected phenomena. We make no claims about the performances of these algorithms except for the cases tested here, but propose additional experiments to provide evidence on which to base such claims.

> All life is an experiment. The more experiments you do the better.
> Ralph Waldo Emerson

1. Background and General Issues

The preceding summary gives some indication of the scope and methodology of the research reported here. In this section we give some background on the subject and discuss briefly a few methodological issues.

Any instance of a certain sort of satisficing (i.e., non-optimizing) assignment problem (SAP, defined formally in section 2) can be solved using the so-called backtrack search algorithm, as defined in general form by Golomb & Baumert [1965], but little is known in general about the computational requirements of this algorithm. Believing the backtrack algorithm to be inefficient, Waltz [1972, 1974] and other artificial intelligence researchers have devised an alternative general algorithm for SAPs, but claims about its efficiency have been based on skimpy hard data. Mackworth [1977] surveys reports by Sussman & McDermott [1972] and Gaschnig [1974] documenting the inefficiencies of backtrack in specific instances, and Mackworth also surveys the generalizations of Waltz' algorithm given by Gaschnig [1974], Rosenfeld, et al., [1976], and others. Mackworth claims of the backtrack algorithm that "the time taken to find a solution tends to be exponential in the number of variables, both in worst case and in average case" [1977, p. 100], and that Waltz-type algorithms are "clearly better than automatic backtracking" [1977, p. 116].

Here we put such general, informally stated claims to the test of hard data, comparing algorithms under identical conditions. Our premise is that detailed quantitative performance data over many cases can provide a firmer basis than unsubstantiated speculation for predicting performance in novel circumstances. As Knuth [1975, p.121] points out, inability to predict algorithm behavior in novel circumstances can be a major impediment to its widespread application.

We attempt to be completely rigorous in defining terms and the conditions of the experiments, so that all of the results reported

* This research was done at the Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213, and was supported by the Defense Advance Research Projects Agency under Contract no. F44620-73-C-0074 and monitored by the Air Force Office of Scientific Research.

here are completely replicatable and hence are objectively verifiable. The experimental observations are estimates of mathematically well-defined quantities. To make explicit the amount of evidence reported here, we count the number of distinct algorithm executions represented in each figure. Note that our technical objective is simply to obtain the performance values plotted in the figures. Careful and rigorous quantitative analysis of the experimental values is beyond the scope of the present paper. Consequently, we eschew here attempts to "explain" the data. Hence we provide a body of quantitative data against which to test future speculations and mathematical theories.

In some senses, the context of our results is both narrow and broad: It is narrow in the sense that the problems considered satisfy precise mathematical conditions, they include only satisficing problems (as opposed to optimizing), and they have only binary constraint relations (as opposed to ternary or r-ary). Nor do we examine the probabilistic Waltz-type method of Rosenfeld [1976]. Nevertheless, this context is broad in that a large class of problems satisfy these conditions, and in that each of the algorithms we investigate can be applied to any of these problems.

Mostly, we are concerned with the number of steps to find any solution as opposed to all solutions.

2. Definitions and Examples

To insure that our experimental results are meaningful and replicatable, in this section we give precise definitions for problem, algorithm, performance measure, and conditions of the experiment. For clarity, we supplement the formal descriptions with examples and illustrations.

Our formalism for problems, algorithms, and performance measures is essentially an extension of that of Gaschnig [1974] and Mackworth [1977, pp. 99-100]. Unlike Mackworth, however, we find it more useful to define the P_{ij} constraints as relations rather than as predicates (to facilitate the definitions of section 7).

DEFINITION 5.1. A <u>satisficing assignment problem</u> (SAP) is a tuple $\{N, R_1, R_2, \ldots, R_N, P_{12}, P_{13}, \ldots, P_{1N}, \ldots, P_{N-1,N}\}$ such that:

N is a positive integer (denoting the number of <u>problem variables</u> $x_1, x_2, \ldots, x_N$)

$R_1, R_2, \ldots, R_N$ are arbitrary finite sets. (Associated with each problem variable x_i is a specified set of a priori possible <u>candidate values</u> $R_i = \{v_{i1}, v_{i2}, \ldots, v_{ik_i}\}$.)

An <u>assignment</u> $A = (y_1, y_2, \ldots, y_N)$ of candidate values to problem variables is an element of the cross product $U = R_1 x R_2 x \ldots x R_N$ (i.e., so $y_i \in R_i$), and A(i) denotes the i'th component of A, for $1 \leq i \leq N$.

For every i and j such that $0<i<j\leq N$, $P_{ij} \subseteq R_{ij}$. Also $P_{ij} = P_{ji}$.

An assignment $A \in U$ is a <u>solution</u> iff for every i and j such that $0<i<j\leq N$, $(A(i),A(j)) \in P_{ij}$.

The "8 Queens" problem is a well-known SAP in which 8 queens must be placed on a chess board so that no two queens can take each other. In this problem N = 8, the problem variable x_i corresponds to the i'th queen, and the candidate values of each queen consist of the a priori legal squares on which that queen can be placed. Since in any solution the queens must occupy distinct rows of the board, we take R_i to consist of the 8 squares in row i (k_i=8, for $1<i\leq 8$). For symmetry reasons, however, R_1 may be restricted to the leftmost four squares of row 1 (i.e., $k_1 = 4$). In predicate form, P_{ij}(A(i), A(j)) is satisfied for assignment A if queen i on square A(i) does not attack queen j on square A(j).

The "N-Queens" problem is a generalization of the 8-Queens problem: place N queens on an NxN chess board so that no two queens attack each other. (In our formulation, $k_1 = \lceil N/2 \rceil$ and $k_2 = k_3 = \ldots k_N = N$.) Other problems that can be formulated as SAPs include map coloring (e.g., see Nijenhuis & Wilf [75, pp. 181-183]), labeling in a particular way each of the line segments in a two-dimensional projection of a scene of polyhedra [Waltz 1972, 1974], finding Euler circuits or Hamiltonian circuits or spanning trees of a graph [Nijenhuis & Wilf 1975], cryptarithmetic [Simon 1969, Gaschnig 1974], the Instant Insanity puzzle [Knuth 1975], the SOMA cube puzzle [Fillmore & Williamson 1974, p. 51], and space planning problems [Eastman 1972]. Other examples are cited in [Golomb & Baumert 1965] and in [Mackworth 1977].

The version of the backtrack algorithm used in these experiments, called BKTRAK, is defined subsequently in this section, and also in Gaschnig 1977]. As in the latter, we define as an elemental unit of computation an inquiry by the algorithm to determine whether $(x,y) \in P_{ij}$, where $x \in R_i$ and $y \in R_j$. (In predicate form, the unit is an execution of P_{ij}(A(i), A(j)).) We call such an inquiry a <u>pair-test</u>, and identify a pair-test formally by a 4-tuple (i,x,j,y).

The following incomplete trace of the BKTRAK algorithm applied to the 8-queens problem illustrates the inefficiencies of the algorithm, and provides a basis for illustrating the behavior of algorithms BKMARK (Gaschnig [1977]) and BKJUMP. The numerous distinct partial instantiations of problem variables form a tree, as depicted below. Each occurrence of "T" and "F" in the trace indicates the outcome of a single pair-test. For

example, the entry "6,4 TTTF" in portion "A" indicates that the pair-tests with arguments (6,4,1,1), (6,4,2,1), (6,4,3,5), and (6,4,4,2) returned the values True, True, True, and False, respectively. Since a queen on square (6,4) attacks a queen on square (4,2), this instantiation of problem variable 6 fails, and hence PAIRTEST(6,4,5,4) is not executed.

Incomplete trace of BKTRAK for 8-Queens SAP:
x,y = queen x on square at row x, column y

```
1,1
     2,1  F
     2,2  F
     2,3  T
         3,1  F
         3,2  TF
         3,3  F
         3,4  TF
         3,5  TT
             4,1  F
             4,2  TTT
                 5,1  F
                 5,2  TTTF
                 5,3  TF
                 5,4  TTTT
                     6,1  F       \
                     6,2  TTF      |
                     6,3  TF       |
                     6,4  TTTF     |   A
                     6,5  TTF      |
                     6,6  F        |
                     6,7  TF       |
                     6,8  TTF     /
                 5,5  F           \
                 5,6  TF           |   C
                 5,7  TTF          |
                 5,8  TTTT        /
                     6,1  F       \
                     6,2  TTF      |
                     6,3  TF       |
                     6,4  TTTF     |   B
                     6,5  TTF      |
                     6,6  F        |
                     6,7  TF       |
                     6,8  TTF     /
             4,3  TF
             4,4  F
                 .
                 .
                 .
```

Inspection of the trace above illustrates the basic ideas underlying algorithms BKMARK and BKJUMP. All of the pair-tests in section B of the trace are unnecessary: they were executed in section A; between A and B only the assignment of queen 5 has changed; but in section A no c.v. of queen 6 was "pair-tested" against queen 5, since each c.v. failed when pair-tested against the assigned c.v.s of the first four queens; hence the outcome in section B is necessarily identical to that in section A. Algorithm BKMARK eliminates these redundant pair-tests in a more general way than is illustrated by this example.

Algorithm BKJUMP (defined in section 4) capitalizes on a different effect. Since none of the c.v.s of queen 6 in section A passes the pair-tests against the assigned c.v.s of queens 1, 2, 3, and 4, it is necessarily the case that no assignment having the latter c.v.s as its first four elements can be a solution. Hence the pair-tests in section B and C are unnecessary, and one can backtrack from section A two levels instead of the customary one level and proceed directly with the pair-tests following section C. BKJUMP achieves this effect in a general way.

As seen above, algorithms for SAPs may be redundant in the sense that some pair-tests may be executed more than once. Accordingly, we define *

T = the total number of pair-tests executed by an algorithm B for a SAP Z

D = the number of distinct pair-tests executed under the same conditions

M = T / D

So M = 1 indicates that all pair-tests executed are distinct, i.e., none are recomputed. In general, $M \geq 1$, and indicates the average number of times each distinct pair-test is executed during a given search. To illustrate our performance measures, for the portion of the execution traced T = 77, D = 58, and M = 77/58 = 1.33.

To provide standards against which to compare the performances of the algorithms, we plot in the figures the values of T_{min}, D_{max}, and SAS, defined as follows. The minimum number of pair-tests executed by any of the algorithms considered here is dependent only on N, the number of problem variables. This is achieved if the assignment consisting of the first candidate value of each problem variable happens to be a solution. In this case, each candidate value of the assignment is "pair-tested" against every other, for a total of $T_{min}(N) = N\,(N-1)/2 = O(N^2)$ pair-tests for any SAP having N problem variables. Note that T_{min} equals the number of pair-tests required to verify that a given assignment is a solution if in fact it is.

For a given SAP, the total number of possible distinct pair-tests is determined by the values of N and the k_i, thus:

$$D_{max}(N, k_1, \ldots, k_N) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} k_i * k_j \quad \text{(for SAPs in general)}$$

$$= (N-1)\, N \lceil N/2 \rceil + N^2 \,(N-1)(N-2)/2 = O(N^4) \quad \text{(for N-Queens SAPs)}$$

A measure of the size of the search space is the total number of distinct possible assignments (SAS is mnemonic for "size of assignment space"):

$$SAS(N, k_1, \ldots, k_N) = \prod_{i=1}^{N} k_i \quad \text{(for SAPs in general)}$$

---

* Note: the performance measure called M here was called D in Gaschnig [1977].

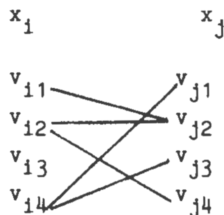
$$= O(N^N)$$

(for N-Queens SAPs)

For sake of comparison, the values of $T_{min}(N)$, $D_{max}(N)$, and SAS(N) for N-Queens SAPs are plotted as a function of N in Figure 2-1 and in subsequent figures. T_f, D_f, and M_f denote values observed when the solution criteria is to find any solution (i.e., a first solution); T_a, D_a, and M_a similarly denote values observed when the criteria is to find all solutions.

The Waltz-type algorithm used in these experiments, called DEEB, is essentially identical to algorithm CS2 defined in Gaschnig [1974]. DEEB* combines backtracking with a procedure, called DEE, of the generic form of "arc-consistency" algorithm that Mackworth calls AC-3 [1977] and Gaschnig [1974] calls CS-1. Mackworth [1977, p. 114] suggested certain modifications to algorithm CS-1 with the intent of improving its efficiency. DEE is a functionally equivalent variation of CS-1 that achieves the efficiencies suggested by Mackworth and eliminates other unnecessary pair-tests as well, so that DEE is strictly more efficient than AC-3, as we shall now show informally.

For brevity, we assume that the reader is familiar with Mackworth's argument and notation. The following hypothetical example illustrates informally the differences between the approach of AC-3 (i.e., to distinguish arc (i,j) from (j,i)) and the approach of DEE (i.e., to process a P_{ij} relation "as a whole"). The diagram below depicts the constraint relation P_{ij} as a set of links between the candidate values of two problem variables x_i and x_j. Hypothetically, x_i and x_j could be problem variables of a SAP having other problem variables as well.



In the case depicted above, CS-1 executes the equivalent of Mackworth's function REVISE((i,j)), which executes 2 pair-tests (p.t.) to determine that v_{i1} is supported by v_{j2}, and then 2 p.t. to establish support for v_{i2}, then 4 p.t. to determine that v_{i3} is not supported by x_j and hence can be eliminated, followed by 1 p.t. for v_{i4}, for a total of 9 p.t. CS-1 then executes REVISE((j,i)), determining at a cost of 11 p.t. that all c.v.s of x_j are supported. Mackworth correctly points out that CS-1's execution of REVISE((j,i)) is often superfluous, because the execution of REVISE((i,j)) cannot cause arc (j,i) to become "arc-inconsistent" if it is not already. Therein lies the rub: since AC-3 initially puts all arcs (i,j) and their complements (j,i) on the queue Q, AC-3 executes each REVISE((i,j)) and REVISE(j,i)) at least once and for these executions AC-3 executes unnecessary pair-tests that are not executed by DEE.

DEE executes a single procedure REVISEBOTH((i,j)) that has the effect of first doing a REVISE((i,j)), but at the same time marking those c.v.s of x_j that provide support to the c.v.s of x_i. REVISEBOTH then executes the equivalent of a REVISE((j,i)) modified so that only unmarked c.v.s of x_j are checked for support by x_i. Hence in the above example REVISEBOTH((i,j)) executes only 9 p.t., since all c.v.s of x_j are marked. Generalizing, in precisely the cases that the REVISE((j,i)) of CS-1 is superfluous due to the conditions described by Mackworth, in these same cases all c.v.s of x_j are marked, and hence REVISEBOTH((i,j)) executes exactly those p.t. executed by REVISE((i,j)). Hence DEE using REVISEBOTH executes no more p.t. than AC-3 using REVISE. Since DEE executes fewer p.t. than AC-3 for the first executions of REVISE((i,j)) and REVISE((j,i)), it follows that DEE executes strictly fewer pair-tests than AC-3 for all SAPs except the degenerate cases of SAPs that are arc-consistent initially.

Orthogonal to the issues just discussed, AC-3 maintains a queue of pending arcs (i,j) to REVISE, whereas CS-1 uses a triangular matrix for the same purpose, but without the FIFO discipline of the queue. DEE could use either priority policy, but in fact uses the triangular matrix mechanism of CS-1 (See Gaschnig [1974] for details.)

3. Elementary Comparison results

Now, having defined problems, performance measures and algorithms, we can compare BKTRAK and DEEB by objective criteria. As evidence against which to test Mackworth's claim that Waltz-type algorithms are "clearly better" than the backtrack algorithm, Figure 3-1 compares BKTRAK with DEEB by $T_f(N)$ for N-Queens SAPs, N = 4, 5,..., 17. These data do not support the claim, but it is risky to extrapolate to larger values of N, or to other problems. Sections 5, 6, and 7 provide more extensive comparative data comparing T_f, D_f, and M_f.

To show the relation between total number of pair-tests executed and the number of distinct pair-tests executed, both for the case of finding one solution and of finding all solutions, Figure 3-2 plots $T_f(N)$, $D_f(N)$, $T_a(N)$ and $D_a(N)$ for BKTRAK. The total number of solutions to the N-Queens puzzles, for N = 4,5,6,7,8,9,10 is 1, 6, 2, 23, 46,

* "DEEB" is mnemonic for "Domain Element Elimination with Backtracking", which would seem to describe the behavior of this type of algorithm more accurately than the term "network consistency algorithm" proposed by Mackworth [1977].

203, 362, respectively. The following tabulation compares BKTRAK with DEEB by $T_a(N)$:

N	4	5	6	7	8	9
BKTRAK	42	236	1008	5345	23376	136807
DEEB	78	379	1032	4218	14118	68239

Figure 3-3 plots the redundancy ratios $M_f(N)$ and $M_a(N)$ based on the data in Figure 3-2. These data show that the redundancy of BKTRAK grows sharply with increasing size of the problem. Again, sections 5, 6, and 7 report much more extensive data. Perusal of such $M_f(N)$ data in fact motivated an attempt to define a backtrack-like algorithm (namely BKMARK) that eliminates much of this redundancy. Here then was a case in which performance measurement experiments yielded insights that yielded a new algorithm. Note that three performance measures were involved, one (T) the product of the other two (D and M).

Figure 3-4 introduces a new variation, that of randomizing the ordering of the candidate values of each problem variable, in the manner of Gaschnig [1977]. In the experiments of Figures 3-1, 3-2, and 3-3 the candidate squares for each queen are ordered from left to right, which we call the "obvious" candidate value (c.v.) ordering. In fact, for each queen (i.e. problem variable) $k_i!$ distinct orderings are possible, giving a total of $\prod_{i=1}^{N} k_i!$ distinct c.v. orderings. For each value of N, we selected m(N) c.v. orderings from among this set, where m(N) = 30 for $4 \leq N < 8$; m(N) = 70 for $8 \leq N < 15$; m(N) = 100 for $N \geq 15$. So Figure 3-4 depicts the results of m(4) + m(5) + ... + m(15) + m(16) = 810 distinct algorithm executions.

Note that $T_f(N)$ using "obvious" c.v. ordering is generally much larger than the mean value of $T_f(N)$ using random c.v. ordering. The ratios of these values for each N and for each algorithm are given in section 5. The vertical bars for mean $T_f(N)$ indicate an interval of two sample standard deviations of the sample mean -- a standard statistical measure of how closely the observed mean approximates the true mean. The mean $T_f(N)$ curve in Figure 3-4 is taken from Figure 1 in Gaschnig [1977]. Algorithms that randomize their inputs are of some inherent interest in analysis of algorithms research (e.g., see Weide [1977, p. 304]). Subsequently, T_f denotes a mean value unless otherwise specified.

4. Definition of BKJUMP

We define now a new general backtrack-type algorithm for SAPs, called BKJUMP, that sometimes backtracks across multiple levels of the search tree instead of across only a single level. BKJUMP is the result of an attempt to produce the domain-element-elimination effect of DEE in a backtrack-like control context. DEE eliminates candidate values when it detects a global inconsistency; BKJUMP does so in the context of candidate values already instantiated higher in the search tree.

The following SAIL procedure defines algorithm BKJUMP, in a form that halts after finding a first solution. (SAIL is a variant of ALGOL; see Swinehart [1971].) In the code, pairtest is an external procedure implementing the problem-specific P_{ij} constraint relations. Procedure BKTRAK is defined the same, minus the underlined portions, except that the statement "return(returndepth)" in BKJUMP is replaced by "return(0)" in BKTRAK. This code for BKJUMP indicates visually that it is short in length, and that it is very similar to the code for BKTRAK. Note that BKJUMP uses only two local variables beyond those used by BKTRAK. The invocation conditions for BKJUMP are identical to those given for BKMARK in Gaschnig [1977] and for brevity are not repeated here. BKJUMP (and BKTRAK as defined here) returns -1, with solution in array A, or returns 0 if no solution exists. For brevity, "!" stands for "comment" below; "<-" denotes the assignment operator.

```
recursive integer procedure bkjump(integer var, n;
                                    integer array a, k);
begin
integer i, val, returndepth, faildepth;
boolean testflg;
returndepth <- 0;
for val <- 1 step 1 until k[var] do
                              ! see comment C1 below;
   begin
   testflg <- true;
   for i <- 1 step 1 while i < var and testflg do
                                              ! See C2;
      testflg <- pairtest(i, a[i], var, val);
   if not testflg then faildepth <- i - 1;     ! C3;
   if testflg then                             ! C4;
      begin
      a[var] <- val;
      if var = n then return(-1)               ! C5;
      else
         begin
         faildepth <- bkjump(var+1, n, a, k);
         if faildepth < var then return(faildepth)
                                               ! C6;
         end
      end;
   returndepth <- returndepth max faildepth
   end;
return(returndepth)                            ! C7;
end;
```

Comments for above code:
- C1: check each candidate value (c.v.) of the var'th problem variable.
- C2: test this c.v. against each instantiated c.v.
- C3: note: uses final value of loop variable i.
- C4: if passed all tests, then ....
- C5: solution found, so unwind to outermost call.
- C6: unwind to level given by value of faildepth.
- C7: backtrack and continue search.

We claim without formal proof that BKJUMP is

functionally equivalent to BKTRAK and to BKMARK in the sense that for any SAP the solution assignments found by the three algorithms (if any exist) are identical. We further claim that every distinct pair-test executed by BKJUMP is also executed by BKTRAK (and by BKMARK), and that each such pair-test is executed at least as many times by BKTRAK as by BKJUMP.

5. Results for N-Queens SAPs with random c.v. ordering

Figure 5-1 compares the mean number of pair-tests (i.e., mean $T_f(N)$) executed by algorithms BKTRAK, BKMARK, BKJUMP, and DEEB, respectively, to find a first solution for N-Queens SAPs. The sample set of SAPs over which these measurements are taken is the same for each algorithm, namely the set described in section 3 of m(N) randomly selected candidate value orderings for N = 4, 5,..., 16. (The values constituting the curves labeled "BKTRAK" and "BKMARK" in figure 5-1 are taken from figure 1 in Gaschnig [1977]. The curve labelled "BKTRAK" is identical to the one labelled "$T_f(N)$ (mean)" in Figure 3-4 in this report.)

We observe in Figure 5-1 that among the four algorithms being compared, BKMARK executes the fewest pair-tests on the average, followed in order by BKJUMP, BKTRAK, and DEEB, and that this ordering among the four algorithms is observed to hold for each value of N. Note also that the performance of BKJUMP differs very little from that of BKTRAK, indicating (we speculate) that in this case the average number of levels jumped over is approximately one. Note also that $T_f(N)$ for BKMARK is much less than for the other three algorithms, but also much greater than $T_{min}(N)$. Note also that none of the four curves is closely approximated by a straight line in this semilog plot, as would be the case if $T_f(N)$ grew exponentially as Mackworth suggests. Note also the relatively poor performance of algorithm DEEB: we have identified one set of SAPs for which DEEB is less efficient than BKTRAK by the mean $T_f(N)$ measure for each value of N observed, and much less efficient than BKMARK under the same conditions. The factors to which this inefficiency can be attributed remain uncertain at present (but see sections 6 and 7 for additional data).

To provide further evidence for larger values of N, we extended the previous experiment to the cases N = 20, 25, 30, 35, 40,and 50, with m(N) = 50 samples per value of N, and we measured mean $T_f(N)$ for BKMARK only. Combining these data with some of those for BKMARK in figure 5-1, the observed values for N = 5, 10, 15, 20, 25, 30, 35, 40, 50 are 23.6, 542, 1513, 2696, 4715, 11520, 28415, 21890, 55020, respectively. Approximating the latter values by the formula $T_f(N) = N^{C(N)}$ and solving for C(N) we obtain the formula $C(N) = \log T_f(N) / \log N$. For the above list of values of N and $T_f(N)$, the values of C(N) are 1.964, 2.734, 2.704, 2.637, 2.628, 2.750, 2.884, 2.709, 2.79 respectively. Note that with the exception of N = 5, these C(N) values fall in the interval 2.75 ± 0.14. Note that our purpose in presenting this approximation is simply pragmatic: to show how well a particular approximation fits the observed data, without suggesting that the approximation is valid generally. Pragmatically, the data and approximation would seem to cast doubt on the proposition that mean $T_f(N)$ grows exponentially with N in this case.

Figure 5-2 plots the ratios of the value of $T_{f|ALG}(N)$ using "obvious" candidate value ordering to the corresponding observed value of mean $T_{f|ALG}(N)$ using random c.v. ordering, where $ALG \in$ {BKTRAK, BKMARK, BKJUMP, DEEB}. The ratio values plotted in figure 5-2 indicate apparently that differences in performance between random c.v. ordering and "obvious" c.v. ordering are exhibited by each of the four algorithms, and that these differences are generally much larger for $14 \leq N \leq 16$ than for N < 14.

Figure 5-3 plots the corresponding mean values of the redundancy ratio $M_f(N)$ collected during the same experiments. Due to page limitations, we omit plotting the corresponding $D_f(N)$ data, which show DEEB to execute more distinct pair-tests than the other algorithms for all values of N tested, considerably more for large values of N.

6. Results for "Random-N-Queens" SAPs

To generalize the results beyond N-Queens SAPs, in this section we define a parameterized equivalence relation on the set of all possible SAPs, partitioning this set into (disjoint) equivalence classes, such that members of a given equivalence class have identical values of certain parameters representing size and "degree of constraint" of a SAP. We then define a procedure for selecting SAPs randomly (independently, uniformly, with replacement) from among the members of a specified equivalence class. We use this procedure to generate randomly for each N a set of SAPs each of whose size and degree of constraint parameters matches that of the N-Queens SAP to which it corresponds (one parameter set per value of N).

DEFINITION 4-1. SAPs Z = $\{N, R_1, R_2, \ldots, R_N, P_{12}, P_{13}, \ldots, P_{N-1,N}\}$ and Z' = $\{N', R'_1, R'_2, \ldots, R'_{N'}, P'_{12}, P'_{13}, \ldots, P'_{N'-1,N'}\}$ are <u>N-similar</u> iff N = N'. SAPS Z and Z' as defined above are <u>N-k_i-similar</u> iff Z and Z' are N-similar and $k_i = k'_i$ (recall $k_i = |R_i|$ and $k'_i = |R'_i|$), for each i = 1,2,..,N.

For example, let the "8-Queens-Knights" SAP be defined like the 8-Queens SAP except that the "chess pieces" in the former move either as queens

or as knights. Then the 8-Queens SAP and the 8-Queens-Knights SAP are N-k_i-similar.

We define the "degree of constraint" of a SAP to be the fraction of distinct pair-tests for that SAP that map to the value "true". Formally, given a SAP Z,

$$L_Z = F_Z / D_{max}$$

where $F_Z = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} |P_{ij}|$

So $0 \leq L \leq 1$ by definition. To illustrate, the value of P_{ij} in the case of the diagram given in section 2 concerning DEE is 5, equal to the number of links, and L = 5/16 for these two problem variables.

DEFINITION 4-2. SAPs Z and Z' as defined in definition 4-1 are <u>N-k_i-L-similar</u> iff Z and Z' are N-k_i-similar and $L_Z = L_{Z'}$.

We use the following procedure for randomly selecting a SAP having specified values of N, k_1, k_2,..., k_N, and L. For each i and j such that $1 \leq i < j \leq N$, we create a boolean-valued matrix U_{ij} of size $k_i \times k_j$. To each element of each such matrix we assign (by means of pseudo-random number generator) the value "true" with probability L, and the value "false" with probability 1 - L. (Note that using this procedure the percentage of matrix elements assigned the value "true" does not necessarily equal exactly the given value of L, but rather approximates it. For the present cases, it turns out that the difference is negligible. This is assured by the law of large numbers.)

By exhaustively enumerating the set of distinct pair-tests and counting the number of those that map to the value "true", the values of L for the N-queens problems for N = 4, 5,..., 16 are determined to be (to 3 decimal places) 0.444, 0.552, 0.622, 0.676, 0.714, 0.746, 0.770, 0.791, 0.808, 0.823, 0.835, 0.846, 0.856, respectively. (When plotted against N, these values give the appearance of a smooth curve. For brevity, we show no such plot in this report.) The sample set of what we shall call "random-N-queens" SAPs consists of 50 independently and randomly generated SAPs having $N = k_2 = k_3 = k_4 = 4$, $k_1 = 2$, and L = .444 (i.e., the values for the 4-Queens SAP); a similar set of 50 SAPs for each of N = 5,6,7; 100 such samples for N = 8,9,10,11,12; 150 samples for N = 13; and 250 samples for N = 14. These SAPs are N-k_i-L similar to the corresponding N-Queens SAPs.

Figure 6-1 shows the mean values of $T_f(N)$ observed using algorithms BKTRAK, BKMARK, BKJUMP, and DEEB to find first solution for the SAPs in the random-N-queens sample set. Comparing these data with those in Figure 5-1, note that the relative ordering of the algorithms is the same in both figures: BKMARK executes the fewest pair-tests on the average for each value of N, followed in order by BKJUMP, BKTRAK, and DEEB.

To more easily compare the values shown in Figure 6-1 with the corresponding values in Figure 5-1, Figure 6-2 plots the ratio of each value plotted in the latter figure to its corresponding value in the former figure. The values so plotted represent the results of 7340 distinct algorithm executions. We observe that the differences between $T_f(N)$ using DEEB and $T_f(N)$ using BKTRAK are larger for random-N-queens SAPs than for the corresponding N-queens SAPs, and that the magnitude of this difference grows with N, and that the same holds for the corresponding differences between the performance of BKTRAK and that of BKJUMP. Other observations about the observed values plotted in Figures 6-1 and 6-2 are summarized in section 8 under point 4. Analogous results assuming $M_f(N)$ as the performance measure instead of $T_f(N)$ are not plotted here (for brevity), but show less extreme difference differences between N-Queens SAPs and "random-N-Queens" SAPs. (The ratio is less than 2 for all values of N tested.)

7. Results for ISVL SAPs

Next we report the results of an experiment designed to show how the cost of solving a SAP depends on the degree of constraint possessed by the problem. From only the results plotted in the figures of the preceeding sections, it is difficult to infer the dependence of the mean value of $T_f(N)$ on the value of L, because the SAPs in the N-queens sample set differ among each other both in size (i.e., N and k_i values) and in L values, and the same holds for the random-N-queens sample set. Moreover, L ranges only from 0.444 to 0.856 among the N-Queens SAPs in our sample set.

Accordingly, we performed experiments using as sample set a set of randomly generated SAPs that are identical to each other in size but differ systematically in value of L. We used the method of the previous section to generate randomly 150 SAPs, each having N = 10, $k_1 = k_2 = \ldots = k_{10} = 10$ and link percentage value L = 0.1; iterating, we generate randomly in similar fashion a set of 150 distinct SAPs for each of L = 0.2, 0.3,..., 0.9. For these values of N and the k_i, $T_{min} = 45$, $D_{max} = 4500$, and SAS = 10^{10}.

The four curves plotted in Figure 7-1 show the mean values of $T_f(L)$ observed when each of the algorithms BKTRAK, BKMARK, BKJUMP, and DEEB, respectively, is applied to the SAPs in this "Identical size, varying L" (ISVL) sample set. Figures 7-2 and 7-3 show the corresponding mean values of $D_f(L)$ and $M_f(L)$ observed for the ISVL sample set.

The T_f values plotted in Figure 7-1 for the boundary cases L = 0.0 and L = 1.0 are derived analytically rather than observed experimentally. The values plotted are $T_f = k_1 \cdot k_2 = 100$ at L=0.0 for

all four algorithms, and $T_f = N(N-1)/2=45$ at L=1.0 for BKTRAK, BKMARK, and BKJUMP, and T_f=1305 at L=1.0 for DEEB. The value for DEEB is large because it first applies DEE, determining that the SAP is arc-consistent (see Mackworth [1977]), then instantiates problem variable x_1 and again applies DEE, and continues in like manner until all N problem variables are instantiated. Several observations about the values plotted in Figures 7-1, 7-2, and 7-3 are summarized in section 8 under point 5.

> Add little to little and there will be a big pile.
>
> Ovid

8. Conclusions and Future Work

1) In all observed cases of N-Queens SAPs, algorithm BKMARK executes fewer pair-tests ($T_f(N)$) than do the other three algorithms under identical conditions, in some cases fewer by a factor of 10. BKMARK is observed to be more optimal than the other three algorithms with respect to the M_f search redundancy measure.

2) In almost all observed cases of N-Queens SAPs, the Waltz-type algorithm DEEB executes more pair-tests on the average than do the other three algorithms under identical conditions.

3) For N-Queens SAPs and each algorithm, we observe that randomizing the ordering of candidate values of each problem variable in a certain uniform way before commencing the search generally causes fewer pair-tests to be executed than if a certain "obvious" c.v. ordering is used, fewer by as much as a factor of 500 in one case observed (N=20 in Figure 5-2).

4) Conclusions 1 and 2 above are further supported by analogous data for "random-N-Queens" SAPs. Comparison of these "random-N-queens" data with the "actual-N-Queens" data shows that these two sample sets of SAPs are sharply more distinguishable for $N \geq 10$ than for $N < 10$, and are sharply less distinguishable by algorithm DEEB than by the other three algorithms. Note in particular that for $N \geq 10$, N-Queens SAPs require many <u>more</u> pair-tests to be executed on the average than is the case for the corresponding random-N-Queens SAPs. Hence we have demonstrated how algorithm performance experiments can aid in defining what is meant by the "structure" exhibited by a problem.

5) Results reported for the "identical size, varying degree of constraint" (ISVL) SAPs also support conclusions 1 and 2. Furthermore, the results indicate that mean T_f depends strongly on L, spanning a range whose extremes differ by a factor of 791. The data suggest the existence of a single sharp peak in $T_f(L)$ at L ~ 0.6. Analogous data for $D_f(L)$ and $M_f(L)$ show similar peaks and range of performance.

Given additional computer time, it is straightforward to extend each of the present experiments to other values of the experiment parameters, and to other problems. In particular, it would be interesting to obtain comparable results using Waltz' line drawing problem or map coloring. In these problems many of the problem variables do not constrain each other directly (i.e., $P_{ij} = R_i x R_j$, the universal relation), whereas in the present results each problem variable (e.g., queen) constrains each other problem variable. Perhaps DEEB performs relatively better and BKTRAK relatively worse on these problems. In addition to experiments, it is important to obtain analogous results analytically, to the extent this is possible, and to compare the analytic predictions with the experimental observations.

Note: The numeric values plotted in the figures constitute the actual "results" of the experiments, and yet they are not tabulated in this report (due to page limitations). The author is pleased to supply to anyone who so requests tables listing all of the numbers plotted in the figures, including as well sample standard deviation values and maximum and minimum values. These tables will also appear in the author's forthcoming Ph.D. thesis.

References

Bobrow, D., and B. Raphael, New programming languages for AI research, Computing Surveys, 6 (1974), 153-174.

Eastman, C., Preliminary report on a system for general space planning, CACM, Vol. 15, No. 2, February 1972.

Fillmore, J., and S. Williamson, On backtracking: A combinatorial description of the algorithm, SIAM J. Computing (3), No. 1 (March 1974), 41-55.

Gaschnig, J., A constraint satisfaction method for inference making, Proc. 12th Annual Allerton Conf. Circuit and System Theory, U. Ill. Urbana-Champaign, Oct. 2-4, 1974.

Gaschnig, J., A general backtrack algorithm that eliminates most redundant tests, Proc. Intl. Joint Conf. Artificial Intelligence, Cambridge, MA, 1977, p. 457.

Golomb, S. and L. Baumert, Backtrack programming, J.A.C.M. (12), No. 4 (Oct. 1965), 516-524.

Knuth, D., Estimating the efficiency of backtrack programs, Mathematics of Computation (29), No. 129 (Jan. 1975), 121-136.

Mackworth, A. K., Consistency in networks of relations, Artificial Intelligence (8), No. 1, 1977, 99-118.

Nijenhuis, A., and Wilf, H.S., Combinatorial Algorithms, Academic Press, New York 1975.

Rosenfeld, A., R. Hummel and S. Zucker, Scene labelling by relaxation operations, IEEE Trans. on Systems, Man, and Cybernetics SMC-6 (1976), 420-433.

Simon, H.A., Sciences of the Artificial, MIT Press, Cambridge, MA. 1969.

Sussman, G.J., and McDermott, D.V., Why conniving is better than planning, Artificial Intelligence Memo. No. 255A, MIT (1972).

Swinehart, D., and B. Sproull, SAIL, Stanford AI Project Operating Note No. 57.2, January 1971.

Waltz, D.E., Generating semantic descriptions from drawings of scenes with shadows, MAC-AI-TR-271, MIT 1972.

Waltz, D., Understanding line drawings of scenes with shadows, in P. Winston (ed.) The Psychology of Computer Vision, McGraw-Hill Book Co., New York 1975, 19-91.

Weide, B., A survey of analysis techniques for discrete algorithms, Computing Surveys 9, no. 4 (December 1977).

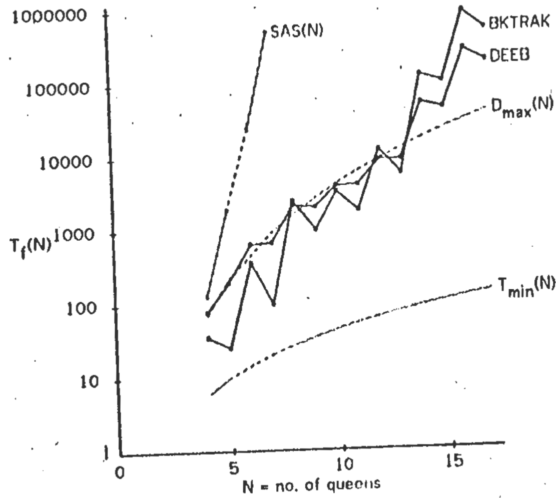


Figure 3-1 $T_f(N)$ = No. of pair-tests to solve N-Queens puzzle (to find first solution)
algorithm BKTRAK vs. Waltz-type algorithm DEEB
1 algorithm execution per plotted point (solid curves)

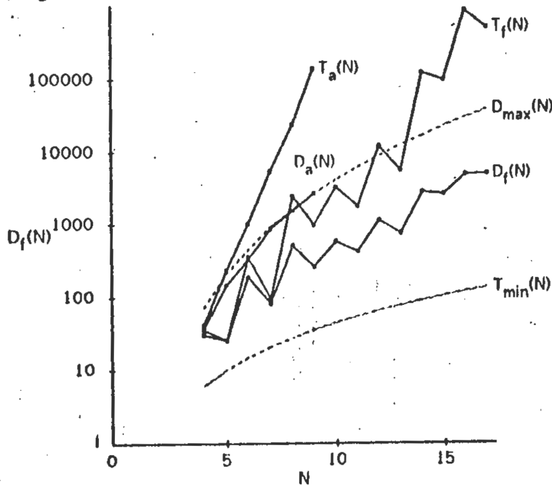


Figure 3-2 No. of pair-tests (T) and no. of distinct pair-tests (D) to find first solution (T_f, D_f) and all solutions (T_a, D_a)
N-Queens, BKTRAK; 1 algorithm execution for every plotted point
$T_f(N)$ values same as those plotted in figure 3-1

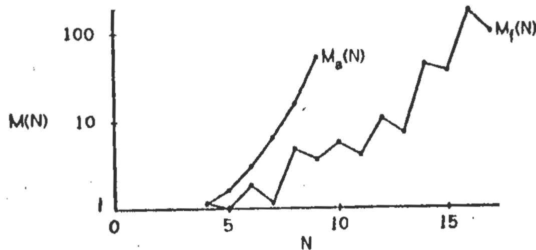


Figure 3-3 Redundancy ratio M(N) = T(N) / D(N)
= Total no. of pair-tests executed / no. of distinct pair-tests executed
N-Queens, algorithms BKTRAK, first solution and all solutions
T(N) and D(N) values in computation of M(N) are those in Figure 3-2

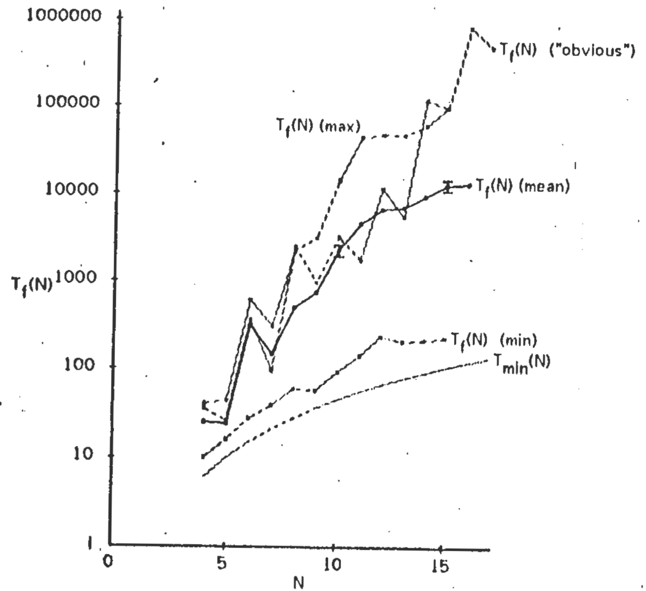


Figure 3-4 $T_f(N)$: mean, max and min values over m(N) samples of random candidate value ordering, compared with $T_f(N)$ for "obvious" c.v. ordering
N-Queens, algorithm BKTRAK, first solution
m(N) algorithm executions for each value of N; 30 ≤ m(N) ≤ 100 (see text)
810 algorithm executions (a.e.) total

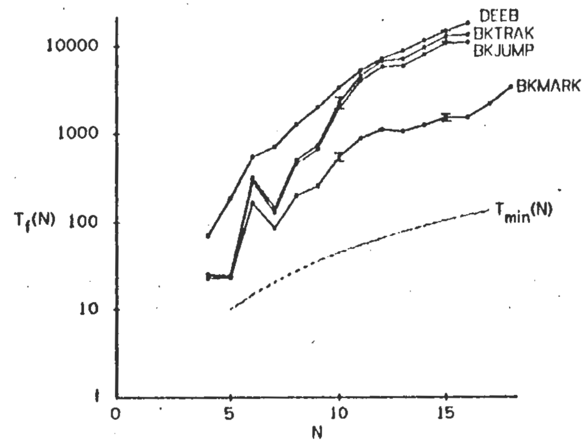


Figure 5-1 Comparison of algorithm performances by mean number of pair-tests
N-queens, first solution, random candidate value ordering
Same sample set for each algorithm. 810-1010 a.e. per algorithm, 3440 a.e. total
mean $T_f(N)$ values for BKTRAK are those in Figure 3-4

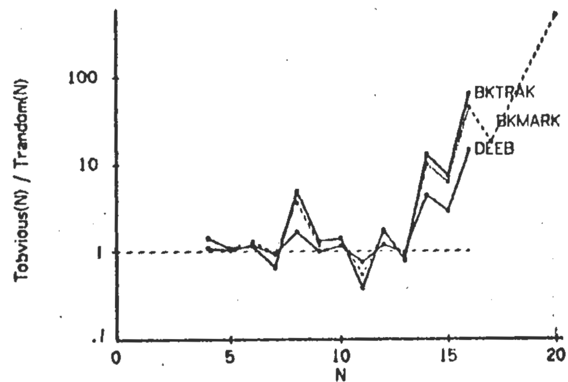


Figure 5-2 Ratio of $T_f(N)$ with "obvious" c.v. ordering to mean $T_f(N)$ with random c.v. ordering
N-Queens, first solution, random c.v. ordering
same set of algorithm executions as those for Figure 5-1
(Values for BKJUMP ~ values for BKTRAK)

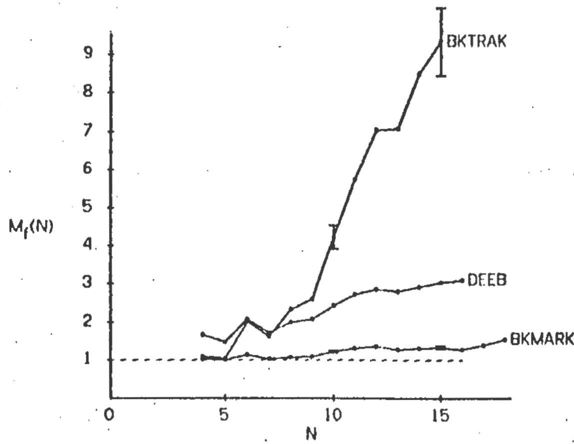


Figure 5-3 Algorithm comparison by mean redundancy ratio
$M_f(N) = T_f(N) / D_f(N)$
N-Queens, first solution, random c.v. ordering
Same set of algorithm executions as in Figure 5-1

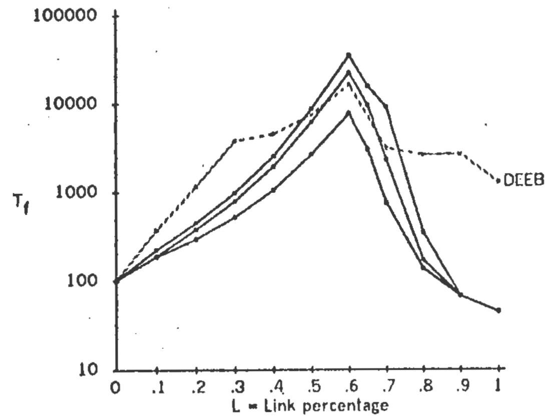


Figure 7-1 Dependence of no. of pair-tests (T_f) on degree of constraint (L)
150 randomly generated SAPs of size $N = k_i = 10$ for each plotted point
1350 a.e. per algorithm, 5400 a.e. total
upper unmarked curve: BKTRAK; middle: BKJUMP; lower: BKMARK
first solution

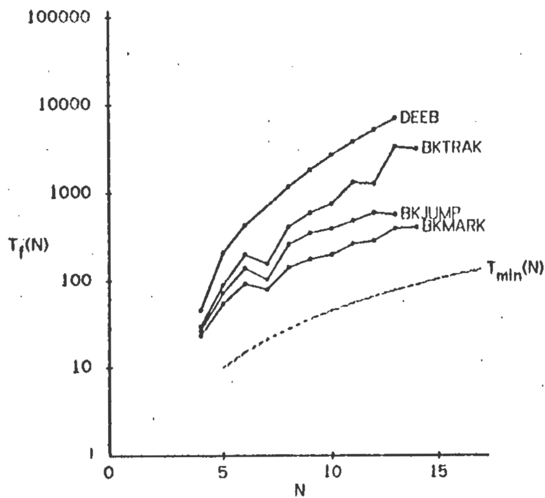


Figure 6-1 Analogous to Figure 5-1, but using sample set of randomly generated SAPs having same size and degree of constraint as N-Queens SAPs
first solution, random c.v. ordering
50-250 a.e. for each (N, algorithm) pair;
850-1100 a.e. total per algorithm; 3900 a.e. total

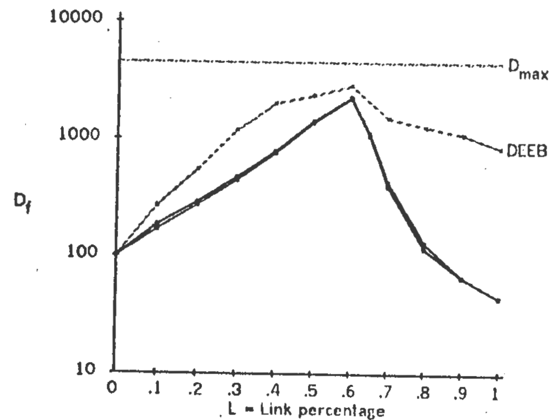


Figure 7-2 Dependence of mean no. of distinct pair-tests (D_f) on L
Same set of algorithm executions as in Figure 7-1
upper unmarked curve: BKTRAK & BKMARK; lower curve: BKJUMP
(lower and upper unmarked curves almost identical values)
first solution

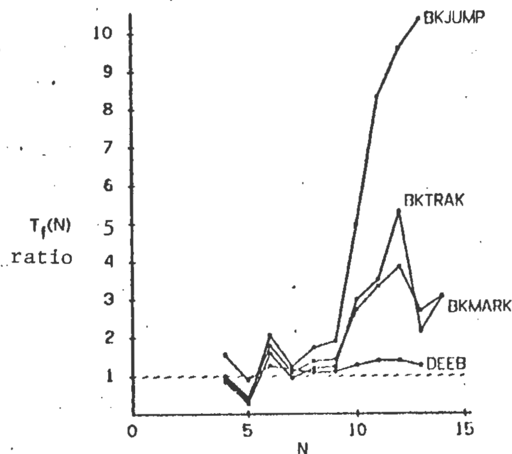


Figure 6-2 Ratio of $T_f(N)$ for N-Queens to $T_f(N)$ for "Random-N-Queens" SAPs
Data from Figures 5-1 and 6-1, respectively. 3440 + 3900 = 7340 a.e. total
Experimental data by which to distinguish "natural" SAPs from parametrically similar i.i.d.-random SAPs

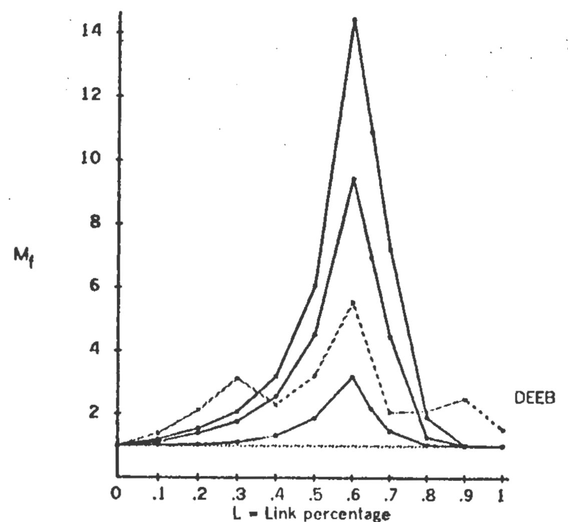


Figure 7-3 Dependence of mean redundancy ratio (M_f) on L
Same set of algorithm executions as in Figure 7-1
upper unmarked curve: BKTRAK; middle: BKJUMP; lower: BKMARK
first solution

DISTRIBUTED PROBLEM SOLVING: THE CONTRACT NET APPROACH

Reid G. Smith and Randall Davis[1]

Heuristic Programming Project
Department of Computer Science
Stanford University
Stanford, California, 94305.

Abstract

We describe a problem solver based on a group of processor nodes which cooperate to solve problems. In a departure from earlier systems, we view task distribution as an interactive process, a discussion carried on between a node with a task to be executed and a group of nodes that may be able to execute the task. This leads to the use of a control formalism based on a contract metaphor, in which task distribution corresponds to contract negotiation.

We also consider the kinds of knowledge that are used in such a problem solver, the way that the knowledge is indexed within an individual node, and distributed among the group of nodes. We suggest two primary methods of indexing the knowledge (referred to as "task-centered" and "knowledge-source centered"), and show how both methods can be useful.

We illustrate the kind of information that must be passed between nodes in the distributed processor in order to carry out task and data distribution. We suggest that a common internode language is required, and that task-specific "expertise" required by a processor node can be obtained by internode transfer of procedures and data.

We consider the operation of a distributed sensor net as an instantiation of the issues we raise.

Finally, the approach presented here is compared with those taken by the designers of earlier systems, such as **PLANNER, HEARSAY-II,** and **PUP6**.

1 Introduction

The ongoing revolution in LSI technology is drastically reducing the cost of computer components, making multiple processor architectures economically viable. These architectures have the potential to provide several computational advantages over uniprocessor architectures, including speed, reliability, and efficient matching of available processing power to problem complexity [**Baer, 1973**]. This has led to a search for problem solving methods which can exploit the new technology. In this paper we present one approach to problem solving in such architectures.

We propose a model of a ***distributed problem solver*** which consists of a collection of processors connected with communications and control mechanisms that enable them to operate concurrently, and enable them to cooperate in solving complex problems. We use the term "distributed" rather than "parallel" to emphasize that the individual processors are ***loosely-coupled***; that is, the time a processor node spends in communication is small with respect to the time it spends in computation.

Loosely-coupled systems are desirable for a number of reasons. First, such systems are highly modular, and hence offer considerable conceptual clarity and simplicity in their organization. Second, and equally important from our perspective, systems designed to be loosely-coupled require less communication by an individual node. This is an important practical consideration because a major problem that arises in the design of multiple processor architectures is interconnection of the nodes [**Anderson, 1975**]. Complete interconnection (so that a node can communicate directly over a private channel to every other node) is extremely expensive because it entails a number of channels proportional to the square of the number of nodes. One way to reduce this expense is to employ a single broadcast communications channel which is shared by all nodes.

Unfortunately, such a channel can be a major source of contention and delay when the number of processor nodes is large. The communications medium connecting the nodes is thus a valuable (and limited) resource that must be conserved if a large number of processor nodes is to function together effectively. It is thus desirable to reduce the amount of message traffic, and designing the system so it is loosely-coupled is one way to accomplish this goal. Loose-coupling can in turn be effected by careful partitioning of the top-level problem to insure that individual processor nodes work on tasks that are relatively independent of each other, and that require processing times which are large with respect to the time required for internode communication.[2]

1.1 A Human Model

The operation of a problem solver working in a distributed processor architecture may be likened to the operation of a group of human experts experienced at working together to complete a large task.[3] In such a situation we might see each expert spending most of his time working alone on various subtasks that have been partitioned from the main task, pausing occasionally to interact with other members of the group in specific, well-defined ways. When he encounters a subtask too large to handle alone, he further partitions it into manageable (sub)subtasks and makes them known to the group. Similarly, if he encounters a subtask for which he has no expertise, he attempts to pass it on to another more appropriate expert. In this case, the expert may know another expert (or several other experts) in the group who have the necessary expertise, and may notify him (them) directly. If the expert does not know anyone in particular who may be able to assist him, or if the new subtask requires no special expertise, then he can simply describe the subtask to the entire group. If some other expert chooses to carry out the subtask, then that expert will request further details from the original expert, and the two may engage in further direct communication for the duration of the subtask. The two experts will have formed their own subgroup, and similar subgroups of variable size will form and break up dynamically during the course of the work on the problem. Subgroups of this type offer two advantages. First, communication among the subgroup members does not needlessly distract the entire group of experts. Such distraction may be a major source of difficulty in large groups (see, for example [**Brooks, 1975**]). In addition, the subgroup members may be able to communicate with each other in a language that is more efficient for their purposes than the language in use by the group as a whole.

It is worthy of note that among the tasks which can be posed by an expert in the group are those that involve a transfer of "expertise" from one expert to another; that is, one expert may request instruction in the execution of a particular task.

In our human model, no one expert is in control of the

[1] This work has been 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. It has been carried out on the SUMEX-AIM Computer Facility, supported by the National Institutes of Health under grant RR-00785. Reid Smith is supported by the Department of National Defence of Canada, and Randall Davis is supported by the National Science Foundation and a Chaim Weizmann Postdoctoral Grant for Scientific Research.

[2] Partitioning of this kind is, of course, a well-known problem solving strategy, often referred to as "divide and conquer".

[3] The group of experts model has also been used as a starting point by [**Lenat, 1975**] and [**Hewitt, 1977a**], but has resulted in approaches with different characteristics than that considered in this paper. We compare the different approaches in Section 5.

others, (although one expert may be ultimately responsible for communicating the solution of the top-level problem to the "customer" outside the group). As a result, one of the major problems facing such a group is integration of information held by the individual members. The group members must find ways to share and build on one another's information, and find ways to examine and resolve differences in order to reach a consensus.

2 Problem Solving Protocols

We now consider the design of a problem solver that can exploit the characteristics of a distributed processor architecture. In doing this, we make a rough correspondence between human experts and individual processor nodes, but our aim is the design of an effective problem solver, not a simulation of human performance. The question is then, "What techniques will supply the requisite communications and control mechanisms?" We will see that one of the necessary mechanisms is a ***problem solving protocol*** designed to enable the individual nodes to communicate for the purpose of cooperative problem solving. It is based on the more traditional notion of ***communications protocol***.

The use of communications protocols in networks of resource-sharing computers, such as the ARPAnet, is by now quite familiar [Kahn, 1972]. These protocols have as their primary function reliable and efficient communication between computers. The layers of protocol in the ARPAnet, for example, serve to connect IMP's to IMP's (the subnet communications devices), hosts to hosts (the processor nodes of the network), and processes executing in the various hosts to other such processes [Crocker, 1972].

Communications protocols are, however, only a start - a prerequisite for distributed problem solving. We need to build upon the work of network and communications protocol designers to focus on *what* to say in the context of distributed problem solving, as opposed to *how* to say it. In ARPAnet terms, we must move above the process-to-process protocol to add yet another layer - one concerned with the management of tasks.

2.1 Design Goals

Before presenting the specific protocol to be used throughout the remainder of this paper, we review the general design goals for a problem solving protocol.

First, we are concerned with the communication of messages between the nodes of a distributed problem solver. We must therefore insure that our protocol is sufficiently general that it allows the communication of a broad class of information, and allows interactions capable of supporting complex problem solving behavior.

Second, the protocol must be well-suited to systems that are loosely-coupled. As noted earlier, it is important to minimize communication since communications channel capacity is expensive. While careful task partitioning has the greatest potential impact on the amount of internode communication required, the problem solving protocol also plays a role. Therefore, the protocol should be efficient in terms of its use of communications resources (i.e., terse).[4]

The protocol should also foster distribution of control and data in order to insure that advantage can be taken of potential gains in speed and reliability that may be achieved through the use of multiple processors. Centralized control could create an artificial bottleneck (slowing the system down), and could make it difficult for the system to recover from failure of critical components.

Finally, the protocol should aid in maintaining the ***focus*** of the problem solver, to combat the combinatorial explosion which besets almost all AI programs. For a uniprocessor, focus involves selection at each instant in time of the most appropriate task to be executed [Hayes-Roth, 1977]. For a distributed processor, focus can be reformulated as finding the most appropriate tasks to be executed and matching them with processor nodes appropriate for their execution.

In a uniprocessor, focus of attention is generally handled by a single, global, heuristic evaluation function used to rank order all tasks in the system (see, for example [Lenat, 1976]). In a distributed processor, however, each individual processor node has its own local evaluation function. Task selection and decisions are thus based on local considerations, and this locality gives rise to the problem of inducing global coherence in the actions of the individual processor nodes. Since this can be a major source of difficulty, the problem solving protocol should also offer some assistance in overcoming it.

2.2 The Contract Net

These considerations lead to the notion of task distribution as an interactive process, one which entails a discussion between a node with a task to be executed and nodes that may possibly be able to execute the task. The contract net approach to distributed problem solving [Smith, 1977] uses an announcement - bid - award sequence of ***contract negotiation*** to effect this matching. We present a simplified description of the approach in this section.

A ***contract net*** is a collection of interconnected processor nodes whose interactions are governed by a problem solving protocol based on the contract metaphor. Each processor node in the net operates asynchronously and with relative autonomy. Instances of the execution of individual tasks are dealt with as ***contracts***. A node that generates a task advertises existence of that task to the other nodes in the net as a ***task announcement***, then acts as the ***manager*** of that task for its duration. In the absence of any information about the specific capabilities of the other nodes in the net, the manager is forced to issue a ***general broadcast*** to all nodes. If, however, the manager possesses some knowledge about which of the other nodes in the net are likely candidates, then it can issue a ***limited broadcast***. Finally, if the manager knows exactly which of the other nodes is appropriate, then it can issue a ***point-to-point*** announcement.[5] As work on the problem progresses, many such task announcements may be made by various managers.

The other nodes in the net have been listening to the task announcements, and have been evaluating their own level of interest in each task with respect to their specialized hardware and software resources. When a task is found to be of sufficient interest, a node may submit a ***bid***. Each bid indicates the capabilities of the bidder that are relevant to execution of the announced task. A manager may receive several such bids in response to a single task announcement; based on the information in the bids, it select one (or several) node(s) for execution of the task. The selection is communicated to the successful bidder(s) through an ***award*** message. These selected nodes assume responsibility for execution of the task, and each is called a ***contractor*** for that task.

A contract is thus an *agreement* between a node that generates a task (the manager) and a node that executes the task (the contractor). Note that establishing a contract is a process of mutual selection. Available processor nodes evaluate task announcements made by several managers until they find one of interest; the managers then evaluate the bids received from potential contractors and select one they determine to be most appropriate. Both parties to the agreement have evaluated the information supplied by the other and a mutual decision has been made.

The contract negotiation process is expedited by three forms of information contained in a task announcement. An ***eligibility specification*** lists the criteria that a node must meet to be eligible to submit a bid. This specification reduces message traffic by pruning nodes whose bids would be clearly unacceptable. A ***task abstraction*** is a brief description of the task to be executed, and allows a potential contractor to evaluate its level of interest in executing this task relative to others that are available. An abstraction is used rather than a complete description in order to reduce message traffic.

---

[4] Current evidence [Galbraith, 1974] suggests that effective human organizations operate in an analogous manner, minimizing unnecessary communications among the members.

[5] Restricting the set of addressees (which we call ***focused addressing***) of an announcement is typically a heuristic process, since the information upon which it is based may not be exact (e.g., it may be inferred from prior responses to announcements).

Finally, a ***bid specification*** details the expected form of a bid for that task. It enables a potential contractor to transmit a bid which contains only a brief specification of its capabilities that are relevant to the task (called a ***node abstraction***), rather than a complete description. This both simplifies the task of the manager in evaluating bids, and further reduces message traffic.[6]

The normal contract negotiation process may be simplified in two instances. First, a ***directed contract*** does away with the announcement and bid, and is awarded directly to a selected node. Second, a request - response sequence is used without further embellishment for tasks which amount to simple requests for information. These two simplifications serve to enhance the efficiency of the protocol.

It is important to note that individual nodes are not designated a priori as managers or contractors. Any node can take on either role, and during the course of problem solving a particular node normally takes on both roles (perhaps even simultaneously for different contracts).

In addition to effecting task distribution, a contract between two nodes serves to set the context for their communication. Setting up such a context facilitates their communication. A contract is also of assistance in forming subgroups of nodes. As in the human model discussed above, such subgroups can communicate among themselves without distracting the entire group. Furthermore, an established context permits the use of a specialized language for their communication. This helps to reduce message traffic.

The award message contains a ***task description***, which includes the complete specification of the task to be executed. After the task has been completed, the contractor sends a ***report*** to its manager. This message includes a ***result description***, which contains the results that have been achieved during execution of the task.

The manager may ***terminate*** contracts as necessary, and ***subcontracts*** may be let in turn as required by the size of a contract or by a requirement for special expertise or data that the contractor does not have.

Contracting distributes control throughout the network, helping to create a flexible system; that is, a number of different (potentially dynamic) approaches to problem solving can be implemented. Distributed control and two-way links between managers and contractors also enhance system reliability, in that they enable recovery from individual component failure. The failure of a contractor, for example, is not fatal, since its manager can re-announce the appropriate contract and recover from the failure. This strategy allows the system to recover from any node failure except that of the node that holds the original top-level problem.[7]

While the contract net protocol is a general problem solving protocol, it has been designed so it can be pruned to meet the specific requirements of the application at hand, and hence reduce message traffic and message processing overhead. In its simplest form, it reduces to a standard communications protocol, sending messages between specified sources and destinations. At a slightly more general level, broadcasting of tasks and results is possible, thus effecting a more implicit form of addressing. At progressively more general levels, complex bidding and award mechanisms are added. The contract net can thus be a useful approach to distributed problem solving at many different levels of complexity.

We can now consider how well the contract net protocol meets the design goals specified earlier for a problem solving protocol.

The protocol is well suited to loosely-coupled systems in two respects. First, it provides a very general form of guidance in determining appropriate partitioning of problems: the notion of tasks executed under contracts is appropriate for a grain size larger than that typically used in problem solving systems. (Section 5.1 contains further discussion of this issue.) Second, the protocol is efficient with respect to its use of communications channels. The information in task announcements, for instance, helps minimize the amount of channel capacity consumed by communications overhead. Such efficiency helps to preserve whatever loose-coupling character is already present in the system as a result of problem partitioning.

The use of autonomous contract nodes interacting through a process of contract negotiation fosters distribution of control and data throughout the system, thus meeting the third design goal.

Maintenance of focus is perhaps the most difficult of the design goals to meet, and we do not yet have a good understanding of the underlying issues involved. Our approach is to attack the problem explicitly through "appropriate" definition of the functions used to evaluate task announcements and bids. In addition, each node maintains a list of the "best" recent task announcements it has seen - a kind of window on the tasks at hand for the net as a whole. This window enables the evaluation functions to "integrate" the local situation over time to assist in maintenance of focus.

It is of interest to note that the focus problem does not necessarily have to be attacked explicitly. Some problems lend themselves to a relaxation style of problem solving. Low level vision operations, for example, are suitable candidates for this approach [Zucker, 1977]. The nature of the relaxation process itself tends to produce global coherence from the actions of individual processes even though focus is not addressed explicitly as a problem. In this approach, a lack of appropriate global coherence shows up as oscillation in the relaxation process.

3 Example - Distributed Sensing

In this section, we demonstrate the use of the contract net approach in the solution of a problem in area surveillance, such as might be encountered in ship or air traffic control. The example will help to demonstrate the ideas which form the central foci of the remainder of this paper: (a) task distribution as an interactive process, and (b) indexing and distribution of knowledge.

We consider the operation of a network of nodes, each of which may have either sensing or processing capabilities, and which are spread throughout a relatively large geographic area. Such a network is called a ***Distributed Sensing System (DSS)***. The primary aim of the system is rapid, reliable, accurate, and low-cost analysis of the traffic in a designated area. This analysis involves detection, classification, and tracking of vehicles; that is, the solution to the problem is a dynamic map of traffic in the area which shows vehicle locations, classifications, courses and speeds. Construction and maintenance of such a map requires integration and interpretation of a large quantity of sensory information received by the collection of sensor elements.

There are a number of tradeoffs involved in the design of a DSS architecture, and we present only one possible approach. The primary intent of the example is to act as a vehicle for demonstration of the contract net approach to distributed problem solving.[8]

The example we present here is a hand simulation, but is based on a working SAIL simulation of the contract net that has been applied to a related distributing sensing problem.

3.1 Hardware

The DSS is organized as a contract net that is monitored by a distinguished processor node called the ***monitor node***. All communication in the net is assumed to take place over a broadcast channel. The nodes are assumed to be in fixed positions known to themselves but not known a priori to the monitor node, and they may have two different kinds of

---

[6] We discuss the encoding of this information in Section 4.3.

[7] At the top level, contracting can distribute control "almost" completely, hence removing the bottlenecks that centralized controllers may create. There still remains, however, the reliability problem inherent in having only a single node responsible for the top-level problem. Since this cannot be handled directly by the manager-contractor links, standard sorts of redundancy are required.

[8] Further discussion of the background issues inherent in DSS design is presented in [Smith, 1978a]; a more detailed discussion of this example is presented in [Smith, 1978b], which includes examination of several of the design options and tradeoffs that can only be mentioned briefly here due to space limitations.

capability: sensing and processing. The sensing capability includes low level signal analysis and feature extraction. We assume that a variety of sensor types exist in the DSS, that the sensors are widely spaced, and that there is some overlap in sensor area coverage.

Nodes with processing capability supply the computational power necessary to effect the high level analysis and control in the net. They are not necessarily near the sensors whose data they process. These nodes are able to acquire (if necessary) the procedures essential to effect any of the information processing functions, by transfer from other nodes.

3.2 Data And Task Hierarchy

The DSS must integrate a large quantity of signal data, reducing it and transforming it into a symbolic form meaningful and useful to a human observer. We view this process as occurring in several stages, which together form a data hierarchy (Figure 3.1). The hierarchy offers an overview of DSS function and suggests a task partitioning suitable for a contract net approach.

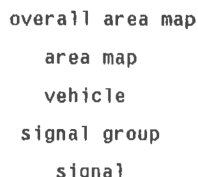


Figure 3.1. Data Hierarchy.

For purposes of this example, the only form of signal processing we consider is narrow band spectral analysis, and the ***signal*** has the following features: frequency, time of detection, strength, characteristics (e.g. increasing signal strength), name and position of the detecting node, and the name, type, and orientation of the detecting sensor.

Signals are formed into ***signal groups*** at the second level of the data hierarchy. A signal group is a collection of related signals. For this example, the signal groups have the following features: the fundamental frequency of the group, the time of group formation, and the features of the signals in the group (as above).

The next level of the hierarchy is the ***vehicle***. It has one or more signal groups associated with it, and is described in terms of position, speed, course, and type. Position can be established by triangulation, using matching groups detected by several sensors with different positions and orientations. Speed and course must be established over time by tracking.

The next level of the data hierarchy is the ***area map***. This map incorporates information about the known vehicle traffic in an area. It is an integration of the vehicle level data. There will be several such maps for the DSS, corresponding to areas in the span of coverage of the net.

The final level is the complete or ***overall area map***. In this example, the map is integrated by the monitor node.[9]

The hierarchy of tasks follows directly from the data hierarchy. The monitor node manages several ***area*** contractors (Figure 3.2). These contractors are responsible for formation of traffic maps in areas defined by the monitor node. Each area contractor in turn manages several ***group*** contractors that provide it with signal groups for its area (Figure 3.3). Each group contractor integrates raw signal data from ***signal*** contractors that have sensing capabilities.

The area contractors also manage several ***vehicle*** contractors that are responsible for integration of information associated with individual vehicles. Each of these contractors manages a ***classification*** contractor that determines vehicle type, a ***localization*** contractor that determines vehicle position, and a ***tracking*** contractor that tracks the vehicle as it passes through the area.[10]

[9] A DSS may have several functions, and not all of these functions will require integration of overall area data at a single node.

[10] In a real solution to the DSS problem, it is possible that not all of these tasks would be large enough to justify the overhead of contracting; that is, some of them might be done in a single node. It is also of interest to note that some of the tasks in the hierarchy are *continuing* tasks (e.g., the area task), while others are *one-time* tasks (e.g., the localize task).

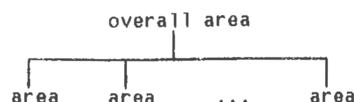


Figure 3.2. Traffic Map Partitioning.

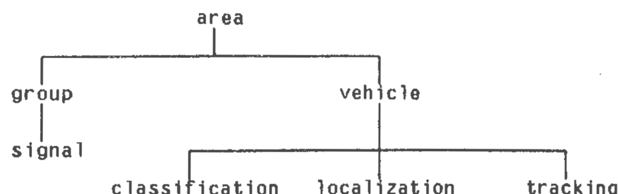


Figure 3.3. Area Task Partitioning.

3.3 DSS Initialization And Operation - The Contract Net Approach

This section reviews in qualitative terms how the DSS problem can be attacked using the contract net approach, and illustrates several of the ideas central to its operation. Appendix A gives specific examples of the message traffic that is described here in more general terms.

3.4 Initialization

The monitor node is responsible for initialization of the DSS and for formation of the overall map. It must first partition the overall span of coverage of the system into areas, and select other nodes to be area contractors. For purposes of illustration we assume that the monitor node knows the names of nodes that are potential area contractors, but must establish their positions in order to partition the overall span of coverage. Hence, it begins by announcing contracts for formation of area maps of the traffic. Because the monitor node knows the names of potential area contractors, it can avoid a general broadcast and use a focused addressing scheme. The announcement contains the three components described in Section 2.2, a task abstraction, eligibility specification, and bid specification. The bid specification is of primary interest for this task. It informs a prospective area contractor to respond with its position. The monitor node uses the positions returned in bids on the task to form appropriate areas and select a subset of the bidders to be area contractors. Each contractor is informed of its area of responsibility in the contract award message.[11]

The area contractors now attempt to solicit other nodes to provide signal group data. In the absence of any information about which nodes might be suitable, each area contractor announces the task using a general broadcast. The eligibility specification in these announcements indicates the area for which the individual area contractor is responsible; that is, a node is only eligible to bid on this task if it is in the same area as the announcing area contractor. Potential group contractors respond with their respective positions, and based on this information, the area contractors award group contracts to nodes in their areas of responsibility.

At this point, the group contractors attempt to find

[11] The full announcement - bid - award sequence is necessary (rather than a directed contract) because the monitor node needs to know the positions of all of the potential area contractors in order to partition the overall span of coverage of the DSS into manageable areas. Note that this means that the DSS will automatically adjust to a change in the number or position of potential area contractors.

nodes that will provide raw signal data. This is done with signal task announcements. The task abstraction in these announcements indicates both the area of responsibility of an individual group contractor and its position. This position information will assist potential signal contractors in determining the group contractors to which they should respond. The eligibility specification in the announcements ensures that a bidder is located in the same area as the announcer, and that it has sensing capabilities.

The potential signal contractors listen to the task announcements from the various group contractors. They respond to the nearest group contractor with a bid that supplies their position and a description of their sensors. The group contractors use this information to select a set of bidders that covers the vicinity with a suitable variety of sensors, and then award signal contracts on this basis. The awards specify the sensors that each signal contractor is to use to provide raw data to its managing group contractor.

The signal contract is a good example of the contract negotiation process, illustrating that the matching of contractors to managers is an interactive process. It involves a mutual decision based on local processing by both the group contractors and the potential signal contractors. The potential signal contractors base their decision on a distance metric and respond to the closest manager. The group contractors use the distribution of sensor types and numbers observed in the bids to select a set of signal contractors which ensures that every area is covered by every kind of sensor. Thus each party to the contract evaluates the proposals made by the other, using a different evaluation function, and arriving at a task distribution agreement via mutual selection.

3.5 Operation

We now consider the activities of the system as it commences operation.

When a signal is detected, or a change occurs in the features of a known signal, the detecting signal contractor reports this fact to its manager (a group contractor). This node in turn attempts to integrate the information into an existing signal group or to form a new signal group.

A group contractor reports the existence of a new signal group to its manager (an area contractor) which must then decide what to do with it. Whenever a new group is detected, the managing area contractor attempts to find a node to execute a vehicle contract. The task of a vehicle contractor is to classify, localize, and track the vehicle associated with the signal group. Since a newly detected signal group may be attributable to a known vehicle, the area contractor first requests from the existing collection of vehicle contractors a measure of the confidence that the new group is attributable to one of the known vehicles. Based on the responses, the area contractor either starts up a new vehicle contractor, or awards a contract to (augments the existing contract of) the appropriate existing vehicle contractor, with the task of making certain that the new group corresponds to a known vehicle. This may entail, for example, the gathering of new data via adjustment of sensors, or contracts to new sensor nodes.

The vehicle contractor then makes two task announcements: vehicle classification and vehicle localization. The announcement of the classification task includes an abstraction of the available description of the vehicle (i.e., the currently known information). In this example, the abstraction contains a list of the fundamental frequencies of the signal groups currently associated with the vehicle. This information may help a potential classification contractor select an appropriate task (a contractor may, for example, already be familiar with vehicles that have signal groups with the announced fundamental frequencies). The award includes the complete current description. A classification contractor may be able to classify directly, given the signal group information, or on the other hand it may require more data, in which case it can communicate directly with the appropriate sensor nodes.

A localization task announcement includes data on the positions of the detecting nodes. The bid is simply an affirmative response to the announcement and the contract is awarded to the first bidder, which does the required triangulation to obtain the position of the vehicle.

Once the vehicle has been localized, it must be tracked. We assume that this is handled by the vehicle contractor which enters into follow-up localization contracts from time to time and uses the results to update its vehicle description.

There are a variety of other issues that must be considered in the design and operation of a real distributed sensing system; they are discussed in more detail elsewhere **[Smith, 1978b]**. In the following sections, we focus on issues of knowledge organization and use in the contract net, and refer back to this example to instantiate the issues raised.

4 Organization Of Knowledge

In this section we consider the contract negotiation process from a different perspective, examining the kinds of knowledge that are used, the way that the knowledge is indexed within an individual node, and distributed among the nodes.

We begin with a few definitions. ***Indexing*** indicates the "handles" placed on knowledge modules so that they can be accessed. In the next section we will see that knowledge in a contract net is indexed according to its utility for selecting suitable knowledge sources (KS's) (i.e., processor nodes) for a particular task, or for selecting suitable tasks for a particular KS. ***Distribution*** indicates where the knowledge resides; that is, in which processor nodes. We can distinguish two aspects of distribution of knowledge in a contract net: ***static*** distribution - dealing with the question of how knowledge is pre-loaded into the net (i.e., the a priori distribution), and ***dynamic*** distribution - how knowledge is acquired by a node as work on the problem progresses.

In the following sections we will concentrate primarily on the issue of knowledge indexing, together with the mechanisms that are necessary to use the knowledge in problem solving. We will also see that these same mechanisms permit interactive transfer of expertise between nodes in much the same way as any other form of information is transferred.

4.1 Indexing Of Knowledge

To consider knowledge indexing, the discussion focuses on (a) the two primary questions that must be answered by a node during the contract negotiation process, and (b) the types of knowledge that are used by the manager and potential contractors to effect this negotiation.

A manager has two questions to answer during the contract negotiation process. First,

(1) ***To whom do I address my task announcement?***

Then, once it has received a number of bids in response to an announcement, the manager must answer the question,

(2) ***How can I select the best candidates from among the potential contractors for my task?***

A node that receives an announcement must also answer two questions during the negotiation process. First,

(3) ***Am I relevant to this task and is it appropriate for me to consider making a bid?***

In addition, a node must also determine,

(4) ***Is this task the one that I want to execute next?***

In order to facilitate the contract negotiation process, we find it convenient to specify the indexing of knowledge as being either ***task-centered*** or ***knowledge-source-centered*** (***KS-centered***).

Task-centered knowledge is indexed from the point of view of a particular task, and provides information about KS's with respect to that task. At least two forms can be imagined:

(a) ***IF I have a task of the form [...] to be executed, THEN KS's of the form [...] are potentially useful.***

or,

(b) ***IF I have a task of the form [...] to be executed, THEN KS's of the form [...] are more useful than KS's of the form [...].***

KS-centered knowledge, on the other hand, is indexed from the point of view of a particular KS, and provides information about tasks with respect to that KS. Again, at least two forms can be imagined:

(c) ***IF my knowledge base contains information of the form [...], THEN tasks of the form [...] are appropriate for me.***

or,

(d) *IF my knowledge base contains information of the form [...], THEN tasks of the form [...] are more appropriate for me than tasks of the form [...].*

Both kinds of knowledge are used during the contract negotiation process. Task-centered knowledge is used first to determine the subset of nodes to which to address a task announcement (i.e., (a) provides the answer to question (1)). This type of knowledge reduces message traffic and message processing overhead because it enables focused addressing, as in the DSS, for example, where the monitor node uses task-centered knowledge to effect focused addressing in announcing the area tasks.

Task-centered knowledge is also used to determine the best course of action once bids are received (i.e., (b) provides the answer to question (2)), and hence is an effective mechanism for encoding strategies. That is, since bid evaluation functions are used to select the next KS to invoke, they are an appropriate location for strategy information that guides the operation of the problem solver.

KS-centered knowledge is used by a node that receives an announcement, first to determine that it is relevant to the announced task (i.e., (c) provides the answer to question (3)). Associating knowledge with KS's allows enhancement of the concurrency in a distributed processor because many KS's can simultaneously determine their relevance to a task; that is, each KS carries information allowing it to determine the range of tasks to which it is relevant. KS-centered knowledge of type (c) is used by nodes in the DSS example to determine that they are eligible to bid on signal tasks.

KS-centered knowledge is also used by a node to select the task it wishes to execute next (i.e., (d) provides the answer to question (4)). This type of KS-centered knowledge is another effective mechanism for encoding strategies. In the DSS, for example, the initialization strategy for signal contractors is encoded in this way.

4.2 Distribution Of Knowledge

We noted at the beginning of this section that distribution of knowledge has two aspects - static and dynamic. Static distribution is largely task-specific, and the criteria for a good static distribution of knowledge are similar to those for good problem partitioning. The distribution chosen should minimize message traffic, and should not create any bottlenecks in the system. Dynamic distribution of knowledge is the means by which nodes can acquire and transfer information and expertise as the problem progresses. The ability to effect dynamic knowledge distribution places several constraints on the design of the distributed problem solver.

Dynamic distribution of knowledge enables more effective use of available computational resources: a processor node that is standing idle because it lacks information required to perform a previously announced task can acquire the procedures necessary to execute that task. This also facilitates the task of adding a new node to an existing net, since the node can dynamically acquire the procedures and data necessary to allow it to participate in the operation of the net.

This means that nodes do not have to be functionally defined a priori; that is, any node can acquire the procedures necessary to execute any task that its physical attributes (e.g., memory, peripherals, etc.) will support. The alternative would be either forcing the node to remain idle until it hears a task announced for which it already has the necessary procedures, or pre-loading each node in the net with all the procedures that will ever be used. Neither of these alternatives is very attractive.

Procedures can be transferred between nodes in three ways. First, a node can transmit a request directly to another node for transfer of a procedure. The response to the request is the procedure code. Second, a node can transmit a task announcement in which the task is transfer of a procedure. A bid on the task indicates that another node has the code and is willing to transmit it. Finally, a node can note in its bid on a task that it requires the code for a particular procedure in order to execute the task. This is useful when the managing node already has the relevant code but wants to work on some other aspect of the task.

4.3 Internode Communication

Thus far we have concentrated on the role of the contract net problem solving protocol for interaction between nodes. In this section, we consider the ***common internode language*** which serves as the foundation on which the protocol is based. The language provides the primitive elements with which such items as task abstractions, eligibility specifications, and bid specifications are encoded. It thus provides the medium in which nodes "discuss" tasks and KS's, as well as pose the questions about eligibility to bid on tasks, rank ordering of tasks, and control of task distribution that arise during the contract negotiation process.

A relatively simple language, capable of supporting the DSS communication, has been designed. Sample messages are shown in Appendix A. It is believed that the language will support a range of other applications, but, of course would have to be increased in complexity for behavior significantly more complex than that shown in the DSS example.

The language is organized as a collection of associative triples, and has a set of domain-independent, "core" vocabulary items that can be extended with task-specific items.

The current grammar of the language is relatively simple, with the result that the messages shown in Appendix A are somewhat verbose. These messages have the advantage of being easy to write and understand for a human, but have the disadvantage of being less efficient than they might be in their use of communications resources.

The common internode language permits explicit statements of requirement to be made in messages. It is also useful in that it assists a new node in isolating the information it must acquire to participate in the operation of the net. This isolation is an aid to active distribution of knowledge (discussed in the preceding section). Finally, the language simplifies the use of local processing by a node, for example, to evaluate announcements and bids from its own point of view. A node is able to process the information in these messages because the common internode language affords a uniform interpretation of the vocabulary items by all nodes in the net.

Specialized communication is also possible. Two nodes that are linked via a contract, for example, can adopt a more compact form of communication for their messages, since no other nodes need interpret the messages. This compact form of communication can be viewed as a specialized language that the nodes use to communicate with other nodes that share their expertise. In the DSS, for example, once the area and vehicle contractors have established communication through the contract negotiation process, they might alter the language in which they communicate in order to reduce the length of messages and simplify message processing. This is possible because a context has been established through a contractual relationship.

5 Other Systems

The contract net draws upon a variety of ideas from the AI literature. In this section we relate the approach to those used in other systems.

5.1 PLANNER And Actors

The contract net task announcement is analogous to the **PLANNER** [Hewitt, 1972] goal specification, and functions similarly in providing a mechanism for advertising a task to a group of KS's, instead of invoking a specific KS by name.

By way of contrast, the contract net allows complex local processing by a node in determining its relevance to a particular task, rather than the pattern-matching that is allowed in **PLANNER**. In addition, the actor model of computation that succeeded **PLANNER** is based on the concept of a group of experts that communicate by passing point-to-point messages [Hewitt, 1977a], [Hewitt, 1977b], while there are a variety of addressing modes used in contract net messages (general broadcast, limited broadcast, and point-to-point). These different modes serve to reduce message traffic and message processing overhead. Finally, the contract net assumes a loose-coupling of tasks, whereas the actor model does not. This assumption implies a difference in the grain size of tasks into which a problem is

decomposed (large for contractors and small for actors), and results from the different motivations of the designers of the two formalisms: where actors have been used as a means of studying fundamental issues involving the nature of computation, control, and program correctness, the contract net is designed as a mechanism for problem solving, and hence views its primitive operations in terms of comparatively large, domain-specific tasks.

5.2 HEARSAY-II

The concept of a group of cooperating KS's has been used to advantage in the HEARSAY-II speech understanding system [Erman, 1975]. The contract net draws upon this model with respect to the modularity and independence of KS's. Unlike this model, however, the contract net enables focused addressing and doesn't use a blackboard, primarily due to the problems such a global data structure can cause in a distributed environment (e.g., reliability and bottleneck problems).

In addition, KS's in the HEARSAY model were seen primarily as information gathering and dispensing processes [Reddy, 1975], so that hierarchical control was not considered necessary. The contract net, on the other hand, is well-suited to hierarchical control as a result of the manager-contractor structure.

Finally, HEARSAY-II did not preserve state information about a hypothesis. In particular, there was neither a way to specify the processing that had already been applied to a hypothesis, nor the kind of processing that might yet be applied, and this made scheduling difficult [Lesser, 1977]. The contract provides a data structure with which to associate this type of information and is one way of avoiding such problems.

5.3 PUP6

The model of a group of human experts cooperating to solve a large problem was also used effectively in PUP6, a system designed to write programs based on informal specifications [Lenat, 1975]. Where interaction between the modules in PUP6 was accomplished by pattern-matching, the contract net expands on this through the use of a contract negotiation process, based on a common internode language.

PUP6 had no notion of acquired expertise, since each module in the system had a standard set of parts that did not vary over time. Contract nodes, on the other hand, have a standard core structure but have in addition a common internode language which enables them to acquire expertise via transfer of procedures and data.

5.4 Task Distribution / Transfer Of Control - A Progression

It will be useful at this point to compare the approach to task distribution provided by the contract net framework with that provided by previous problem solving formalisms. This will help make clear the ways in which the contract net view is unique and the advantages that uniqueness offers. We consider as points of comparison the techniques used in subroutine calling, PLANNER, CONNIVER [McDermott, 1974], HEARSAY-II, a hypothetical task agenda system, and the PUP6 system. We show that the contract net presents a view that is a natural successor to previous systems but is unique in several respects.

5.4.1 Terminology

We have used the term "task distribution" throughout the paper as a generalized view of what is more traditionally referred to as ***transfer of control***. That is, in a distributed system, when one processor decomposes a problem it is working on and hands one of the resulting subtasks to another processor, both processors continue working on their respective tasks; hence we refer to it as task distribution. In a uniprocessor, however, problem decomposition involves transfer of control: one process selects another process to work on a selected subtask and yields (perhaps temporary) control.

Since all of the systems we wish to use for comparison were designed for uniprocessors, we will adopt this perspective and make the comparison on the basis of transfer of control. This will provide a familiar basis for comparison without losing sight of any of the important issues. It will also serve to demonstrate that the issues we deal with in this section are fundamental issues of KS invocation and problem solving, independent of distributed processing.

5.4.2 The Basic Questions And Fundamental Differences

To make clear the place of the contract net in the sequence of invocation mechanisms that have been created, we consider the process of transfer of control from the perspective of both the caller and the respondent. We focus in particular on the selection aspects and consider what opportunities a calling process has for selecting an appropriate respondent, and what opportunities a potential respondent has for selecting the task on which to work. In each case we ask two basic questions from the perspective of both the caller and the respondent:

What is the character of the choice available?
On what kind of information is that choice based?

The answers to these questions will demonstrate our claim that the contract net view of control transfer differs with respect to:

(a) information transfer: The announcement-bid-award sequence means that there is more information, and more complex information transferred in both directions (between caller and respondent) before invocation occurs.

(b) local selection: The computation devoted to the selection process, based on the information transfer noted above, is more extensive and more complex than that used in traditional approaches, and is "local" in the sense that selection is associated with and specific to an individual KS (rather than embodied in, say, a global evaluation function).

(c) mutual selection: The local selection process is symmetric, in the sense that the caller evaluates potential respondents from its perspective (via the bid evaluation function), and the respondents evaluate the available tasks from their perspective (via the task evaluation functions).

5.4.3 The Comparison

Subroutine invocation represents a degenerate case, since all the selection is done ahead of time by the programmer and is "hardwired" into the code. As a result there is no non-determinism at runtime and hence no opportunity for choice.

A degree of non-determinism (and hence opportunity for choice) for the caller is evident in traditional production rule systems, since a number of rules may be retrieved at once. A range of selection criteria have been used (see [Davis, 1977]), but these have typically been implemented with a single, syntactic criterion hardwired into the interpreter.

PLANNER's pattern-directed invocation provides a facility at the programming language level for nondeterministic KS retrieval and offers, in the "recommendation list", a specific mechanism for encoding selection information. The THUSE construct provides a way of specifying which KS's (theorems) to try in which order, while the theorem base filter (THTBF) construct offers a way of invoking a predicate function of one argument (the name of the next theorem whose pattern has matched the goal) which can "veto" the use of that theorem.

Note that there is a degree of selection possible here, selection that may involve a considerable amount of computation (by the theorem base filter), and selection that is local in the sense that filters may be specific to a particular goal specification. But the selection is also limited in several ways. First, in the standard invocation mechanism the information available to the caller is at best the name of the next potential respondent; in effect a one-bit answer of the form "yes I match that pattern". The caller does not receive any additional information from the potential respondent (such as, for instance, exactly how it matched the pattern), nor is there any easy way to provide for information transfer in that direction. Second, the choice is, as noted, a simple veto based on just that single KS. That is, since final judgement is passed on each potential KS in turn, it is not possible for instance to make comparisons between potential KS's, nor to pass judgment on the whole group and choose the one that

looks (by some measure) the best. (Both of these shortcomings can be overcome if we are willing to create a superstructure on top of the existing invocation mechanism, but this would be functionally identical to the announcement-bid-award mechanism described above. The point is simply that the standard **PLANNER** invocation mechanism has no such facility and the built-in depth-first with backtracking makes it expensive to implement.)

CONNIVER represents a useful advance in this respect, since the result of a pattern-directed call is a "possibilities list" containing *all* the KS's that matched the pattern. While there is no explicit mechanism parallel to **PLANNER's** recommendation list, the possibilities list is accessible as a data structure and can be modified to reflect any judgments the caller might make concerning the relative utility of the KS's retrieved. Also, paired with each KS on the possibilities list is an association-list of pattern variables and bindings, making it possible to determine how the calling pattern was matched by each KS. This mechanism offers the caller some information about each respondent that can be useful in making the judgments noted above. As an indirect mechanism, however, it is less effective for information transfer than, for instance, an explicit bid mechanism.

The **HEARSAY-II** system illustrates a number of similar facilities in an event-driven system. In particular, the focus of attention mechanism has available a pointer to all the KS's that are ready to be invoked (so it can make comparative decisions), as well as information (in the "response frame") estimating the potential contribution of each of the KS's. The system can effect some degree of selection regarding the KS's ready for invocation and has available to it a body of knowledge about each KS on which to base its selection. The response frame thus provides information transfer from respondent to caller, which, while fixed in format, is more extensive than previous mechanisms. There is also a fair amount of computation devoted to the selection process, but note that the selection is not local, since there is a single, global strategy used for every selection.

There are several things to note about the systems reviewed thus far. First, we see an increase in the amount and variety of information that is transferred (before invocation) from caller to respondent (e.g., from explicit naming in subroutines to patterns in **PLANNER**) and from respondent to caller (e.g., from no response in subroutines to the response frames of **HEARSAY-II**). Note, however, that in no case do we have available a general information transmission mechanism. In all cases the mechanisms have been designed to carry one particular sort of information and are not easily modified. Second, we see a progression from the retrieval of a single KS at a time to explicit collection of the entire set of potentially useful KS's, providing the opportunity for more complex varieties of selection. Finally, note that all the selection so far is from one perspective; the selection of respondents by the caller. In none of these systems do the respondents have any choice in the matter.

To illustrate this last point, consider a (hypothetical) task agenda system in which there is a central "task blackboard" which contains an unordered list of tasks that need to be performed. As a KS works on its current task, it may discover new (sub)tasks that require execution, and add them to the blackboard. When a KS finishes its current task, it looks at the blackboard, evaluates the lists of tasks there, and decides which one it wants to execute. Note that in this system the respondents would have all the selection capability; that is, rather than have a caller announce a task and evaluate the set of KS's that respond, we have the KS's examining the list of tasks and selecting the one they wish to work on.

PUP6 was the first system to view transfer of control as a "discussion" between the caller and potential respondents. If, in response to a task broadcast, a KS receives more than one reply offering to do the task, it may "ask" questions of the respondents to determine which of them ought to be used. While this interchange is highly stylized and not very flexible, it does represent an attempt to build in explicit two-way communication.

The contract net differs from all these in several ways. First, from the point of view of the caller, we have improved the standard task broadcast and response interchange by making possible a more informative response. That is, instead of the traditional tools which allow the caller to receive simply a list of potential respondents, we have available a mechanism which makes it possible for the caller to receive extensive information from each respondent describing potential utility.

Second, the contract net emphasizes the utility of local selection. That is, an explicit place in the framework has been provided for mechanisms with which both the caller (in the bid evaluation function) and the respondents (in the task evaluation function) can invest computational effort in selecting KS's for invocation or selecting tasks to work on, respectively. These selection functions are also "local" in the sense that they are associated with and written from the perspective of the individual KS (as opposed to, say, **HEARSAY-II**'s global focus of attention function). While we have labelled this process "selection", it might more appropriately be labelled "deliberation", to emphasize that its purpose is, for the caller, for example, to decide in general what to do with the bids received, and not merely which of them to accept. Note that one possible decision is that *none* of the bids is adequate, and thus none of the potential respondents will be invoked. (Instead, the task may be reannounced later.) This choice is not typically available in other problem solving systems and hence emphasizes the wider perspective taken by the contract net on the transfer of control issue.

Finally, and perhaps most important, is what appears to be a novel symmetry in transfer of control process. Recall that **PLANNER**, **CONNIVER**, and **HEARSAY-II** all offered the caller some ability to select from among the respondents, while our hypothetical task agenda system allowed the respondents to select from among the tasks. The contract net, however, relies on the notion of contract negotiation as a metaphor, and emphasizes an interactive, ***mutual selection*** process in which task distribution is the result of a discussion between processors. As a result of the information exchanged in this discussion, the caller can select from among potential respondents (with its bid evaluation function), while the KS's can select from among potential tasks (with their task evaluation functions).

6 Limitations And Caveats

There are of course a number of limitations and caveats to consider. First, much of what we have proposed is a framework for problem solving that provides some ideas about what kinds of information are useful and how that information might be organized. There is still a considerable problem involved in instantiating that framework in the context of a specific task domain. Beyond the general guidelines offered earlier, it is not obvious, for instance, exactly what information should be in a task abstraction, bid, or task evaluation function. Yet the successful application of the machinery described above depends strongly on the choices made. In this sense, several of the mechanisms we have proposed are similar in spirit to the concept of the recommendation list in **PLANNER**: The mechanism provides a site for embedding a certain kind of information, but does not specify for a particular problem what goes in there, nor how to instantiate it in a particular domain. The utility of such mechanisms lies in their ability to help a user structure and understand a problem: We tread the traditional thin line between too much generality that provides too little guidance, and too much structure that overly constrains the user's options. More work on this is forthcoming, as we attempt to specify more detailed guidelines on appropriate use of the framework.

An important caveat in considering use of the contract net framework has been touched on earlier, in the issue of loose-coupling and the grain size of the problems attacked. It is apparent, for instance, that the communication involved in task announcements, bids, awards, etc., and the computation involved in the deliberation phase (the task and bid evaluations) may add up to a non-trivial amount of overhead. The size of the tasks being distributed must be such that it is worth this effort. It would make little sense, of course, to go through an extended mutual selection process to get some simple arithmetic done or to do a simple database access. While we discussed earlier how the full protocol can be abbreviated to an appropriately terse degree of interchange

(e.g., directed contacts and the request/response mechanism), many other systems are capable of supporting this variety of behavior. The interesting contribution of our framework lies in applications to problems where the more complex interchange provides an efficient and effective framework for problem solving.

7 Conclusion

We have described the operation of a problem solver that is based on a collection of asynchronous processor nodes that cooperate according to a contract metaphor to solve problems. In this metaphor, task distribution is viewed as an interactive process of contract negotiation.

We have noted the ways in which the contract net protocol helps to reduce message traffic and message processing overhead - through the use of task abstractions, eligibility specifications, and bid specifications in task announcements, through the use of focused addressing, and through the use of specialized interactions like directed contracts and requests.

We have considered the indexing and distribution of knowledge in such a problem solver. In this context, we have suggested two forms of knowledge indexing - task-centered and knowledge-source-centered - and demonstrated their utility in the context of a distributed sensing example.

We have noted that a common internode language is required to enable effective use of the knowledge in a distributed problem solver, and have sketched a rudimentary design for such a language.

While the ideas which form the basis of this paper have been derived from the point of view of designing a problem solver that can effectively exploit the multiple processor computer architectures that have been made possible by LSI technology, they appear to be more general in scope. Knowledge indexing and distribution, for example, are of interest in the design of future uniprocessor as well as multiple processor problem solvers.

8 Acknowledgements

The authors wish to acknowledge the assistance of Penny Nii in formulating the DSS example. Jan Aikins, Bruce Buchanan, Janet Friendshuh, Tom Mitchell, Earl Sacerdoti, Mark Stefik, and John Treichler provided helpful comments on earlier drafts of the paper.

Appendix A

DSS Sample Messages

This appendix includes abbreviated sample messages for the signal task in the DSS example. For brevity, the messages shown contain only the information mentioned in Section 2.2. Terms written in upper case are included in the core internode language, while terms written in lower case are specific to the DSS application.

For purposes of explanation, pseudo-English equivalents to the messages are also shown. The DSS of course has no human-like language processing capabilities.

Signal Task

Announcement: Needed - signal data for traffic in area A. My position is p. If in possession of sensors and located in area A, respond with position, and type and number of sensors.

```
Task Abstraction: TASK NAME signal
                  area name A
                  NODE POSITION p
Eligibility Specification: MUST HAVE DEVICE TYPE sensor
                           MUST HAVE OWN NODE POSITION
                                 area name A
Bid Specification: BID OWN NODE POSITION
                   BID EVERY DEVICE TYPE
                       sensor type number
```

Bid: Position - q. Sensors: Type S - 3, Type T - 1.

```
Node Abstraction: NODE POSITION q
                  sensor type S number 3
                  sensor type T number 1
```

Award: Report signals. Use sensors S1 and S2.

```
Task Description: sensor name S1
                  sensor name S2
```

Report: Detected signal: frequency f0, time of detection t0, strength s0, characteristics (...), detecting-node s1, position p2, sensor A1, orientation a.

```
Result Description: signal name S1
                        frequency f0
                        time-of-detection t0
                        strength s0
                        characteristics ( ... )
                        detecting-node name s1
                                       position p2
                        sensor name A1
                               type A
                               orientation a
```

References

[Anderson, 1975]
G. A. Anderson and E. D. Jensen, Computer Interconnection Structures: Taxonomy, Characteristics, and Examples, *Computing Surveys*, Vol. 7, No. 4, December 1975, pp. 197-213.

[Baer, 1973]
J.-L. Baer, A Survey of Some Theoretical Aspects of Multiprocessing, *Computing Surveys*, Vol. 5, No. 1, March, 1973, pp. 31-80.

[Brooks, 1975]
F. P. Brooks, Jr., *The Mythical Man-Month*, Addison-Wesley, Reading, Mass., 1975.

[Crocker, 1972]
S. D. Crocker, J. F. Heafner, R. M. Metcalfe, and J. B. Postel, Function-Oriented Protocols For The ARPA Computer Network, *SJCC Proceedings*, 1972, pp. 271-279.

[Davis, 1977]
R. Davis and J. King, An Overview Of Production Systems, in E. W. Elcock and D. Michie, eds., *Machine Intelligence 8*, Wiley, N. Y., 1977 pp. 300-332.

[Erman, 1975]
L. D. Erman and V. R. Lesser, A Multi-level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge, *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, September 1975, pp. 483-490.

[Galbraith, 1974]
J. R. Galbraith, Organizational Design: An Information Processing View, in D. A. Kolb, I. M. Rubin, and J. M. McIntyre, eds., *Organizational Psychology*, 2nd edition, Prentice-Hall, Englewood Cliffs, N. J., 1974, pp. 313-322.

[Hayes-Roth, 1977]
F. Hayes-Roth and V. R. Lesser, Focus Of Attention In The HEARSAY-II Speech Understanding System, *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, Mass., August 1977, pp.27-35.

[Hewitt, 1972]
C. Hewitt, *Description And Theoretical Analysis (Using Schemata) Of PLANNER: A Language For Proving Theorems And Manipulating Models In A Robot*, MIT AI TR 258, April 1972.

[Hewitt, 1977a]
C. Hewitt, Viewing Control Structures As Patterns Of Passing Messages, *Artificial Intelligence*, 8, 1977, pp. 323-364.

[Hewitt, 1977b]
C. Hewitt and H. Baker, Laws For Communicating Parallel Processes, in B. Gilchrist, ed., *Information Processing 77*, North-Holland, 1977, pp. 987-992.

[Kahn, 1972]
R. E. Kahn, Resource-Sharing Computer Communications Networks, *Proc. IEEE*, Vol. 60, No. 11, November 1972, pp. 1397-1407.

[Lenat, 1975]
D. B. Lenat, Beings: Knowledge As Interacting Experts, *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, September 1975, pp. 126-133.

[Lenat, 1976]
D. B. Lenat, *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*, STAN-CS-76-570 (HPP-76-8), Department Of Computer Science, Stanford University, July 1976.

[Lesser, 1977]
V. R. Lesser and L. D. Erman, A Retrospective View Of The HEARSAY-II Architecture, *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, Mass., August 1977, pp. 790-800.

[McDermott, 1974]
D. V. McDermott, and G. J. Sussman, *The Conniver Reference Manual*, MIT AI Memo 259a, January 1974.

[Reddy, 1975]
D. R. Reddy and L. D. Erman, Tutorial On System Organization For Speech Understanding, in D. R. Reddy, ed., *Speech Recognition*, Wiley, N. Y., 1975, pp. 457-479.

[Smith, 1977]
R. G. Smith, The CONTRACT NET: A Formalism For The Control Of Distributed Problem Solving, *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Cambridge, Mass., August 1977, p. 472.

[Smith, 1978a]
R. G. Smith, *Issues In Distributed Sensor Net Design*, HPP-78-2 (Working Paper), Heuristic Programming Project, Stanford University, January 1978.

[Smith, 1978b]
R. G. Smith, *A Framework For Problem Solving In A Distributed Processing Environment*, doctoral dissertation, Stanford University, 1978, (in preparation).

[Zucker, 1977]
S. W. Zucker, *Vertical And Horizontal Processes In Low Level Vision*, Report No. 77-4, Department of Electrical Engineering, McGill University, May 1977.

Describing Programming Language Concepts in LESK

Douglas R. Skuce
Dept. of Computer Science
Concordia University
Montreal

ABSTRACT

LESK (Language for Exactly Stating Knowledge) is a synthesis of concepts from programming languages, linguistics and logic, and is intended for eventual implementation as a "knowledge base system", though it is now valuable "on paper" to make precise the definition of terminology in some domain. In this paper, an approach to describing basic programming language concepts is proposed using LESK as an alternative to the usual mixture of formal (e.g. BNF) and informal (i.e. natural language) descriptions found in typical programming language manuals. We demonstrate the utility of LESK to make precise, in a unified way, the use of both formal expressions and natural language phrases required in defining a programming language both syntactically and semantically. We thereby alleviate the common problems of vagueness, ambiguity and inconsistency of terminology which plague most programming language manuals. Some of our examples will use PASCAL. The method is general enough to apply to any field with well-defined terminology.

INTRODUCTION

In previous papers (Skuce 75, 76) and in (Skuce 77), the language LESK, whose function is to precisely state knowledge, was introduced. A natural application of LESK for computer scientists is to the description of programming languages; anyone who has been frustrated by the imprecise terminology of many programming language manuals will appreciate this need. Such confusion often involves terms like "value", "number", "constant", "object", "type", "variable", etc. so that one is often not sure, e.g., "is an array a value?", "are 'reals' a scalar type?", and what kind of entity is 'red' in 'color = (red, wt, blu)'? (these last two are from Jensen and Wirth 75).

Proper understanding of any well-defined field must begin with precise and unambiguous definitions of the terminology, thereafter consistently applied. At present, most manuals and other descriptions of programming languages introduce terminology using an unsatisfactory combination of precise syntactic definitional methods (e.g. BNF), which usually include little semantic information, possibly some non-standard formalism for semantic description, and a considerable bulk of prose description, on which we rely most.

In this paper we will briefly explore an alternate descriptive technique: using LESK to introduce the terminology of a typical programming language in a manner which synthesizes the three methods into one easily readable form. The eventual implementation of a LESK "knowledge base system" (KBS) would assure that this usage was consistent.

LESK SEMANTIC STRUCTURES

All knowledge in LESK is expressed as symbol strings, which are English-like versions of predicate calculus expressions, composed of LESK primitives and user-defined terminology. The knowledge contained in a set of LESK expressions is the (finite) set of all expressions deducible from the set using simple substitution and transformation rules, i.e. this set of "theorems" determines the answers to a set of answerable questions.

The most elementary kind of definable knowledge items are termed *atoms*. These have no sub-items accessible via operators, and may be denoted by noun phrases (NPs) or formal expressions. One may then define composite notions called *collections* using either atoms or other collections. This process is analogous to writing LISP functions. By "define", we do not mean "generate in a computer memory a structure which physically contains the components", but rather "create, in a mind or machine, rules which define *categorically* (sometimes called *generically*) the behaviour of all *instances* of the knowledge item". Thus we are merely constraining the behaviour of sets of symbol strings, much as a formal grammar does.

Collections are said to be composed of *elements*; an explicit list of all of these is called the *extension*. LESK is intended mainly for defining knowledge about collections at the categorical level, i.e. without involving the elements of the extension at all. There are four kinds of collections, depending on whether or not the extension is ordered, and whether or not the extension may have repeated elements, as follows:

kind	ordered?	repeat?	example
class	no	no	(1,2,3) = (2,3,1)
bag	no	yes	(1,1,2,3) = (3,1,2,1)
seq	yes	yes	(A,B,C,C) ≠ (B,A,C,C)
uniseq	yes	no	(A,B,C,D)

We will not need explicit notation to distinguish these. The *size* of a collection, i.e. the number of possible elements, is finite and variable. A *tuple* is a seq of fixed size.

SEMANTIC CATEGORIES

All terminology must be assigned one of a small number of *semantic categories:*

Concepts are either atoms, classes of concepts, or tuples of concepts, and play noun-like roles. They may be denoted either by NPs or by formal expressions.

Functions map concepts into concepts, and are denoted like concepts.

Relations are either stative English verb phrases (VPs) having two or more "cases", or else are mathematical relations like . The arguments (i.e. the cases) are usually concepts.

a state is a set of relations which hold over some time interval.

an action is a definition of a change of state. When a LESK system *executes* an action, the effect is analogous to the permanent effect of a conventional procedure.

DEFINING BASIC NOTIONS OF PROGRAMMING LANGUAGES IN LESK

Consider the student who is encountering basic programming language concepts and terminology for the first time in a typical text which uses terms like "value", "object", "type", "variable", "array", "integer", etc. somewhat loosely. Frequently one sees phrases like "3 is an integer" and "integers are a type"; shall we conclude that 3 is a type? Of course, once one understands what is intended, there is no problem; the problem is in the initial learning phase before the meaning "dawns". Meanings shouldn't have to "dawn", they should be initially clear, as they are in good mathematical texts.

Let us decide that the most important basic notion is that of "value". We then define:

```
VALUE/S  are a class of concepts X

kinds: 1. SIMPLE VALUES
       2. STRUCTURED VALUES

X has a TYPE T
X is a member of T
X can be ASSIGNED TO a VARIABLE V
end
```

Thus we have introduced the following terminology (lower case denotes LESK primitives): VALUES (singular: VALUE) are *explicitly* defined as a class of concepts, i.e. a VALUE is a concept. There are just two disjoint subclasses (the primitive 'kinds'). There is a functional mapping (the primitive 'has') into a class to be termed TYPES, here *implicitly* introduced, about which we shall say more in a moment. The primitive 'is a member of' behaves differently from 'is a'; the latter permits "property inheritance", the former does not (see below). The 2-place relation 'is ASSIGNED TO' is possible between instances of VALUES and instances of an implicitly introduced class VARIABLES, to be defined later. This relation does not necessarily hold (can) but is introduced here to clarify the relationship between the terms VALUE and VARIABLE. It will be explicitly defined later.

Now we may define:

```
SIMPLE VALUE/S are a class of VALUES X

X is an atom

kinds: 1. ENUM VALUES
       2. REALS

end

ENUM VALUE/S are a class of SIMPLE VALUES

kinds: 1. INTEGERS     2. CHARACTERS
       3. BOOLEANS     4. ENUM IDENTIFIERS

end

INTEGER/S are a class of ENUM VALUES X

X :- [<SIGN>]<DIGITSTRING>
the TYPE of X = the INTEGER TYPE

end
```

The first of these definitions states that SIMPLE VALUES are atoms (whereas VALUES may be collections) and introduces the terms for the two kinds of SIMPLE VALUE. In the second definition, the class of ENUM IDENTIFIERS will be used later on to clarify what is ambiguously termed "scalar types" in Jensen and Wirth (75) (this is an excellent example of inadequate terminology in a well-known programming language description). The third definition provides an actual denotation (-) for INTEGERS; collections whose elements are to be syntactically recognizable require such a specification. We will leave SIGN and DIGITSTRING undefined. At this point, to clarify some of Jensen and Wirth's terminology, we might make the following *stand alone* statement:

BASIC VALUES = SIMPLE VALUES - ENUM IDENTIFIERS

DEDUCTION AND PROPERTY INHERITANCE

All question answering in a LESK system is to be done by a small number of specialized transformations on the actual expressions found either in the bodies of definitions or in stand alone statements. The basic transformations are substitution of variables and "property inheritance". Any definition "inherits" properties (i.e. statements in a definition) from its supercollections, though not all properties inherit (e.g. 'kinds'). Thus, one may substitute the phrase 'an INTEGER' for X in VALUES, or, since 3 is syntactically recognizable as an INTEGER, 3 may be substituted. One could not freely substitute any denotation for subclasses or instances of TYPES however, because of the functional dependence.

Any question then must be analysable into a series of elementary deductive transformations, just as database query languages must formulate acceptable questions in terms of a number of basic accessing operations on the database. This approach means that LESK systems would be modeled more after database systems than the more traditional "AI" predicate calculus-based theorem prover. Implementation of a LESK system is therefore seen as an extension to existing relational database technology.

Now let us consider some STRUCTURED VALUES. How, for example, should we treat the term 'array', which question we take to be equivalent to: what is the most basic knowledge we require about arrays? Is it correct to consider an array to be a VALUE? Since all the contexts we allow for STRUCTURED VALUE will accept the term 'array' as it is usually understood, we make the following definition:

```
ARRAY/S are a class of STRUCTURED VALUES X
   "    are a class of tuples

X :-  (X1, X2,...,Xn)
  Xi is a VALUE
  Xi is called a COMPONENT of X
the TYPE of X is an ARRAY TYPE
X has a TYPE called its COMPONENT TYPE C
the TYPE of Xi = C

end
```

Thus ARRAYS inherit properties like 'an ARRAY can be ASSIGNED TO a VARIABLE', and properties from the primitive class tuples, such as fixed size (n). The 'is called' primitive is very frequently used to give a local name to some variable. A question answering procedure looking for the TYPE associated with an ARRAY would find something called an ARRAY TYPE, which is of course a TYPE. If one were searching either for the COMPONENT TYPE of an ARRAY, or the TYPE of a COMPONENT of an ARRAY, C would be accessed. We have used the standard denotation for tuples so that we could refer to the Xi; of course in most programming languages, one cannot denote instances of arrays.

Next we ought to define TYPE:

```
TYPE/S are a class of classes X of VALUES

kinds: 1. ENUM TYPES  2. the REAL TYPE
       3. ARRAY TYPES

X has a class of OPERATIONS
X has a class of relations
X has a unique TYPE FORMAT or TYPE NAME

end
```

Thus, an instance of a TYPE is a class of VALUES, of which we will discuss only three kinds. Every TYPE has a class of OPERATIONS, which we would define as actions, since they define the mappings from X into itself. The relations on a TYPE are the usual binary predicates. (This term is in lower case since is not being given a name here.) The TYPE FORMAT or TYPE NAME specify how a TYPE is to be denoted in an actual program, which is often different from the term we wish to use for the concept itself. Such essential distinctions are often unclear in many presentations. The REAL TYPE is the name we have chosen to refer to the TYPE of the REALS; there is only one REAL TYPE, hence 'the'.

In PASCAL we have the notion of the ENUM (enumerative) TYPES, though in the PASCAL manual (Jensen and Wirth 75) this term is not used. Instead the term "scalar type" is used, but this term is ambiguous there, as we noted above, sometimes meaning what we call ENUM TYPES and sometimes what we term DECL ENUM TYPES. To clarify:

```
ENUM TYPE/S are a class X of uniseqs of VALUES

kinds: 1. STD ENUM TYPES  2. DECL ENUM TYPES
X has the ENUM OPERATIONS, the ENUM RELATIONS

end
```

Though we have said that ENUM TYPES are uniseqs, whereas TYPES were classes, this is not contradictory, since a uniseq is a restricted form of class (i.e. it is an ordered class). Now:

```
DECL ENUM TYPE/S are a class of uniseqs Y of
ENUM IDENTIFIERS

Y :- (Y1,Y2,...,Yn)

end
```

whereas the

```
STD ENUM TYPE/S

kinds: 1. the INTEGER TYPE  2. the BOOLEAN TYPE
       3. the CHARACTER TYPE

end
```

(The default rule is that <ADJ><NOUN>is a subclass of <NOUN>.) The three kinds of STD ENUM TYPE which we consider correspond to the three kinds of VALUE. The relation between the INTEGERS and the INTEGER TYPE is:

the INTEGER TYPE X is a STD ENUM TYPE

X belongs to any INTEGER
the extension of X = the extension of the INTEGERS

end

Thus we have equated (=) the extensions of two classes, but not the classes themselves, for had we written 'the INTEGER TYPE = the INTEGERS' this would mean that the two phrases could be substituted anywhere one for the other, which clearly we do not want. A LESK system would correctly conclude that 3 is a member of the extension of the INTEGER TYPE, but not that 3 is a TYPE. This example illustrates again the basic principle that LESK semantics base class equality or containment on syntactic substitutive ability.

Again:

BOOLEAN/S are a class X of ENUM VALUES

the extension of X = ('true', 'false')

end

the BOOLEAN TYPE X IS A STD ENUM TYPE

the extension of X = the extension of the BOOLEANS
X belongs to the BOOLEANS

end

The primitive 'belongs to' is the inverse of 'has', hence 'the TYPE of the BOOLEANS = the BOOLEAN TYPE'.

Before we go on to illustrate other semantic categories, we will need:

VARIABLE/S are a class of concepts X

X :- an IDENTIFIER or a VARIABLE EXPRESSION
X has a TYPE T
X can have a VALUE V
the TYPE of V = T

end

Hence there are two ways of denoting VARIABLES, and if a VARIABLE has a VALUE (which will be the case if it is ASSIGNED TO a VALUE) then their TYPES must be equal.

ASSIGNMENT: A RELATION AND AN ACTION

We illustrate next the two other most important semantic categories, relations and actions. The relation 'is ASSIGNED TO' has been used between VALUES and VARIABLES implicitly. If we made no other use of 'is ASSIGNED TO' then we would know no more about it than it 'can' hold between these classes. If we make an explicit definition, we can say more:

X is ASSIGNED TO Y is a relation Z

X is a VALUE
Y is a VARIABLE
the TYPE of X = the TYPE of Y
Z is caused by the ASSIGNMENT action

end

Had we not already constrained the use of this relation by its implicit use, we can constrain it as above. What we have added is a *causal* relation which holds between relations and actions, or between actions and actions. This allows a LESK system to answer questions like "why does TEMP have the VALUE 3?", or "how could 3 be ASSIGNED TO a VARIABLE?".

Relations denoted by stative verb phrases are a form of tuple called 'identified' tuples, i.e. an instance of one can be recognized as such. Tuples used as concepts may be 'unidentified', in that their denotation is simply of the form (x,y,...), which leaves unknown which concept they are an instance of. For example, '3 is ASSIGNED TO TEMP' is an identified 2-tuple, but '(3, TEMP)' would be an instance of any 2-tuple concept which paired a VALUE with a VARIABLE.

The notion of an action is the most complex of the semantic categories, since it alone involves defining change. Recall that a *state* is a set of LESK statements, while an *action* defines a change of state. There are two methods of defining state changes, one copied from STRIPS (Fikes and Nilsson 71) and one modeled after algorithmic programming languages. We illustrate first the simpler former one:

X := Y is an action A called ASSIGNMENT

the environment of A is a PROGRAM
X is a VARIABLE
Y is an EXPRESSION E
the VALUE of E = V
the denotation of A is called an ASSIGNMENT STATEMENT

precond: the TYPE of V = the TYPE of X
remove: Z is ASSIGNED TO X
assert: V is ASSIGNED TO X

end

This definition has the following features. The noun ASSIGNMENT is used to refer to the action, as we did in defining the 'is ASSIGNED TO' relation. The *environment* of any category is a state, i.e. a name of a context in which it may appear. Thus the

action called ASSIGNMENT can appear in a PROGRAM. All of our definitions would require some environment, which we have not specified. We might call it 'programming languages'. The classes of PROGRAMS and EXPRESSIONS are complex; we shall define the former. The *denotation* of a phrase X is 'X', i.e. by enclosing it in ' marks, we may refer to it. If asked "what is the denotation of an ASSIGNMENT?", a LESK system would reply 'X := Y', where X.... If asked "what is 'X := Y'?" the reply is "an ASSIGNMENT STATEMENT", whereas "what does 'X := Y' denote?" is answered ASSIGNMENT.

The various labeled statements are intended to be individually accessible, hence the labels. For example, one could ask "what is the precond of ASSIGNMENT?" In the context PROGRAM, if a LESK system were *executing* the action of ASSIGNMENT, the precond would be checked first, i.e. unless it were true, the action would abort. The statements labeled 'remove' are then to be removed from the envionment (Z is anything found in this relation in the environment) and the statements labeled 'assert' are added, as in STRIPS. This ensures that only one is 'ASSIGNED TO' 2-tuple involving X is then in the environment. By modeling change by removing and adding n-tuples to an environment, we make LESK compatible with relational database technology.

BASIC FACTS ABOUT PROGRAMS

What are the most elementary facts we would want a LESK system (or a human beginner) to recall about PROGRAMS? As usual, we equate this question to: "What statements can the term PROGRAM appear in?" First we define PROGRAMS in general:

```
PROGRAM/S are a class of seqs X

the environment of X = PROGRAMMING LANGUAGES
X contains a DECLARE LIST D
X contains a STATEMENT LIST S
D precedes S
X has a SYMBOL TABLE
X can be COMPILED
X can be EXECUTED
X SPECIFIES an ALGORITHM or COMPUTATION

end
```

Obviously PROGRAMS are seqs. Their environment's name has been chosen loosely; one might prefer just 'programming'. The main function of an environment is to allow a LESK system to find needed facts without having to search its whole database monolithically, e.g. if we said we were discussing PROGRAMMING LANGUAGES, it would not search under DINOSAURS, unless we allowed it to. Environments are to be organized in the same way as all other categories, so that one can define them and relations between them.

The primitive 'contains' applies to collections to require that certain elements be present. 'Precedes' applies to elements of a seq. SYMBOL TABLE will be defined in a moment. The form verb root ED is recognized as defining an action applicable to, in this case, PROGRAMS. One can thereby state what actions apply to a concept without having to define them. SPECIFIES is simply a 2-place relation.

To define a PASCAL PROGRAM, unless we want to give more properties peculiar to PASCAL PROGRAMS, we may simply give the syntactic form, i.e. how it is denoted:

```
PASCAL PROGRAM/S X

X :- X1 X2 X3 X4 X5 X6 X7

X1 :- 'PROGRAM'
X2 is a PROGRAM IDENTIFIER
X3 is a tuple of FILE IDENTIFIERS
X4 is the DECLARE LIST of X
X5 :- 'BEGIN'
X6 is the STATEMENT LIST of X
X7 :- 'END.'

end
```

If one were to input this definition to a LESK system and interchanged X4 and X6, the system would reply: 'the DECLARE LIST of a PROGRAM precedes the STATEMENT LIST'. Thus definitions, as well as instances, must conform to their superclasses.

Next we define:

```
DECL LIST/S are a class of uniseqs X of DECLARATIONS

X contains some TYPE DECLARATIONS T
X contains some VARIABLE DECLARATIONS V
T precedes V

end

DECLARATION/S are a class of tuples

kinds: 1. TYPE DECLARATIONS
       2. VARIABLE DECLARATIONS

end

TYPE DECLARATION/S X

X :- X1 '=' X2
X1 is a TYPE IDENTIFIER or TYPE NAME
X2 is a TYPE NAME or TYPE FORMAT

end

VARIABLE DECLARATION/S X

X :- X1 ':' X2
X1 is a VARIABLE
X2 is a TYPE IDENTIFIER or TYPE NAME
X1 :- a VARIABLE IDENTIFIER

end
```

We have left out of these definitions certain semantic restrictions which are expressible in LESK but which would obscure the clarity for our expository purposes. An example would be that the X2 of a VARIABLE DECLARATION must have appeared earlier as the X1 of some TYPE DECLARATION if it is a TYPE IDENTIFIER. It is exactly these context sensitive rules that are so difficult to formally express. Other formal semantic methods (Marcotty, Legard and Bochman 76) are considerably more unreadable for these purposes than LESK.

Next we define:

```
SYMBOL TABLE/S are a class of classes X of tuples Y

X belongs to a PROGRAM P
Y :- (Y1, Y2, Y3)
Y1 is a VARIABLE
Y2 = the TYPE of Y1
Y3 = the VALUE of Y1 or 'UNDEF'

end
```

Here we have simplified matters by including Y3 to hold the VALUE of Y1, to avoid having to discuss 'locations'.

The second form of action definition is illustrated by the following definition:

```
to CREATE a SYMBOL TABLE X is an action A

X belongs to a PROGRAM P
D is the DECLARE LIST of P
A is caused by a COMPILATION

begin: X := ∅
       for each VARIABLE DECLARATION V in D do
                X := X union (the VARIABLE of V,
                              the TYPE of V,
                              'UNDEF')
end
```

Thus we define in a very conventional manner the algorithm for constructing a SYMBOL TABLE. This is simply because such notation is the best we have for defining sequences of changes involving the notion of assignment. Note that := here is a LESK primitive, to be modeled by database updates just as the earlier definition we gave for it in the environment of PROGRAMMING LANGUAGES.

Finally we may define the action of COMPILATION:

```
to COMPILE a PROGRAM P is an action A called a
COMPILATION

S is the SYMBOL TABLE of P
A has a MACHINE MEMORY M
Y is a CODE SEGMENT
Z is a uniseq of CODE SEGMENTS

begin: CREATE S
       put S in M
       for each STATEMENT X in the STATEMENT LIST
                of P do
                     begin
                     COMPILE X INTO Y
                     append Y to Z
                     end

       put Z in M

end
```

Hence the action of COMPILATION begins by executing the 'CREATE a SYMBOL TABLE' definition. Only the top-most aspect of COMPILATION is specified, i.e. that the action to COMPILE a STATEMENT INTO a CODE SEGMENT is to be executed for all STATEMENTS. Hence we would next have to define this action, of which there would be many kinds, one for each kind of statement. We shall not go lower in the detail hierarchy. The primitives 'put' and 'append' have an obvious meaning. The notion MACHINE MEMORY is difficult; one suggestion would be:

```
MACHINE MEMOR/Y/IES are a class of tuples X

X :- X1 X2 X3 X4
X1 is an INPUT STATE
X2 is an OUTPUT STATE
X3 is a SYMBOL TABLE of a PROGRAM P
X4 is a CODE SEGMENT of P

end
```

A HYPOTHETICAL DIALOGUE

The following "dialogue" should assist the reader in obtaining a better appreciation of how our basic definitions (plus a few more) would function in a LESK KBS. User inputs begin with : and we will not distinguish LESK primitives in lower case.

:IS -7 A VALUE?
YES. (The syntactic form is recognized.)
:IS X A VARIABLE?
YES, BUT IT COULD BE OTHER THINGS. (i.e. the character X is acceptable as a denotation for a VARIABLE, but it could denote other entities as well, unlike -7.)
:LET X AND Y BE VARIABLES. (i.e. we create two instances of the class VARIABLE. Now the above question would be answered 'YES'.)
:LET P BE A PROGRAM. 'X := Y' BELONGS TO P.
DOES P HAVE A SYMBOL TABLE?
YES.
:SHOW ME IT. (IT refers to the last concept.)
∅ (The SYMBOL TABLE is empty before COMPILATION.)
:COMPILE P
I CANNOT. THE STATEMENT LIST OF P CONTAINS A STATEMENT 'X := Y' WHICH HAS NO TYPE DECLARATION FOR X OR Y. (This necessary precond for COMPILATION was not specified in our definition, but would be in a real system.)
:WHAT IS A TYPE?
(The system would output the TYPE definition.)
:IS INTEGER A TYPE?
NO. AN INTEGER IS AN ENUM VALUE.
:WHAT IS THE TYPE OF AN INTEGER?
THE INTEGER TYPE.
:IS THE TYPE OF A VARIABLE RELATED TO THE TYPE OF ITS VALUE?
WHEN A VARIABLE X HAS A VALUE Y, THE TYPE OF X = THE TYPE OF Y.
:WHAT CAUSES THE 'CREATE A SYMBOL' TABLE ACTION?
A COMPILATION.

RELATED WORK

This work has been strongly influenced by several projects. The concept of a KBS for scientific purposes was developed extensively by Thompson and Thompson (75), though in a much more elementary way (they did have a running system though). More recent projects which attempt to provide a natural-language-like access to technical knowledge include MYCIN (Shortliffe 76) and PROPHET (Duda et al. 77). The former stresses its ability to accept rule definitions from users in an English-like format (all rules are simple production rules) and to 'explain' how it performed a deduction using these rules.

The surface structure and deductive semantics of LESK are based on mappings between relatively rigid syntactic forms called 'patterns' in the Planner family of languages (Bobrow and Raphael 74). However these languages did not provide primitives which match the nature of knowledge, but rather catered to how people seem to like to write LISP programs. More recently, systems have appeared which, like LESK, attempt to provide more epistemologically oriented structures, e.g. KRL (Bobrow et al. 77), FRL (Roberts and Goldsmith 77) and K-NET (Fikes and Hendrix 77). All these systems share with LESK the attempt to 'package' knowledge in natural 'chunks'.

A third influence has been relational databases, and the various 'front-ends' which have been devised to supply more subtle natural-language-like interaction, and deduction based on categorical knowledge about the data domain. The work of Mylopoulos' group (e.g. Wong and Mylopoulos 77) typifies this research.

CONCLUDING REMARKS

In considering the merits of our proposal, the reader is invited to compare several contrasting techniques of programming language description. In a typical manual (e.g. Jensen and Wirth) the readability and ease of finding answers to questions are highly variable (it is difficult to answer the question "what is a scalar type?" in Jensen and Wirth). What is most lacking is the precise definition of the semantic terminology and rules. This is usually only done in the other extreme of description, typified by Donahue (76), who offers a precise semantic description of a subset of PASCAL, in which the notational subtlety precludes its being useful to learn the basic ideas, or as a working manual.

The ALGOL 68 Revised Report (van Wijngaarden et al. 77) offers the precision and completeness missing in other approaches. Indeed, many of its sentences are virtually LESK statements in their rigid clarity. It is felt that it is still far less readable than LESK, though it is in fact a complete description of a very complex language. The proper test, of course, would be to describe all of ALGOL 68 in LESK and submit the two descriptions to a panel of judges. We feel that the hierarchically introduced LESK definitions are a better compromise in the gap between the ALGOL 68 report style and the typical programming language manual style.

A word is in order regarding 'semantic nets'. We haven't drawn any net-like diagrams; why not? The key tool in LESK and indeed in all mathematical notation is the notion of variables, which unfortunately are awkward to indicate on semantic net diagrams. "Partitioned semantic nets" are one attempt to do this. One could make from our notation some form of net diagram, but we do not recommend it as contributing to the clarity. If such diagrams had a distinct advantage over linear notation with variables, they would have been in wide use before AI "discovered" them.

It is sometimes objected that the stilted rigidity of LESK statements is a drawback to ease of expression. The danger with systems which permit a much wider syntactic range is that the user may be misunderstood by the system, i.e. that the user will not be able to predict how the system will interpret a statement. It is assumed that any user of LESK is knowledgeable in LESK semantics, and the surface structure is intended to make it easy to see what can and cannot be inferred from a statement. If one wanted more natural syntax, well-known techniques are available. The effect of any more elaborate input could be verified by having the system rephrase the input in LESK format, which the user would have to understand.

It should be finally noted that this approach to knowledge description is in no way limited to programming languages, or even to computer science. The intention of LESK is to provide a generally applicable capacity to combine formal and natural language descriptions of any well-defined subject in a manner comprehensible to both humans and machines.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the support of the Izaak Walton Killam Scholarship at the Montreal Neurological Institute for part of the research reported here.

REFERENCES

Bobrow, D. and Raphael, B. (1977)
New Programming Languages for Artificial Intelligence Research. Computing Surveys, v. 6 no 3 (Sept.).

Bobrow, D., Winograd, T., et al. (1977)
Experience with KRL-0: One Cycle of a Knowledge Representation Language. IJCAI5* (August).

Donahue, J. (1976)
Complementary Definitions of Programming Language Semantics. Springer-Verlag, New York.

Fikes, R. and Hendrix, G. (1977)
A Network-Based Knowledge Representation and its Natural Deduction System. IJCAI5, pp. 235-246.

Fikes, R. and Nilsson, N. (1971)
STRIPS: A New Approach to the Application of Theorem proving in Problem Solving. Artificial Intelligence, v. 2, pp. 189-208.

Jensen, K. and Wirth, N. (1975)
PASCAL User Manual and Report, second edition. Springer-Verlag, New York.

Marcotty, M., Legard, H. and Bochman, G. (1976)
A Sampler of Formal Definitions. Computing Surveys, v. 8 no. 2, pp. 191-276 (June).

Roberts, B. and Goldstein, I. (1977)
The FRL Primer. MIT AI Lab Report 408.

Skuce, D. (1975)
An English-like Language for Qualitative Scientific Knowledge. IJCAI4, pp. 593-600.

Skuce, D. (1976)
Towards a Semantics for a Scientific Knowledge Base. Proceedings of the First National Conference on the Canadian Society for Computational Studies in Intelligence, University of British Columbia (August).

Skuce, D. (1977)
Toward Communicating Qualitative Knowledge Between Scientists and Machines. Ph.D. dissertation, Dept. of Electrical Engineering, McGill University, Montreal.

Shortliffe, E. (1976)
Computer-Based Medical Consultations: MYCIN. Elsevier, New York.

Thompson, F. and Thompson, B. (1975)
Practical Natural Language Understanding: the REL System as Prototype. in: Advances in Computers, M. Rubinoff and M. Yovits, eds. v. 13. Academic Press, New York.

van Wijngaarden, A., Mailloux, B., Peck, J., Koster, C., Sintzoff, M., Lindsey, C., Meertens, L. and Fisker, R. (1977)
Revised Report on the Algorithmic Language ALGOL68. Sigplan Notices, v. 12 no. 5 (May).

Wong, H. and Mylopoulos, J. (1977)
Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management. INFOR v. 15 no. 3, pp. 344-383.

---

* IJCAIn means the nth International Joint Conference on Artificial Intelligence

TONAL
Towards a New AI Language

D.J.M. Davies
Department of Computer Science
University of Western Ontario,
London, Canada N6A 5B9

5-May-1978

ABSTRACT

This paper describes with some examples a new programming language for Artificial Intelligence applications. The language TONAL is based generally on the syntax and semantics of POP-2, but is modified and extended in various ways.

TONAL is a 'structured' programming language for more reliable programming. It is intended to make the programming process in A.I. more reliable and less troublesome. However, the language is still interactive and incremental in nature, permitting easy debugging and experimentation with programs. This is the principal novelty, since hitherto block-structured languages have not normally permitted interactive modification of program code.

TONAL offers basic, extensible facilities for pattern matching and for building special control structures. The pattern matching facilities are integrated with a mode (type) system for variables and data structures. Multiple, potentially parallel processes are provided as a standard facility, and permit coroutine systems to be constructed easily as a special case. Other control regimes such as backtracking may be constructed.

1.0 INTRODUCTION

1.1 Why Another Language?

Why develop another language for AI programming? Most AI programs of the last decade have been written in one of the three 'primary' AI languages - LISP(1), SAIL(2) or POP-2(3) - or in one of the many other systems derived from or implemented in those languages. For example, Micro-Planner, QLISP, PLANNER, Conniver, Popler, Resolution systems, Production language systems, and many other applications have been coded using one of the primary languages as a base.

It can be seen that any AI language has to be suitable not only for 'application' programming, but also for the systems development work involved in constructing interpreters for other special-purpose languages. In addition, users of LISP and POP-2 are accustomed to interactive incremental interpreters which make it easy to experiment with programs and debug them interactively in the source language. This capability must also be rated very important. Sandewall(4) discusses this in connection with LISP usage.

Recent work on programming language design has emphasises the importance of procedural and data abstractions, and of a modular approach to building understandable maintainable programs. In this respect most of the AI languages mentioned above, excluding only SAIL, are partially deficient.

LISP dates from before ALGOL 60, and offers essentially no hierarchical static program structuring. It also offers no automatic data typing for structures, using lists for almost everything. Programmers usually learn to impose their own structure on programs, but the language offers little help in itself. POP-2 offers good data-structuring facilities, but still encourages the creation of a program as a series of independent separate functions. Only SAIL offers block-structured programs, but it does not have a wide, extensible variety of non-numerical data modes.

This characteristic of LISP and POP-2 has been elevated almost into a design principle for AI systems, in some quarters. It is argued that 'knowledge' should be divided up and represented by a large number of independent modules, each of which should be activated automatically when it becomes appropriate. This reaches its extreme case in some PL systems (where this mode of organization was originally selected for reasons of psychological modelling) and in some theorem proving languages. However, it is also apparent in most other languages, such as Micro-Planner, Conniver, POPLER and PLASMA.

These languages do indeed encourage modularization, but only to about the same extent that FORTRAN encourages structuring of programs into subroutines. Most function names are global in scope, and many data-structures are also globally available. My own experience with POPLER programs using demons and data-bases is that many items and demons should not have a global scope, but should have limited access. Many bugs arose in programs because demons or other items were activated unexpectedly, out of context. These bugs can also be difficult to track down, or they may go unperceived for long periods of time. These problems can be met by making use of a block-structured language, so identifiers are not accessible except in the scope in which they are required. By applying the same scope rules to data-base contents, a uniform system is obtained for limiting scopes.

1.2 Related Systems

ECL (Wegbreit 5,6,7) is an extension and modification of LISP motivated by similar considerations. ECL is a complete system of a language and associated tools, and EL1 is the name of the programming language used. It is aimed at 'difficult' projects. Emphasis is placed not only on extensibility, but also on 'contractability': the ability to have the system compact the code generated for a program once the programmer makes irreversible commitments about variable types, function definitions, etc. In some respects, TONAL can be seen as an attempt to provide within the POP-2 environment some of the features of ECL. However, there are also major differences between these languages.

TELOS (Travis 8) is designed with the same desire to improve the structuring and abstraction facilities available in an AI language. However, it is based on PASCAL, so the language makes full typing mandatory, and is not designed specifically for interactive, incremental use. In many other respects, however, the TELOS language offers comparable capabilities.

KRL (9) is one of the many other languages and systems developed recently for AI work. KRL represents a particular approach to representing information through a combination of 'static' descriptions and procedures, with emphasis on multi-processing. TONAL is more 'generalised' than these other systems in that it is intended to be a suitable base for implementing systems like KRL, FRL, etc.

1.Pak and 2.Pak (10) are comparable with TONAL in scope. They are AI implementation languages, based on a SNOBOL environment. They are block-structured, like TELOS and TONAL, but as in the case of TELOS not much emphasis has been placed on facilitating interactive program development.

1.3 Tonal

TONAL is an AI-oriented language; it provides facilities for pattern matching, demons, and data-bases, and for arbitrary data-structures permitting also 'data-directed' programming. It also provides structuring and abstraction mechanisms so that complicated programs can be modularised effectively, and so that encoded 'knowledge' can be restricted to just those program scopes in which it should be applicable.

This paper describes TONAL - a new AI programming language. TONAL is intended for general purpose programming in the AI context, and also comprises a <u>system</u> which permits interactive development and experimentation with programs.

TONAL has its roots in POP-2(3) and POP-10 (11). It provides a variety of control and data abstraction capabilities, which can be further extended in user programs. New data modes can be defined, and the language syntax can be extended by defining new macros and infix operators. The language is so constructed as to encourage understandable programs, particularly by controlling the accessibility of variable and mode names. Nevertheless, it is easy to experiment with programs, mainly by virtue of a built-in editor facility similar to those in POP-10(11,12,13) and some LISPs.

TONAL is based on experience with POP-2, POP-10, LISP and POPLER(14) together with ideas from PL/I, ALGOL 68, SIMULA, EUCLID(15), ECL (5,6,7), SCHEME(16) and other sources. Apart from POP-2, the debts to ECL and EUCLID will be particularly obvious. Some detailed suggestions come from (17). In keeping with the advice of Hoare(18) to language designers, TONAL does not contain any untried new ideas, but does provide as an integrated set of resources a combination different from those previously avialable.

2.0 OBJECTIVES

The primary objectives of the TONAL design are as follows.

1. SIMPLICITY - the language should be understandable, without obscure or irregular features.

2. INTERACTIVE USE - in a reasonably economical implementation. Interactive, incremental construction and testing for programs should be possible.

3. HELPFUL - the language should provide useful tools for program development, and give understandable error diagnostics(19).

4. STRUCTURED But FLEXIBLE
The language should encourage coherent organization of programs, partly through textual scoping for names, and also by providing for variable modes. It should also encourage (so far as possible) adequate documentation and commenting of programs.
On the other hand, the language system must be flexible for interactive incremental use, and the language syntax should not be overelaborate.
These considerations are in conflict to some extent, and the language design must balance them so neither is pre-eminent.

5. Similarity to POP-2 when other considerations do not take precedence.

6. PATTERN MATCHING facilities must be present in an extensible form. This is integrated with the mode (type) system.

7. RE-ENTRANT RECURSIVE COMPILER as in POP-2. There should also be a mechanism for reading a unit of program text (an expression, etc.) from an input stream.

8. A DATA-BASE and DEMONS must be present in a primitive extensible form. The 'context' mechanism is to be tied to program block level. That is, demon procedures and other data-base entries are 'added' with respect to a particular program block -- usually the currently running block or the equivalent to variable access scopes.

9. GENERALIZED CONTROL facilities must be provided. This includes a basic capability to run multiple processes 'simultaneously', with interprocess coordination and message-passing facilities defined. Co-routine facilities are also incorporated, and a more general capability for 'saving states' is included to permit unusual control regimes to be implemented.

10. Good Debugging Tools - there must be adequate facilities for catching errors and introducing break-points in a program. It should be possible to 'reach' into the running code and stack to examine variable bindings, etc., and also to continue or restart a computation after an error.

    A general ability to take 'traps' on a variety of conditions should be included, and integrated on the one hand with error handling, and with the 'demon' system on the other.

11. Efficient Code Generation.
When variables are given modes (types), this should permit more efficient code to be generated, by reducing the degree of checking mandatory at run time. Full typing is not required, however, since that conflicts with general ease of use. When variables are left un-typed, this merely puts more burden on the run time system.

3.0 DISCUSSION OF OBJECTIVES

3.1 Simplicity And Interactive Use

These attributes are possessed by both LISP and POP-2. The language is based on a one-pass compiler rather than an interpreter. The syntax is based on POP-2, but is modified: (i) to deal with identifier modes, and (ii) to permit the compiler to determine which expressions are arguments of which functions and operations.

The language may be regarded, from one point of view, as a modification of POP-2 to clarify the usage of the stack by giving each function activation its own private stack. Among other consequences, every function call in TONAL returns a single result item as its value (cf. LISP), and all formal parameters of functions must be declared explicitly. There is a mechanism for variadic functions. This change from POP-2 makes the programming of Jumpouts more straightforward, and permits a program to be restarted in the middle of execution after an error has occured.

Interactive use is provided by having the system execute all imperatives (statements and declarations) as soon as they are typed in, except for those forming part of a function definition. A function definition is itself a kind of declaration, and is executed as soon as it is typed in.

3.2 Helpful System

The system should provide useful tools for program development. The principal tools are a built-in editor, macro and Abbreviation facilities, and various debugging and tracing tools. A capability for metering program performance is also provided.

The editor is based on the earlier POP-2 '77' editor and the POP-10 editor. These were modelled on TECO (20), but with the addition of an UNDO command. The TONAL editor is extended also to handle multiple buffers, somewhat as in QEDX (21).

The editor can handle arbitrary text files, but has specialized operations to facilitate the management of TONAL program files. All editor commands are functions and operations, which manipulate the text buffers, so programs can be constructed using the normal TONAL language to perform various complex tasks. There are special commands which 'know about' the syntax of TONAL, and programs can be compiled directly from a buffer.

A macro facility is provided as in POP-2, to permit the TONAL language syntax to be extended with new constructs. In addition, there is an Abbreviation facility specifically to permit abbreviations to be created for command strings. The first word in any comment at execute level, after a newline, semicolon or

'print arrow' "=>", can be checked in a hash-coded table for a stored expansion.

For example, the command
    #ab n lml, vc;
will define n as an abbreviation for the rest of that line. (Typing n at the start of a command will make the editor move to and print the next line of text.) #off and #on turn the facility off and on. This is modelled on a facility in the Multics system (21).

The debugging tools are described later.

3.3 Structured But Flexible

TONAL uses essentially the same scope rules for identifiers as Euclid (15). That is, lexical or 'static' scopes are used, but identifiers must be 'pervasive' or specifically imported to be used 'free' in interior scopes. These rules encourage a block-structured type of programming, with the benefit that variable and function identifiers are not made more global than they have to be.

Normally, this would make it harder to debug interactively the interior modules of programs, so special debugging tools are provided as described later.

Identifiers of a program, as part of this mechanism, may be declared as constants or variables, and may be given modes (types). These facilities make programs more specific, which has two main benefits. First, they perhaps become more understandable, and the compiler and run-time system have more chance to detect programming errors. Also, the compiler may be able to produce more efficient code, especially for arithmetic and structure accessing.

However, the typing of variables is not mandatory, and they default to the universal mode any.

Modes are themselves items, as in EL1(6). However, new modes are created by declarations, not by applying functions, because creation of a new mode usually involves declaring several associated operations simultaneously. There are built-in mechanisms for creating new 'record' and 'strip' modes ('structures' and 'rows' in ALGOL-68) and arrays; other 'derived' modes may be created with user-defined operations.

The language syntax is designed to be parsed by an LALR parser (19,22), and is more rigid than that of POP-2 in some respects. However, as far as possible, comma and semicolon separators are optional. For example, a semicolon may sometimes be omitted at a newline.

The following constructs are available for conditional and iterative execution:

```
IF cond THEN .. {ELSEIF ...} [ELSE ..] CLOSE
WHILE cond DO ... ENDDO
REPEAT ... [UNTIL cond] ENDREP
CASE expr  { >: pattern :> clause }
                  [ELSE ..] ENDCASE
FOR { id initialValue, expr;} UNTIL cond
           DO ... ENDFOR
```

The first three are conventional. The CASE statement evaluates an expression, and matches it with 'patterns' until it finds a match, when it executes the appropriate clause. In fact the pattern matches are normally compiled 'open' (see below).

The FOR is similar to the MacLISP DO (23) and can initialize and step several variables in parallel in almost any type of sequence. The program

```
FOR x 1,x+1 UNTIL x>10 DO pr(x) ENDFOR
```

prints the integers 1 to 10.

Labels are not permitted, and a restricted GOTO is provided to escape from or restart an iteration or function. Function calls can also be escaped from dynamically with a Jumpout as in POP-2, or by a CHAIN facility.

TONAL, like POP-2, does permit a function created in one context to be passed as result and later applied in another context. When this happens, the function item must internally be a 'closure' on the access environment needed for its 'free' identifiers. The position taken on this is that all function items are automatically closures on the environment current when they were formed. This is similar to the SCHEME language (16). The TONAL compiler can avoid the construction of closures which are not needed explicitly.

If such a closure is saved for later application, this can tie up memory in saving the access environment. The compiler can tell when all 'free' identifiers are bound at the global level and avoid this waste of memory. Partial Application is provided as in POP-2 for those (frequent) situations where a closure needs to be 'read only'.

3.4 Patterns

A system for using patterns is introduced. This is an extension of the proposal for POP-2 (17). A pattern is primarily a convenient way to express a complex test on a value, and may be used in a CASE statement, or in IS expression. The latter follows the syntax
    expression IS pattern
and evaluates to a truth-value. Patterns may have the syntactic forms:

form	meaning
constant	EQUAL to the constant
=identifier	= value
identifier	assign to variable
:mode	check item mode
.predicate	apply predicate function
[!pattern,..!]	list pattern--check elements in turn
fn(pattern,..)	test item with structure/ component matching.
-	match anything

In particular, expressions such as x is :list and n2 is :num are the best way to check

the mode of an item.

Many pattern matches are compiled 'open' for efficiency. Nevertheless, patterns may also be represented by pattern items, and interpreted later by the standard function match_pattern. This is needed for the demon system (Section 3.6).

3.5 Reentrant Recursive Compiler

This is similar to POP-2 except for the lexical scoping of identifiers. When the compiler is called, its initial scope may be set to be 'global', or to be at the environment of the point of call. This affects the interpretation of 'free' identifiers in that call of the compiler.

In particular, when an error or interrupt occurs, the compiler operates at the point where execution was stopped, permitting local identifiers to be examined, and programs for internal blocks to be modified and recompiled. This does give an air of 'dynamic' scoping to the system.

Because the syntax of TONAL expressions and statements is more closely defined than in POP-2, a facility is provided for reading, from an input, text which forms an expression. This is comparable to the LISP ability to read an S-expression in one call.

3.6 The Database And Demons

Because of the variety of different requirements for data-base and demon-type facilities that AI projects have developed, a minimal extensible facility is provided. No automatic backtracking or any other form of demon invocation takes place without the user specifically programming it. The intention is to provide the 'raw materials' required as data structures, etc., but for the user to decide how they should be combined. The system does enforce the rules limiting access to data.

In TONAL, a Demon is like a POPLER Procedure, a Planner Theorem or a Conniver Method. It is a function with associated pattern; the function itself takes only one parameter, which is matched with the pattern, and the body is only executed if the pattern matches. The pattern match may assign values to locals of the function body. The pattern is also accessible as a 'component' of the demon, so demons can be indexed and selected on the basis of their patterns.

A 'data-base' facility is provided which can store demons or other items, and retrieve them on the basis of pattern matching. The data-base is divided into Demon and Item Classes, so they can be grouped according to use. Initially, there are no classes declared, but to simulate Conniver for example, one might include the following statements in a program:

```
set_item_class("item");
set_demon_class("if_added");
set_demon_class("if_needed");
set_demon_class("if_removed");
```

The argument words "item" etc., are not variables, but just 'handles' used internally in the data-base; presumably they will be descriptive names.

Also, the data-base has its contents saved with respect to various Contexts, again somewhat as in POPLER or Conniver. The special feature here is that a Context always corresponds to a lexical variable scope. This scope is often the global level or a Section (a self-contained module), but it may be just a particular function or demon activation. A dbcontext is identified by placing a special declaration such as

```
dbcontext bags;
```

in the program module concerned. That will declare bags as a variable containing essentially a label for the activation record (or whatever) for the function or section. That database context will only be usable when the module declarations are in scope. (This rule will be enforced at run-time, if an attempt is made to pass the value of bags outside its scope.) the standard identifier gdb is bound to a dbcontext for the global level.

Objects are added to the database and removed again with the primitive operations

```
add_item(item, dbcontext, classHandle)
add_demon(demon,dbcontext, classHandle)
```

and erase_item and erase_demon.
The functions get_items and get_demons are used to retrieve from the data-base, and each takes a pattern as argument and returns a list of items.

```
get_items(pattern, dbcontext, classHandle)
```

Adding or removing items with these functions does not activate any demons or perform any other side-effect. The user who wishes to simulate Planner or Conniver, etc. will write his own functions ADD and REMOVE (for example) which perform all the necessary actions and searches, and invoke demons as necessary.

The TONAL system does not automatically apply demon functions for the user at any time. A list of demons can be retrieved as summarised above, and the user is responsible for deciding how and when to apply them. The ground rule for demons, however, is that if they return false then they are taken to have 'failed'. There is no automatic backtracking in TONAL, though it can be implemented.

In summary, the main intention in TONAL is to permit and encourage the programmer to localise information and demons, without preventint them from being global when this is actually needed.

3.7 Generalized Control Facilities

The main requirement for generalization is the ability to handle multiple processes simultaneously, or as co-routines. Co-routines are regarded as multiple processes with the user programming the context switching explicitly. In the case of multi-processing with the system handling context switching (or even multiple

processors) the need arises to define the synchronization mechanism.

Although Monitors (25) are currently popular for this, we have chosen to use Message Buffers as the primary mechanism. It is well known that semaphores and monitors can be implemented using message buffers, and they also provide a simple inter-process communication facility.

A new process is formed from a function and arguments with consprocess:

consprocess(function,list)

This returns a process item. The process may then be started as a co-routine with run or resume, somewhat as in SIMULA (26) and the new POP-2 (17), or with sprout to run in parallel. The process is deleted when the function exits, and its final value is lost unless the function sends it to another process (preferably through a message buffer).

A message buffer obeys a FIFO queue discipline, and holds 0 or more items, up to its capacity. It is created with consmbuff, and filled and emptied with putmbuff and getmbuff.

putmbuff(item,mbuffer)
getmbuff(mbuffer) => item

Those latter operations block if the buffer is already full or empty respectively.

These mechanisms are related to those of SAIL(2) and ABSET (27), but differ somewhat from TELOS. The details are based on code for POP-2 (17) recently transplanted to POP-10.

TONAL discourages 'naive' backtracking of the Planner/POPLER kind, but this can be programmed if it is really desired, by using a saved-state mechanism similar to that of POP-2 (3). Alternatively, a similar effect can be obtained using 'teams' of co-routines, and this has the advantage that the local environment of a process is not destroyed if it stops.

3.8 Tools For Debugging

The principal debugging tool is the editor, which can contain the text of program files being worked on. Programs can be compiled directly from a buffer and changed interactively, as described in (12,13). This is similar to the approach used in LISP systems for in-core editing of S-expressions (4).

The main problem with a block structured language is that normally when an inner function definition is changed all the enclosing scopes have to be recompiled too. The TONAL compiler will have the capability to recompile inner function definitions without having to recompile the whole enclosing scope.

An interactive tracing and 'break' package is also provided, based on that in POP-10. This permits trace messages and 'interrupts' to be programmed flexibly, so that the state of the program during execution can be monitored. Also as in POP-10, after an error, a 'break' is entered again, and it is possible to modify and recompile a function, and then to restart the computation in the middle instead of having to begin the whole program run again.

It is proposed to provide a flexible system for handling errors and 'traps' of various sorts, by setting up a special Demon class "system-trap", with patterns predefined for the various conditions to be handled. This is similar to the 'event' mechanism of TELOS, and provides also the capabilities of PL/I On-conditions.

Other useful features are provided by the system. With a block-structured language it is desirable to know which identifiers are declared and imported at each level. Depending on the setting of a control variable, the compiler will print an information message at the end of compiling each function and section, which will give more or less detail about the identifiers used.

If an identifier is met in a scope which has not been declared, it will automatically be declared as a variable of mode any, local to that scope, and a warning is printed.

4.0 SUMMARY

A new language system TONAL is being designed for AI applications. It is compiler oriented and block structured, with emphasis on both abstraction mechanisms and on the ability to restrict the availability of information to those contexts where it is required. The system is also designed to facilitate interactive program development and debugging.

Appendix A contains a short program to illustrate the syntax. No implementation of TONAL is complete, and no large programs have been written yet in TONAL. A language Manual is available, but the specification remains subject to amendment in the light of experience.

5.0 ACKNOWLEDGEMENTS

This work was supported in part by the Natural Research Council of Canada. It would not have been possible without the experience gained at Edinburgh working in POP-2 and implementing POPLER. The graduate students and my colleagues at Western have offered welcome encouragement.

6.0 REFERENCES

1. J McCarthy, P W Abrahams, D J Edwards, T P Hart and M I Levin; LISP 1.5 Programmers' Manual. MIT Press, Cambridge, Mass. (1962).

2. J A Feldman, J R Low, D C Swinehart and R J Taylor; Recent Developments in SAIL, an Algol-based Language for AI. Stanford AI Memo-176, STAN-CS-308 (Nov 1972).

3. R M Burstall, J S Collins and R J Popplestone; Programming in POP-2. Edinburgh Univ. Press (1971).

4. E Sandewall; Programming in an Interactive Environment: the "Lisp" Experience. ACM Computing Surveys 10 (Mar 1978) pp.35-71.

5. B Wegbreit, B Brosol, G Holloway, C Prenner and J Spitzen; ECL Programmers' Manual. Center for Research in Computing Technology, Harvard, Cambridge, Mass. (Sept 1972).

6. B Wegbreit; The Treatment of Data Types in EL1. Comm. ACM 17 (May 1974) pp.251-263.

7. --; Procedure Closure in EL1. Computer J. 17 (1973) pp.38-43.

8. L Travis, M Honda, R LeBlanc and S Zeigler; Design Rationale for TELOS, a PASCAL-based AI Language. Proc Symp on AI & Programming Languages, ACM SIGPLAN Notices 12,8 (Aug 77), ACM SIGART Newsletter #64 pp.67-76.

9. D G Bobrow and T Winograd; An Overview of KRL, a Knowledge Representation Language. Cognitive Science 1 (Jan 1977) pp.3-46.

10. L F Melli; Experiences of Programming Languages for Artificial Intelligence Research: a Case Study. INFOR 15 (Feb 1977) pp.107-129.

11. D J M Davies; POP-10 User's Manual. Computer Science Report 25, UWO (June 1976).

12. --; POP-10 System Editor. Computer Science Report 26, UWO (June 1976).

13. B Boyer, J Moore and J Davies; The 77-Editor. DCL Memo 62, School of AI, Edinburgh (Feb 1973).

14. D J M Davies; POPLER 1.5 Reference Manual. TPU Report 1, School of AI, Edinburgh (May 1973).

15. B W Lampson, J J Horning, R L London, J G Mitchell and G L Popek; Report on the Programming Language Euclid. ACM SIGPLAN Notices 12,2 (Feb 1977).

16. G L Steele Jr. and G J Sussman; The Revised Report on SCHEME: a Dialect of Lisp. MIT AI Memo-452 (Jan 1978).

17. R M Burstall and J Scott; Proposals to Enhance POP-2. (Unpublished) Department of AI, Edinburgh (1977).

18. C A R Hoare; Hints on Programming Language Design. Stanford AI Memo-224, STAN-CS-73-403 (Dec 1973).

19. W M McKeenan; Programming Language Design. In Compiler Construction; an advanced course. (Eds F L Bauer and J Eickel) Springer Verlag, 2nd Edn. (1976).

20. Digital Equip. Corp.; TECO Reference Manual. DEC-10-ETEE-D.

21. Honeywell Information Systems; Multics Programmers' Manual: Active Functions and Commands. AG-92 (1977).

22. A V Aho and S C Johnson; LR Parsing. ACM Computing Surveys 6 (June 1974).

23. D A Moon; MacLISP Reference Manual, Revision 0. MIT Lab. for Computer Science, Cambridge Mass. (1974).

24. D V McDermott and G J Sussman; The CONNIVER Reference Manual. MIT AI Memo-259 (May 1972).

25. C A R Hoare; Monitors: an Operating System Structuring Concept. Comm. ACM 17 (1974) pp.549-557.

26. G M Birtwistle, O-J Dahl, B Myhrhaug and K Nygaard; SIMULA Begin. Auerbach Press, Philadelphia, Pa. (1973).

27. E W Elcock, J M Foster, P M D Gray, J J McGregor and A M Murray; ABSET: a programming language based on sets; motivation and examples. in Machine Intelligence 6 (eds. B Meltzer and D Michie), Edinburgh Univ. Press (1971) pp.467-92.

APPENDIX A

EXAMPLE PROGRAM

```
SECTION SETS_FACILITY
      EXPORTS NILSET NULLSET ADDSET
             MEMBSET DELSET SET;

\define a SET facility.
\ a SET has a list and an = predicate
\NILSET(pred) makes an empty set
\NULLSET tests whether a set is empty
\ADDSET(x,set), DELSET modify a set
\MEMBSET(x,set) tests a set membership

TYPE PERVASIVE SET =
  RECORD CONSGENSET, APPSET
    (SL:LIST, SEQ:FUNC) ENDRECORD;
      \the operators are all pervasive
      \ like SET, but are not exported.


OPERATION 1 PERVASIVE NILSET EQP => :SET;
  CONSGENSET(NIL,EQP) END;
      \the result mode may be specified

FUNCTION PERVASIVE NULLSET S:SET;
  S.SL.NULL; END NULLSET;
      \END may be followed by the name
      \if it does not match - a warning

FUNCTION PERVASIVE MEMBSET X, S:SET => :BOOL;
      VARS L:LIST, EQF:FUNC;
  FOR L S.SL, L.TL;  EQF S.SEQ
      \step L, just initialise EQF
    UNTIL L.NULL
    DO  IF EQF(X,L.HD) THEN TRUE EXIT;
  ENDFOR;
  FALSE;    \not in list - return FALSE
END MEMBSET;


FUNCTION PERVASIVE ADDSET X,S:SET => :SET;
  IF MEMBSET(X,S) THEN S
  ELSE CONSGENSET(X::S.SL, S.SEQ)
  CLOSE;
END ADDSET;


FUNCTION PERVASIVE DELSET X, S:SET => :SET;
  VARS EQF:FUNC;
      FUNCTION GENDEL X L; IMPORTS EQF;
        IF L.NULL THEN NIL
        ELSEIF EQF(L.HD,X) THEN L.TL
        ELSE L.HD::GENDEL(X,L.TL)
        CLOSE;
      END GENDEL;

  S.SEQ->EQF;
  IF MEMBSET(X,S)
      THEN CONSGENSET(GENDEL(X,S.SL),EQF)
      ELSE S
  CLOSE;
END DELSET;


ENDSECTION SETS_FACILITY;
```

EXAMPLES OF COMPUTATIONS AS A MEANS OF PROGRAM DESCRIPTION

Michael A. Bauer
Department of Computer Science
University of Western Ontario
London, Ontario, Canada
N6A 5B9

ABSTRACT

The following describes an approach to the synthesis of procedures from examples of computations. An example computation is basically a sequence of instructions obtained from the "execution" of an algorithm on some input. Unlike previous work on this problem, some flexibility in the description of examples is permitted. The synthesis algorithm, in turn, relies on knowledge of variables and instructions to construct a procedure.

INTRODUCTION

A programmer's assistant can involve various facilities to aid a programmer. These facilities might include a language understanding subsystem or a subsystem to assist in program debugging. An assistant might also include a facility for program synthesis. Input to such a synthesizer might be natural language [8,9,11], input/output predicates [10] or input/output pairs [6,7]. Another form of input to a synthesizer might consist of sequences of instructions describing in a step-by-step manner the execution of a particular algorithm on specific inputs. Such descriptions might be the sole form of input to a synthesizer or might provide additional input to a system in which a program was initially described in natural language or specified by input/output predicates.

In this paper we describe some initial work on a "specialist" capable of synthesizing a procedure from examples of computations. This work has concentrated on the development of a synthesis algorithm, given a suitable representation of an example, rather than on the formation of representations from natural language input. The motivation for such an approach is twofold. First, in order to synthesize a procedure, some underlying synthesis algorithm must be available regardless of the particular input format of the examples. Of course, it is reasonable to rely on natural language descriptions of examples to motivate the characteristics of the examples we wish to study. Second, by concentrating on the algorithm we can investigate the role of knowledge about procedures during the synthesis process.

The current "specialist" involves no knowledge of what the intended program is to compute. Rather, it uses certain "common sense" knowledge about procedures, instructions, assignments, variables, constants, etc. Much of this knowledge is in the form of constraints used during the construction of a procedure.

The problem of synthesizing procedures from example computations has also been considered by Biermann [3,4]. In his work, an example of a computation was essentially a sequence of symbols. Little variation between examples was permitted, requiring, for example, the same instruction in two different examples of the same procedure to be identical. Our work is an extension of Biermann's permitting more flexibility in examples. This has required us to incorporate some knowledge of procedures, examples and their components into the algorithm.

CHARACTERISTICS OF EXAMPLES AND PROCEDURES

To motivate certain aspects of such examples and our model of procedures, let us consider a description of an example computation in natural language. Figure 1 illustrates an example of a procedure to perform an interchange sort on a 1-dimensional array.

To sort the array (6,3,5) of 3 elements proceed as follows:

Let A be the array (6,3,5).
Let X be the 6.
Let Y be the 3.
Since X is greater than Y, interchange elements 1 and 2 of A.
Then let X be 3.
Now compare X and element 3 of A.
Since X is not greater than 5, do nothing.
Next, let X be the second element of A, which is now 6.
Compare X and the third element of A.
Since X is greater than 5, interchange elements 2 and 3 of A.
Let X be 5.
Finally, let X be the third element of A.
Since there are only 3 elements in A, we stop.

Figure 1: A Plausible Description of an Example Computation

This example suggests several characteristics of a model of example computations. First, it must involve variables, assignments, functions,

predicates and procedures. The model should also include some facility for composite objects, such as arrays or records. Examples from the same procedure should not be constrained to a single set of variables. It should also be possible to use the inputs, intermediate values of variables and components of composite objects within the example itself.

These characteristics of examples, in turn, suggest that the procedures synthesized must also involve similar instructions and must have parameters.

A MODEL OF EXAMPLES AND PROCEDURES

The instructions we shall consider fall into four classes:

1. <u>Assignments</u>: There are three forms - assigning to a variable, a number of variables or a variable whose value is a composite object:
    1. <u>Simple Assignment</u>: $X \leftarrow t$, where t is a variable, a constant or $f(v_1,\dots,v_n)$, and the v_i are either constants or variables. We call $f(v_1,\dots,v_n)$ a <u>Function Application</u>.
    2. <u>Multiple Assignment</u>: $\langle X_1,\dots,X_n\rangle \leftarrow P(v_1,\dots,v_m)$, where the v_i are constants or variables and P is a procedure returning n values. We call $P(v_1,\dots,v_m)$ a <u>Procedure Call</u>.
    3. <u>Updating Assignment</u>: $U(X,\langle v_1,\dots,v_m\rangle) \leftarrow t$, where t and the v_i are as above; U is an updating function.
2. <u>Predicates</u>: $p(v_1,\dots,v_n)$, p is an n-ary predicate; v_i as above.
3. <u>Termination Statements</u>: $Return(v_1,\dots,v_n)$; v_i as above.

The right hand side (rhs) of an assignment may be a variable, constant, function application (including updating functions) or procedure call. We treat procedures as functions which return an n-tuple of values. We assume that all actual parameters are passed by value and, hence, any side effects must be explicitly done by assignments or updaters. Predicates simply evaluate to true or false.

Updaters are functions which operate on composite objects to change or retrieve components. An updater has two arguments - a composite object and an n-tuple defining a component of the object. As such, an updater may appear on either side of an assignment. On the rhs, the value of a component of the composite object is returned as the value of the updater; on the lhs the component is altered, essentially forming a new composite object (see POP-2 [5] for the use of updaters in an existing programming language).

Finally, a termination statement is used to indicate the end of a procedure and defines the n values to be returned.

This set of instructions provides the basis for a rich class of examples and procedures. In particular, the use of procedures and updaters facilitates examples of non-trivial programs. Notice that the statements in our natural language description of an example are not confined to the syntax of our instruction classes. Rather, we view these classes as a target into which higher level descriptions are to be translated.

An <u>example of a computation</u> is a pair $(P(a_1,\dots,a_n),T)$, where P is the name of the procedure, $a_1,\dots,a_n$ are the arguments used to form the example and T is a directed tree of instructions. Each tree has a single termination statement, a leaf. Other leaves in the tree are predicates which evaluated to False. The successors of a node are ordered (counter-clockwise in figures). The sequence of instructions executed in the example begins with the root and proceeds in order to its successors.

This particular representation was adopted for a number of reasons - avoidance of True/False labels (as in flowcharts), representational convenience, a means to include examples from backtracking procedures. For examples from backtracking procedures, leaves are instructions which failed.

Given a number of examples, the synthesis algorithm attempts to form a procedure. Procedures, in turn, are represented as a pair $(P(X_1,\dots,X_n),D)$, where P is the name of the procedure, $X_1,\dots,X_n$ are its formal parameters and D is a rooted, directed graph of instructions in which successors of a node are ordered. Execution proceeds from the root in a depth-first manner. Should a predicate fail (or in the case of a backtracking procedure, instruction fail), the next successor of its predecessor is executed. If all successors of a node evaluate to False (or fail), execution proceeds to the next successor of its predecessor (see [1] or [2] for a more detailed description of such executions). The interchange sort is illustrated in Figure 2 and two example computations from such a procedure are illustrated in Figure 3.

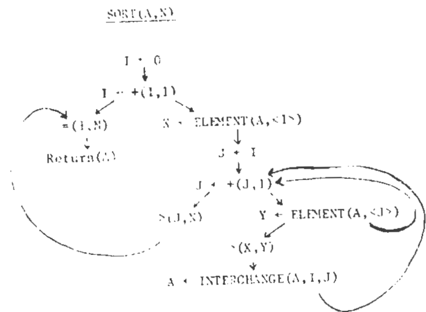


Figure 2: A Sort Procedure

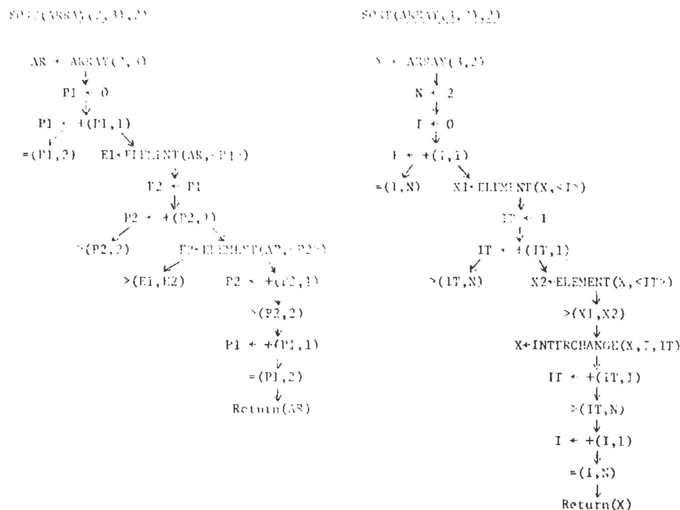


Figure 3: Examples from Sort Procedures

From a formal viewpoint (see [1]) we can talk of a pure example computation as one which originates from a procedure by executing the procedure on some input and simply recording the sequence of instructions executed. An acceptable example computation is one obtained from a pure one via a number of transformations. The transformations corresponding to the characteristics cited above are: a) renaming a variable, b) replacing occurrences of a parameter by its input value and c) adding an assignment to the example in which a parameter is assigned its input value. The class of examples the algorithm accepts is then the set of pure examples closed under the transformations.

Such a methodology is particularly useful. First, it enables one to define what one means by an acceptable example. Knowing the class of examples is important since what one can reasonably consider to be an example can vary greatly. This methodology also lends itself to possible extensions, since one need only consider additional transformations. Second, it provides a mechanism to isolate certain subproblems within the overall synthesis problem. This may not only suggest certain techniques for solution but may also suggest what information is necessary. Finally, we should note that such a methodology is not arbitrary - it was certain characteristics of examples, as in our natural language description, which motivated the particular transformations.

THE SYNTHESIS ALGORITHM

Informally, the synthesis problem is to form a procedure, represented as a directed graph, from a number of example computations. By relying on the model of procedures, a definition of what constitutes an acceptable example and the transformation, this problem can be made precise. We shall have additional comments about this aspect of our work in the concluding section. Currently, we shall concentrate on a description of the synthesis algorithm and the information it uses.

Let us briefly examine how a person might form a procedure from a number of example computations. The first task might be to locate instructions which appear to do the same thing. Once a number of possibly related actions have been located, the person attempts to verify the relationships by examining the variables and constants involved, how corresponding variables are used, what instructions follow, etc.

Our synthesis algorithm proceeds in a similar manner. Given a number of examples, the algorithm attempts to group nodes into classes. Nodes in the same class are, supposedly, occurrences of the same instructions in the algorithm being illustrated. Once a set of classes has been hypothesized, the set is examined to guarantee that certain consistency conditions are satisfied. Knowledge of variables, constants, instructions and procedures is embedded within these consistency conditions.

Intuitively, we can class two instructions together if they seem to be occurrences of the same instruction in the algorithm being described. Occurrences of an instruction may be obtained by renaming variables within the instruction or replacing those variables which are parameters by their input values. Such substitutions must be consistent with certain "common sense" conditions:

-- A variable can only be replaced by at most one other variable in any other example.

-- A variable, which is also a parameter, can be replaced by at most one input in any one example.

We shall define variations of an instruction under such substitutions. This, in turn, will provide a basis for determining when instructions can be grouped together. Let us define a substitution σ to be a set of pairs $(W_i | t_i)$ where:

1. W_i is a variable; t_i is a constant or variable.
2. No two W_i are the same in σ .
3. No two t_i , which are variables, are the same in σ .
4. If $(W_i | t_i)$ and $(W_j | t_j)$ are in σ and $W_i = W_j$, then $t_i = t_j$.

We apply a substitution $\sigma = \{(W_i | t_i)\}$ to a predicate termination statement, left or right hand side of an assignment by simultaneously replacing each occurrence of W_i by t_i .

Assignments introduce special problems since a variable being assigned may only be renamed by another variable and not replaced by an input on the left hand side of the assignment. Also, it is possible that a variable may be both renamed and replaced within a single instruction. Consider the instruction $X \leftarrow +(X,1)$. If X is a parameter, then in an example in which X received an initial value of 3, a reasonable variation of this instruction might be $Y \leftarrow +(3,1)$. As a result, the concept of a substitution is only partially satisfactory.

A mapping π for an instruction s is:

1. A substitution σ if s is a predicate or termination statement.

2. A pair of substitutions (σ_L, σ_R) if s is an assignment where:

   1. For all $(W_i | t_i)$ in σ_L, W_i assigned to in s, then t_i is a variable.

   2. If $(W_i | t_i)$ is in σ_R, t_i is a variable and $(W_i | t_j)$ is in σ_L, then $t_i = t_j$.

   3. If $(W_i | t_i)$ is in σ_R, t_i is a variable and $(W_j | t_i)$ is in σ_L, then $W_i = W_j$.

A mapping π for an instruction s <u>applied to s</u> $(\pi \circ s)$ is:

1. $\sigma \circ s$, if $\pi = \sigma$.
2. $\sigma_L \circ s \leftarrow \sigma_R \circ s = s'$, if $\pi = (\sigma_L, \sigma_R)$.

Finally, let $s_1,\ldots,s_n$ be instructions, $\pi_1,\ldots,\pi_n$ be maps for $s_1,\ldots,s_n$ and let s be an instruction. Call s a <u>generalization</u> of $s_1,\ldots,s_n$ if $\pi_i \circ s = s_i$. Call s a <u>least generalization</u> of $s_1,\ldots,s_n$ if for any generalization s', there is a mapping π' such that $\pi' \circ s' = s$.

The generalization, if it exists, of a set of instructions captures the intuitive idea of an instruction from which others might have been formed. It also a) forms the basis for constructing an instruction from a set of instructions and b) through the mappings constructed, provides additional information which can be examined with respect to certain "common sense" rules. Note that this concept of generalization is similar to that of Plotkin's [11], but because of the nature of instructions, and in particular assignments, the ideas had to be extended.

Now, given a set of instructions, $s_1,\ldots,s_n$ we say that they <u>look alike</u> if:

1. They have a least generalization.
2. Corresponding sets of successors look alike.

In Figure 4 nodes 1 and 5 look alike since their sets of corresponding successors $(\{2,6\},\{3,7\})$ look alike (assuming that $\{3,7\}$ look alike). Nodes 1 and 8 have a least generalization but do not look alike since the set $\{3,10\}$ of corresponding successors do not have a least generalization, i.e. do not look alike. Note that requiring instructions to have identical successors or identical numbers of successors is too severe since they might be occurrences of the same instruction involved in computations taking slightly different paths.

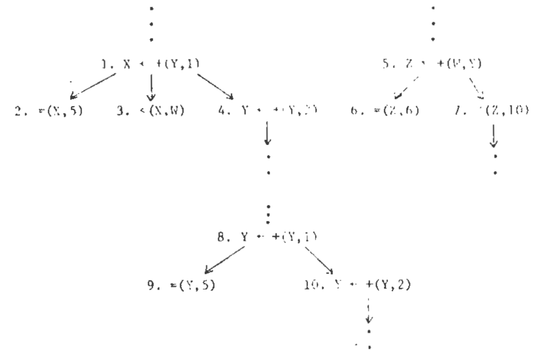


Figure 4: Examples of Nodes Which "Look Alike"

However, just defining what constitutes a single class of nodes is insufficient. One must also define relationships between the classes dictated by the successor relationships between individual instructions. There is also the problem of how to determine reasonable sets of classes, or perhaps more realistically, how to reject unreasonable ones.

This latter problem can be simplified by using <u>key nodes</u>. Intuitively, key nodes are instructions within the examples which involve specific functions and probably coincide or ones which are a priori known to coincide. Key nodes provide an anchor on which to base class formation by providing knowledge of which nodes may have originated from the same instruction in the algorithm. In our case, the key node in each example is determined by first finding the first node on the leftmost path from the root involving a function, predicate, procedure call or termination statement. Then one proceeds in parallel back toward the root in each example until all nodes not yet considered have single successors. Since we have assumed that no instructions are omitted from any example, these must originate from the same node of the algorithm.

Given a set of nodes $S = \{s_1,\ldots,s_n\}$, let us define the set of ith corresponding successors of these nodes to be $COR(i,S)$. Then a <u>cover</u> for examples $E_1,\ldots,E_t$ is a set of sets $C_1,\ldots,C_m$ such that:

1. The key nodes are in the same C_i.

2. Any node reachable from a key node is in some C_i.

3. If ℓ is the maximum number of successors of any node in C_i, then for each j, $1 \le j \le \ell$, $COR(j, C_i)$ is contained in some $C_{i'}$.

Because some additional (extraneous) nodes may have been added to the examples, we may not wish to include all instructions; hence we allow for the exlusion of some. In general, the synthesis algorithm seeks the cover containing the greatest number of nodes in the fewest classes. However, even those nodes not included in a class must satisfy certain constraints (below).

The two example computations in Figure 5 are from a procedure which examines a property list.

A property list is a list of sublists where each sublist contains a variable (first element) followed by its properties (e.g. numeric values). Given a property list and a property, the procedure returns the first variable having such a property. Given a variable (enclosed in quotes) the procedure returns the properties of that variable.

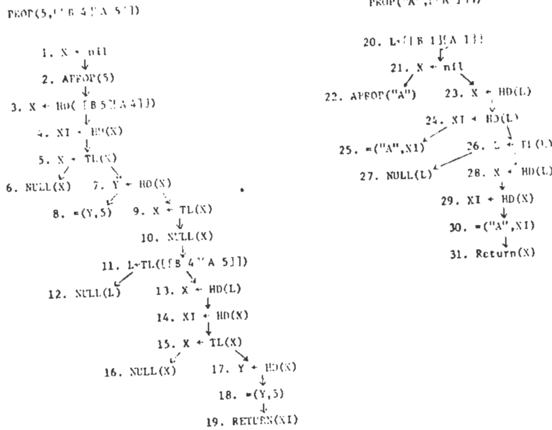


Figure 5: Two Examples of Property-List Algorithms

The key nodes are nodes 1 and 21. Since nodes 1 and 21 must be in the same class, by the third condition, nodes 2 and 22 must be in the same class. This condition, therefore, guarantees that the instructions of the synthesized procedures will follow paths based upon the instructions in the examples.

As one might expect, a number of possible covers do exist. In Figure 6, several covers are presented. Cover C1 is the trivial cover - only nodes necessary to satisfy the conditions are grouped together while remaining nodes are in individual classes. Covers C2 and C3 are acceptable, but do not make "sense". In C2 the instructions NULL(L) and NULL(X) are in the same class. The occurrence of two different variables suggest that these two instructions, in the same example, are used differently and should be kept distinct. In C3 instructions 8,18,25,30 are classed together. If 8 and 25 (=(Y,5) and =("A",XI)) are to be occurrences of the same instruction of the original algorithm, then Y is a renaming of a variable in that instruction and "A" is an input assigned to that variable. Examining node 7, X ← HD(X) , one sees that Y is assigned a value immediately before node 8 and so Y is probably not a parameter. Moreover, if "A" was the value of some parameter, then it was not the same one which Y renamed. In both cases, we wish to reject such partitions. This is done by a number of constraints. Cover C4 is the correct one.

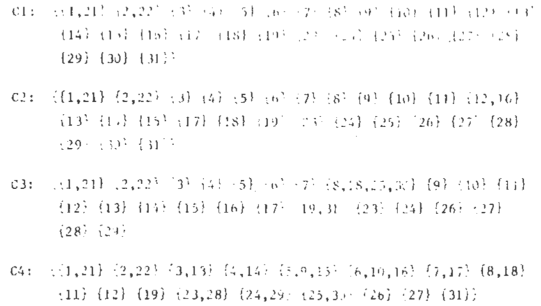


Figure 6: Examples of Possible Covers

KNOWLEDGE AS CONSTRAINTS

The formation of an acceptable cover for instructions within examples is the basic activity of the synthesis algorithm. Once completed, the following information is available:

:The Examples.

:The classes of instruction and the successor relationships between classes defined as: C_i is a successor of C_j if there exists n_i in C_i and n_j in C_j such that n_i is a successor of n_j .

:One instruction (the least generalization) has been associated with each class; their successor relationships are defined by the successor relationships among the classes.

:The mappings used to form each generalization.

As noted in Figure 6, in many cases the formation of classes based solely upon nodes looking alike leads to unacceptable covers. However, these covers can be eliminated by relying on knowledge of variables, instructions, etc. This knowledge is in the form of constraints or conditions on the classes, mappings and nodes not in any class. The failure of a cover to satisfy the conditions causes it to be rejected and the search for a new cover to be resumed.

We shall informally describe the constraints and the knowledge of procedures they embody.

-- Every variable in the synthesized procedure must correspond to at most one variable and/or one constant in any one example.

This condition applies to the mappings produced from the least generalizations. For example, the instructions NULL(X) and NULL(L) considered previously could not be classed together since the variable in their least generalization would have to correspond to both X and L in the same example.

-- Every variable in the synthesized procedure formed from a constant must become a formal parameter of the procedure.

This condition, in general, implies that if one introduces a variable during the synthesis process to replace a constant then it must do so correctly, i.e., the variable introduced must be a parameter. This involves three phases: a) the constant that is replaced in any one example must be in the input list of that example b) over all the examples, there must be an argument position common to any constant replaced by this variable c) any variable which the introduced variable, say W , renames cannot be assigned on the path from some node in a class to a node in which W replaced a constant. Any set of classes having a class containing nodes 8 and 25 of Figure 5 would be rejected because they would violate this constraint.

-- Any assignment not included in a class must be a node which has been added.

From the definition of our acceptable examples, nodes not included in the same class must be ones used to assign initial values to parameters. Such a node must be of the form X ← c . Either a variable W was used in a least generalization to rename X in the synthesized procedure or X has not been renamed. This means that W , or X in the case it has not been renamed, must satisfy conditions a) - c) above.

Note that this constraint is applicable only after the other constraints have been satisfied, since it requires the mappings to have been completed.

In Figure 5 node 20 would be such a node. Assuming that a variable V had been created to replace L in that example, then we would have to consider V to be a parameter which was assigned an initial value [[B 1][A 1]] in the second example. Presumably nodes 3 and 23 would have been already classed together and hence in the first example V would have corresponded to [[B 5][A 4]] . Since both are arguments occurring in the second argument position and node 20 is not in any class, V could become a parameter in the synthesized procedure.

-- The variables of the synthesized procedure which must be considered to be parameters, form a plausible parameter list.

The previous constraints have dealt with the validity of certain variables as parameters. This particular constraint guarantees that such variables can in fact form a parameter list - basically that each variable in the list is unique and occurs exactly once. Thus, for example, if two variables V and W of the synthesized procedure must be parameters but because of the constants they replace, must both be the first parameter, the cover would be rejected.

CONCLUDING REMARKS

The above algorithm has been implemented in POP-10 and a number of examples have been tried. The synthesis of a procedure to multiply two positive integers by repeated addition required two examples. The synthesis of an interchange sort was successfully completed using two examples in one case and one example in another. Interestingly, the synthesis using the single example required much more time than the synthesis using the two examples even though the number of instructions in both cases was nearly equal.

One goal of the work was to define the problem and formalize the algorithm. As a result, it has been possible [1] to prove two results about the synthesis algorithm. The first result shows that the synthesis algorithm is sound. That is, given a number of examples from the same procedure, the algorithm produces a procedure which, if executed on the same inputs as the examples, produces the same results. This guarantees that the algorithm is always faithful to the information within the examples. The second result shows that the algorithm is complete. Completeness guarantees that for any procedure, if one continues to give examples of the procedure, then eventually the synthesis algorithm will produce a procedure weakly equivalent to the original. Weak equivalence implies that the synthesized procedure will produce the same result as the original for any input on which the original halts.

Finally, even though the examples which can be dealt with are quite restricted and the knowledge of procedures is still primitive, the theoretical and pragmatic success of the synthesis algorithm suggests a number of interesting directions for future work. First, one might explore how a natural language front end could be interfaced with such a synthesis system. Second, one could try to extend the synthesis algorithm to a broader class of examples by considering more transformations, especially ones which are more domain dependent. For example, one might permit variables other than parameters to be replaced by their values or permit the direct use of components of composite objects. Similarly, one could investigate how specific knowledge about particular composite objects, e.g., arrays, might be incorporated into the synthesis algorithm. Hopefully, similar theoretical results about the synthesis algorithms for such extended classes could be obtained. Finally, one might explore how such synthesized procedures could be altered or "debugged" given new examples. Some work along these lines has already begun [1].

REFERENCES

1. M.A. Bauer; "A Basis for the Acquisition of Procedures", Ph.D. Thesis, University of Toronto, 1978.

2. M.A. Bauer; "Computation Graphs", Kybernetes, Vol. 5, 1976.

3. A. Biermann; "On the Inference of Turing Machines from Sample Computations", Artificial Intelligence, Vol. 3, No. 3.

4. A. Biermann, R. Krishnaswamy; "Constructing Programs from Example Computations", Technical Report OSU-CISRC-TR-74-5, Ohio State University Computer and Information Science Research Center, Columbus, Ohio.

5. R. Burstall, J. Collins, R. Popplestone; Programming in POP-2, University Press, Edinburgh, 1971.

6. C. Green, D. Shaw, W. Swartout; "Inferring LISP Programs from Examples", Fourth IJCAI, Tbilisi, USSR.

7. S. Hardy; "Automatic Induction of LISP Programs", AISB Summer Conference, 1974.

8. G. Heidorn; "English as a Very High Level Language for Simulating Programming", Proc. Symp. Very High Level Languages, 1974.

9. G. Heidorn; "Natural Language Inputs to a Simulation Programming System", Technical Report NDS-55HD72101A, Naval Postgraduate School, Monterey, California, 1972.

10. Z. Manna, R. Waldinger; "Toward Automatic Program Synthesis", Communications of the ACM, Vol. 14, No. 3.

11. W. Martin, M. Ginzburg, M. Krumland, R. Mark, B. Morgenstern, M. Niamir, B. Sunguroff; Internal Memo, Automatic Programming Group, MIT.

12. G. Plotkin, "A Note on Inductive Generalization", Machine Intelligence 5, ed. D. Michie.

In Defence of Syllogisms

by

Sydney J. Hurtubise

and

Rogatien "Gatemouth" Cumberbatch

Department of Issues, Distinctions, Controversies, and Puzzles
University of Artificial Intelligence
North Bay, Ontario*

Abstract

In this paper we present a serious study of syllogistic reasoning. We believe that powerful new developments in computer technology (such as the process metaphor, the type / token distinction, procedural semantics, pattern recognition, and the INTEL 8080 chip) add significantly to man's understanding of the universe and hence make it possible to break new ground in this classic philosophical endeavour. The paper itself provides the details, and we strongly urge all progressive AIers to read it - it would be a shame to miss out on one of the greatest pieces of research carried out since the advent of the space age.

1. Introduction

Recently there has been a renewal of interest in using tried and true logical methods for solving problems in artificial intelligence. What we propose here is going back to the original tried and true logical method: syllogistic deduction. There are a number of reasons for our decision. First, syllogisms are a nicely limited domain, approachable and understandable by anybody, even the common man. Second, syllogisms have been around so long they are probably by now cognitive primitives underlying all other reasoning. Third, we believe that researchers in AI should not look at more esoteric logics (e.g. propositional logic, predicate calculus, CONNIVER) until there has been a full understanding of earlier approaches. Moreover, much of the so-called power of advanced logical reasoning systems (e.g. resolution theorem proving) could trivially be achieved syllogistically.

In this paper we describe a model called SILLI which accepts as input two premises stated in natural language and produces a syllogistically valid natural language conclusion. We have chosen to use natural language rather than a logical notation because for the common man natural language is very clear cut, but logic, full as it is of brackets and squiggly little symbols, is extremely ambiguous, even incomprehensible at times (brackets are unnatural (at least that's what we think (and we are not alone, either))).

The SILLI model is comprised of four main components:

(i) a natural language front end that translates users' premises into internal notation;

(ii) a deductive component to actually carry out the syllogistic reasoning;

(iii) a semantic network to encode knowledge of the world;

and (iv) a natural language back end which produces the conclusion derived from the premises.

Lets look at these components in more detail.

2. Components of the Model

2.1 The Natural Language Front End

This component must interpret natural language sentences of the form "A verb B" or "All A verb B". Using the revolutionary new picture theory of meaning ("a picture is worth a thousand words"), we convert an input sentence into an analogic representation called a picture.+ Obvious efficiencies

---

*This research was supported in part by charitable donations.

+A picture is a high level gestalt representation that completely circumvents the problems of the usual low-level pixels. See Hurtubise (1976) for more details.

in storage can be achieved by this method. Thus, the sentence

"Pick up the big red block and hit Mary with it."

would take up only .011 of a picture. Another advantage of using pictures is that there is absolutely no need to do pragmatics (this is a well known result from computer vision research). Because all this is done by efficient parallel procedures, we call our approach to language the procedural approach.

2.2 The Deductive Component

This component of the SILLI system takes a couple of interpreted premises and deduces a conclusion from them. To do this it uses two main logical postulates:

(i) The Active / Passive Postulate

If the first (or active) premise is of the form "All A something-or-other" and the second (or passive) premise is of the form "B is A", then the conclusion "B something-or-other" can be derived. Thus, this syllogism is valid:

active: "All Saints Cathedral has stained glass windows."
passive: "Saint James is a saint."
conclusion: "Saint James Cathedral has stained glass windows."

(ii) The EQ-NP Deletion Postulate

If there is a premise of the form "NP1 is NP2", the phrases NP1 and NP2 are equivalent. Either can be deleted and replaced by the other anywhere in any premise. For example, in the syllogism

active: "The temperature is 90°F."
passive: "The temperature is 32.222$\dot{2}$°C."
conclusion: "90°F is 32.222$\dot{2}$°C."

we have used the EQ-NP deletion postulate to delete "the temperature" from the passive premise and replace it with "90°F" from the active premise.

2.3 The Semantic Network

The semantic network component of the model is used to check user's statements of the form "A is B" for real world validity. The network consists of a bunch of nodes and arcs connected up into a generalization hierarchy. There are several kinds of arcs in this hierarchy: IS-A, A-K-O, IS, SUP, and ISA. The distinctions separating these arc types are subtle and are unjustly ignored in most semantic networks.

Figure 1 shows a sample network:

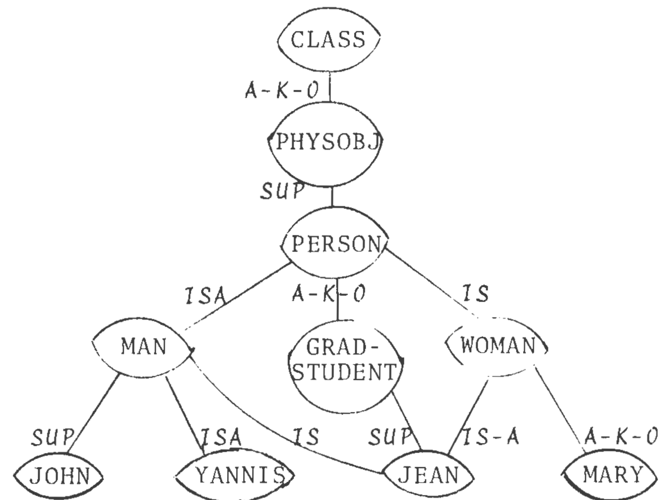


Figure 1 - A Sample Network

To see how the semantic network component is used by SILLI, lets make the simplistic assumption that the above sample net is the model's entire knowledge base. Now, if the user were to say something like

"John is a graduate student."

then this would not check out with the facts and the error message

"You have made an error."

would be printed out. Note that error messages, too, are in natural language, a nice piece of generality.

2.4 The Natural Language Back End

This works just like the natural language front end, only in reverse.

3. Implementation

Unlike many AI systems, our model has been implemented. Our programs are all encoded in a structured production system, consisting of structured productions of the form

IF ξ THEN ψ ELSE θ ;

where ξ, ψ, and θ are LISP procedures. Structured productions are better than normal productions because they have a natural syntax, they have two right-hand sides (ψ, θ) for each left-hand side (ξ), and they give the power of arbitrary LISP procedures to productions.

Our data structures are different from our programs because data and programs are quite distinct in real life (as opposed to the cloistered academic environment where sometimes data and programs can be the same). Data are stored in LISP lists because these are in widespread use throughout the AI community and also because they are flexible enough to store anything if you really try.

We have found that this combination of structured productions and LISP lists is very easy to use and, moreover, allows the construction of programs that are not only

efficient in time and storage, but are also readily understandable to the user. In fact our SILLI program has the following complexity bounds:

space: *lower bound*: $o(n^3 \log n)$

upper bound: $o(\log (\log (\log)))$

time: *lower bound*: $o(n \log (\log n^2))$

upper bound: $o((\log n) \div (\log n))$

We are still trying to precisely work out lower and upper bounds on understandability, although we suspect polysyllabic bounds in both cases.

A note of interest here is that these bounds seem to be related to the bounds discovered by numerical analysts for Runge-Kutta methods of order 6 that solve systems of stiff partial differential equations (PDEs). In fact for any bound "B" of our SILLI program, the equivalent bound for the Runge-Kutta method is "mB". What we would like to know is: with suitable analysis of round-off and truncation error (perhaps using backward error analysis), will we ever be able to show that B = mB?

4. Sample Run

At last we arrive at the moment of truth (or at least not falsity). Lets look at how SILLI handles a set of real syllogisms. We think the following actual run speaks for itself, although we will annotate the output (using comments surrounded by /* --- */) to indicate interesting points, much as is done in chess protocols (see any SIGART Newsletter for more details).

```
     ^C
     .LOGIN 1978,0721
      ENTER PASSWORD
      ██████████        /* The password is hidden from the reading public to protect */
      READY             /* our computer dollar budget.                               */
     .R LISP
     *(INVOKE STRUCTURED_PRODUCTION_SYSTEM_PROGRAMS_AND_DATA_STRUCTURES)
      NIL
     *(START_THE_SYSTEM_UP_NOW_PLEASE)   /* The SILLI system is started up.        */
      WELCOME TO THE SILLI WORLD         /* The syllogisms have been numbered for  */
      ENTER PREMISES                     /* the purposes of identification.        */
 1.  #MARY HAS BROWN HAIR.
     #MARY IS STANDING.                  /* These are the user's premises.         */
      THE CONCLUSION IS
     ¢STANDING HAS BROWN HAIR.           /* This illustrates the EQ-NP deletion    */
                                         /* postulate.                             */
      ENTER PREMISES
     #†§√¶Ξ∫(FAST_MODE_NOW_PLEASE)      /* Here we use the macro characters       */
                                         /* †§√¶Ξ∫ to temporarily call      LISP  */
      OKEY DOKEY                         /* from   within SILLI.  In this case     */
                                         /* we have SILLI stop printing "ENTER     */
 2.  #ALL MEN ARE CREATED EQUAL.         /* PREMISES" and "THE CONCLUSION IS"      */
     #JOHN IS A MAN.                     /* and instead just use # and ¢.          */
     ¢JOHN IS CREATED EQUAL. /* This illustrates the active / passive postulate.  */

 3.  #ALL STUDENTS LAID END TO END WOULD STRETCH AT LEAST
          FROM HERE TO NORTH BAY.
     #JOHN IS A STUDENT.                              /* Premises and conclusion    */
     ¢JOHN LAID END TO END WOULD STRETCH AT LEAST     /* can be more than one line. */
          FROM HERE TO NORTH BAY.

 4.  #JOHN WANTS TO BE THE PRESENT KING OF FRANCE.
     #THE PRESENT KING OF FRANCE IS BALD.             /* A classic problem, handled */
     ¢JOHN WANTS TO BE BALD.                          /* well by SILLI.             */

 5.  #MARY REALLY THINKS SHE IS IT.
     #IT IS RAINING OUTSIDE.                          /* SILLI handles this without */
     ¢MARY REALLY THINKS SHE IS RAINING OUTSIDE.      /* needing to resort to com-  */
                                                      /* plicated structures such as*/
 6.  #PICK IT UP.                                     /* belief spaces.             */
     #IT IS SUCH A PRETTY WORLD TODAY.
     ¢PICK SUCH A PRETTY WORLD TODAY UP.   /* Note the displacement of the particle */
                                           /* "up" in the conclusion.  This flaw    */
     ^C                                    /* is discussed in section 5.            */
     .REE
```

```
7.  #ALL MEN SHOULD LAY DOWN THEIR ARMS
       AND LIVE IN PEACE AND BROTHERHOOD.
    #WHAT IS A MAN?                          /* The "?" in the conclusion is preserved*/
    ¢WHAT SHOULD LAY DOWN THEIR ARMS         /* because of our sophisticated natural  */
       AND LIVE IN PEACE AND BROTHERHOOD?    /* language back end.                    */

8.  #NO MAN IS AN ISLAND.
    #HE IS A REAL NOWHERE MAN.               /* SILLI can handle negation. */
    ¢HE IS A REAL ANWHERE ISLAND.

9.  #ALL MEN ARE MORTAL.
    #SOCRATES IS A MAN.          /* SILLI picks up the user's mistake - since       */
    ¢YOU HAVE MADE AN ERROR.     /* Socrates is dead, how can he still be considered*/
                                 /* to be a man?                                    */
    ^C
    .KJOB      /* This ends the sample run.  We do not show our final statistics    */
               /* on the run due to acute embarrassment.                            */
```

5. Evaluation

Anybody looking at the performance and competence of SILLI on these examples has to be impressed. The deductive component handles a wide variety of syllogisms, usually with great aplomb and amazing grace and without combinatorial explosion. The natural language front end not only does what the user wants but also what he means to want; and vice versa for the back end. The system is very robust, able to overcome erroneous input (e.g. syllogism 9) with ease. Finally, SILLI seems to be well debugged. In fact, the only bug, if you can call it that, which manifested itself in this run was caused when we inadvertently hit ^C after syllogism 6. This just indicates that we are still human and haven't become dehumanized and depersonalized, an ever present danger in AI research that (unlike ours) attempts to down grade mankind by simulating aspects of humanity that are inherently so warm, moist, and fuzzy that no computer could ever do them.

Of course there are still a few unsolved problems. For instance the syllogism

active: "All students laid end to end would stretch at least from here to North Bay."
passive: "Students are people."
conclusion: "All people laid end to end would stretch at least from here to North Bay."

does not work (and in fact will send SILLI into an infinite recursive descent) despite its obvious real world validity. There are also minor difficulties in the natural language processing, as illustrated by the particle displacement in syllogism 6. A final criticism that nit-picking nay-sayers (such as the referees) could use to denigrate our approach is that "a few of the syllogisms in the sample run are a bit suspect" (emphasis mostly mine). Even if this is true, it must be admitted that they do exist (otherwise we wouldn't have been able to use them) and they must therefore be explained. The test of any theory is that it account for all the facts! Subtle problems notwithstanding, we believe our theory does just that.

6. Conclusion

Interesting as it is from a theoretical standpoint, this research as it now stands has only a limited number of practical applications. However, we have plans to vastly expand the usefulness of SILLI. We believe that methods similar to those used for syllogisms will also work well for the closely related areas of limericks and sonnets, and may be applicable to less structured domains such as blank verse, free verse, or even computer vision. The long term future of this research is assured.

Acknowledgements

The authors would like to gratefully acknowledge the Greeks. Without their invention of both the syllogism and the Greek alphabet, this research would have taken somewhat longer to complete.

Bibliography

Hurtubise (1976). S. J. Hurtubise, "A Model and Stack Implementation of a Conversation Between Some Man and a Smart Aleck Computer", Third CSCSI/SCEIO Newsletter, July 1976.

Hurtubise and Cumberbatch (1978). S. J. Hurtubise, R. "G." Cumberbatch, "In Defence of Syllogisms", see this volume.

Hopozopen (1975). Horace P. Hopozopen, "An Extensible Feature-Based Procedural Question Answering System to Handle the Written Section of the British Columbia Driver's Examination", Second CSCSI/SCEIO Newsletter, 1975.