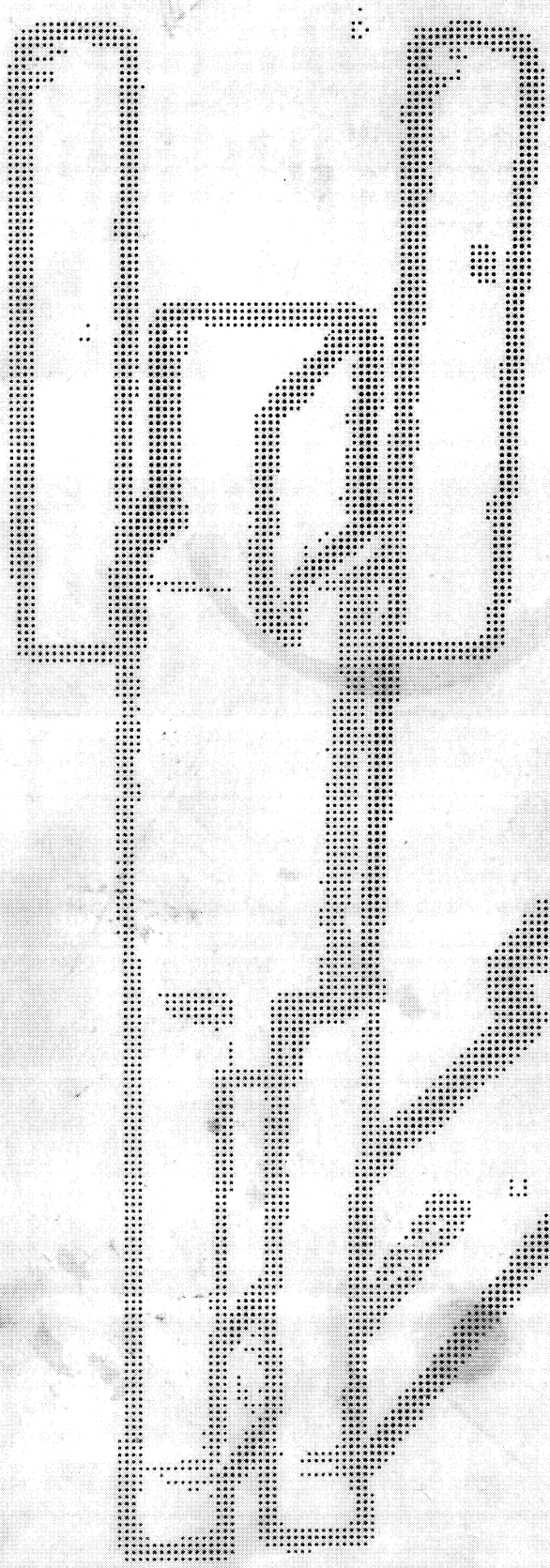
aisb



SUMMER
CONFERENCE
on
ARTIFICIAL
INTELLIGENCE
and
SIMULATION
of
BEHAVIOUR

CONFERENCE
PROCEEDINGS

UNIVERSITY
of
EDINBURGH

July 1976

PREFACE

The papers reprinted in this volume were accepted for presentation at the Second Summer Conference organised by the AISB (The Society for the study of Artificial Intelligence and Simulation of Behaviour). Roughly speaking, AISB is the European counterpart of the American SIGART study group of the ACM. More details of AISB activities can be found in its newsletter, which is circulated to members four times annually.

The conference is scheduled to be held in Edinburgh from July 12th to 14th, 1976, and follows the highly successful first summer conference held in Sussex in July 1974. A provisional timetable for the conference follows this Preface.

The Sussex conference consisted of twenty-one presented papers, and three invited tutorial lectures. The rapid growth of Artificial Intelligence in Europe, and throughout the world, is reflected in the fact that thirty five papers have been accepted for presentation at this year's conference.

In all, sixty six papers were submitted for presentation and each was refereed by two independent active workers in related fields in Artificial Intelligence. I would like to take this opportunity of thanking all the referees for their conscientious efforts, particularly those who accepted papers from me after frantic telephone calls with deadlines of yesterday. I have tried to ensure that the presented papers are of the standard normally required for publication in the field. I believe that, by and large, this has been achieved, despite the limitation that papers should not exceed ten typed pages. Certainly, the papers in this volume represent an accurate indication of the state of A.I. in Europe in 1976.

One of the more pleasant aspects of writing a Preface is the opportunity it affords for publicly thanking people who have made one's work easier. It is the simple truth that this volume would not have been produced, and the conference would not have been organised, without the heroic efforts of Janet Lee and Lesley Daniels.

Mike Brady,
Program Chairman,
Essex University, April 1976

PROGRAMME FOR AISB CONFERENCE JULY 76

MONDAY 12th JULY

Tutorial - *Natural Language: Maps not Chaps*		Y Wilks
Natural Language		G Ritchie J Levin & J Moore M Eisenstadt P Hayes & M Rosner
Project Presentation	:	Rutgers medical resource
Representation of Knowledge		R Young Y Wilks L Steels W Harwood & F Hanna

TUESDAY 13th JULY

Tutorial - *Person Perception*		S Weir
Vision/perception		R Emanuel & S Weir G Hinton F O'Gorman M Adler
Project Presentation	:	Essex FORTRAN coding sheets
Vision/Perception		J Paul I Witton C Lamontagne E Soloway & E Riseman

WEDNESDAY 14th JULY

Tutorial - *Problem Solving*		R Waldinger
Planning/Reasoning		A Bundy et al, D Warren A Stanier O Stepankova & I Havel M Ornato & G Zarri
Project Presentation	:	F Brown's work on arithmetic
Tools for AI		B Anderson R Bornat & B Weilinga G Attardi, C Montangero & G Prini A Sloman & S Hardy

CONTENTS

	Page
Recognition of Peanuts Cartoons M R Adler University of Edinburgh	1
A Brief Critique of LISP B Anderson University of Essex	14
A High Level Machine for Artificial Intelligence G Attardi, C Montangero & G Prini University of Pisa	26
Reasoning about Hand Printed FORTRAN Programs R Bornat University of Essex	38
Finding Blobs of Writing in the FORTRAN Coding-sheets Project R Bornat & J M Brady University of Essex	47
Does AI Programming really have to be like knitting with Spaghetti? R Bornat & B J Wielinga University of Essex	56
Seeing a Pattern as a Character J M Brady & B J Wielinga University of Essex	63
The Role of Extensible deductive systems in Mathematical Reasoning F M Brown University of Edinburgh	74
A Deductive System for Elementary Arithmetic F M Brown University of Edinburgh	84
MECHO: Year One A Bundy, G Luger, M Stone & R Welham University of Edinburgh	94
Processing Newspaper Stories: Some thoughts on Fighting and Stylistics M Eisenstadt Open University	104

Page

Catalysing Communication in an Autistic Child in a LOGO-like learning environment 118
R Emanuel & S Weir
University of Edinburgh

A Distributed Activity Processing System for AI 130
W T Harward & F K Hanna
University of Kent

ULLY: A Program for Handling Conversations 137
P J Hayes & M A Rosner
University of Essex

Using Relaxation to Find a Puppet 148
G Hinton
University of Sussex

Problem Reduction and the 2-dimensional Trim-loss Problem 158
A I Hinxman
University of Liverpool

Clinical Consultation and the Representation of Disease Processes: some AI approaches 166
C A Kulikowski, S Weiss, M Trigoboff & A Safir
Rutgers University

Visual Motion Detection: A Computational Theory and some of the Psychological Data which it integrates 175
C Lamontagne
University du Quebec a Montreal

Dialogue Games: A Process Model of Natural Language Interaction 184
J A Levin & J A Moore
University of Southern California

Edge Detection using Walsh Functions 195
F O'Gorman
University of Sussex

An Application of Artificial Intelligence in Information Retrieval. *RESEDA* project for Medieval Biographies 207
M Ornato & G P Zarri
National Centre for Scientific Research, Paris

Page

Seeing Puppets Quickly 221
J L Paul
University of Sussex

Problems in Local Semantic Processing 234
G Ritchie
University of Edinburgh

Giving a Computer Gestalt Experiences 242
A Sloman & S Hardy
University of Sussex

Planning to make Tricks at Bridge 256
A M Stanier
University of Essex

Incidental and State-Dependent Phenomena in Robot Problem Solving 266
O Stepankova & I M Havel
Institute for Computer Techniques of the CVUT
Czechoslavakia

A Formalism for Case Systems 279
L Steels
Antwerp UTA

Recognising Plans and Summarizing Actions 291
C F Schmidt, N S Sridharan & J L Goodson
Rutgers University

Mechanizing the Common-Sense Inference of Rules which direct Behavior 307
E M Soloway & E M Riseman
University of Massachusetts, Amherst

The Frame and Focus Problems in AI: Discussion in Relation to the *BELIEVER* System 322
N S Sridharan
Rutgers University

Propagation of Information in a Semantic Net 334
M Trigoboff & F Schmidt
Rutgers University

Page

Generating Conditional Plans and Programs 344
D H D Warren
University of Edinburgh

De minimus or the Archaeology of Frames 355
Y Wilks
University of Edinburgh

Generating Natural Speech from Text 366
I H Witten
University of Essex

Design Choices for a World-Modelling System 376
R M Young
University of Edinburgh

M.R. Adler

RECOGNITION OF PEANUTS CARTOONS

The aim of our research is to interpret cartoons which are two-dimensional representations of events which happen in the three-dimensional world. We discuss the difficulties peculiar to this domain of irregular curved line drawings. The strategy we adopt is to reduce the importance of precise shape description by exploiting context information provided by the use of models. The method is described and the action of the program is outlined for its analysis of a particular scene.

Key words: Computer Vision, Curved Line Drawings,
Occlusion Analysis, Scene Description.

I. INTRODUCTION

Previous research work on machine recognition has been concentrated on the analysis of regular planer objects (Winston, 1975) and more recently very regular curved surfaces. (Turner, 1974) Our research enters a quite different universe: curved line drawings in the form of PEANUTS cartoons. This domain has its own set of problems very different from earlier machine recognition work. The main problems stem from the difficulty in describing very curved lines:

1. Shape Descriptions Curved objects are more difficult to describe than plane polyhedra. The difficulty is compounded when the curved objects are also irregular, and vary each time they are hand drawn.

2. Occlusion An object (or its two-dimensional representation) is said to be occluded if it is partially hidden from view by another object or objects. For regular objects this has a minimal effect since the path of regular lines can often be predicted when they disappear behind another object. Contour extention cannot be done for the irregular curves in cartoons.

In particular, the PEANUTS characters are many-jointed, and so

the most usual occlusions which occur are self-occlusions such as a hand in front of a face. In such situations the shape characteristics of that part of the object that remains visible are often dramatically altered, so the recognition process can become very difficult.

The strategy we adopt is to reduce the importance of precise shape description by exploiting context information provided by the use of models. Guzman (1971) outlines a procedure for the analysis of curved line drawings; many of our ideas are founded on his original concepts. He pointed out the context effect: viz. components of a scene are locally ambiguous because each shape can have several possible values. For example, a circle can be the sun, a ball, an eye, etc. He proposed a system of models to provide context information to determine a sensible value for each object. Models are proposed on rather gross features and once invoked determine the interpretation.

II. SCOPE OF THE SYSTEM

We shall not attempt to analyse all possible PEANUTS scenes, but a restricted subset of them. The program is designed to handle typical snapshots of scenes in which one or two PEANUTS characters are engaged in some activity. Included in these scenes will be objects such as baseball equipment, a table top, a brick wall, etc.

The number of characters and objects that the system is capable of recognising is limited by the models available. In principle, by adding more models, the system's scope may be extended. An example of the type of scenes we deal with and the description that will be produced can be seen in figures 1 and 2.

III. MODELS

The region is the basic primitive used in our analysis. It is particularly well-suited for the PEANUTS world since the scenes are composed of closed areas into which they are easily divided. Unlike the BLOCKS world, lines by themselves convey very little information since they are not always edges; regions, however, are always representations of surfaces.

We use a hierarchy of models to guide us. These fall into four categories: STRUCTURE models, COMPONENT models, DESCRIPTION models and COMPOSITION models. They have two main purposes:

1. To provide shape information. That is, a description of the regions which occur in the scene at a local level, i.e. the appearance of the region.

2. To provide information about relationships between regions. This is a more global view. It explains how small regions join together to form larger, more meaningful structures.

III.1 How Models are Used

The analysis is guided step by step by the relation models of objects in the PEANUTS world. By applying information about how regions may be connected to represent people and objects, the analysis system can focus on one region after another. In this way a full description can be constructed, matching each region in the scene to each part of the corresponding model. In principle, once the correct model is invoked, it should be relatively simple to identify the regions. In practice, it is more difficult because of complications, such as occlusion and variations in orientation.

Just as the relation models guide us globally in terms of where to find the next region, the shape models guide us in the local analysis of matching regions to shape descriptions. Each shape description has an associated series of tests which are applied to the data defining the region. The number and type of tests varies with the complexity of the shape. A SOCK is easily tested since it is usually rectangular, but a more complex shape such as LUCY's HAIR requires more tests. The purpose of these tests is not to decide if the region in question matches the model; rather it is to decide if there is sufficient evidence to give such a possibility the benefit of the doubt. The expectation of what the region should be, provided by the relational model, allows us to accept a result on rather flimsy evidence.

The obvious advantage of this technique is that there are fewer checks or tests required before assigning (at least temporarily) a label to a region. This is a short cut. The human visual system might take a similar short cut, that is, recognising an object on the basis of a fleeting impression rather than performing a detailed analysis. Only if an inconsistency is discovered is the detailed analysis necessary to correct a misperception. The disadvantage is that choosing the wrong model often allows the analysis to proceed much further than is desirable before an inconsistency is noticed. A table of model similarities is used at such points to help locate the correct model.

III.2 What the Models Are

Flexible joints, self-occlusion and the variety of possible positions suggest that a complex model is needed. The discussion which follows will center on the PERSON model, since it is both the most frequently used as well as the most complex. Within the PERSON model, there is no distinction between what a person is wearing and his actual body. The analysis is region-dependent; it assumes that bodies are divided into regions, mainly by different articles of clothing. A model for PERSON must deal with:

1. A variety of characters. These are distinguished by various details that characterise each individual.

2. A variety of orientations. Different 'frames' (Minsky, 1975) are used for different orientations. The number and scope of these different frames is limited by the simplicity of the PEANUTS environment.

3. A variety of body positions. The flexibility that allows self-occlusion and can alter the arrangement and shape of the various body components is also handled by Minsky's frame idea.

Briefly, the hierarchy of models for PERSON are:

1. STRUCTURE model-- This is basically a coarse relational model. (see figure 3)

2. COMPONENT model -- There is only one structure model for PERSON, but several COMPONENT models for each subpart of the STRUCTURE model. (see figure 4) These too are relational models, but their scope is more local.

3. DESCRIPTION model -- Each terminal in a COMPONENT model has a DESCRIPTION model that describes its shape. Usually this is a series of tests to be performed on a single region, however some elements are formed of separate regions (e.g. a striped shirt) and these must first be gathered together by the

4. COMPOSITION model -- This model assembles regions into one recognisable region that will match the shape requirements of a DESCRIPTION model.

IV. OCCLUSION

It is important to be able to discover which objects are occluded since this alters their perceived shape. In simple geometric worlds straight lines or even elementary curves can usually be extended to fill the gaps in a contour produced by occluding objects. Cartoon shapes are irregular and the direction the hidden lines take cannot be so predicted. Furthermore, all lines are not true edges since there are curved surfaces. The T-joint is a powerful tool in the study of occluded objects. It is formed by two lines which meet with one line continuing through the vertex, while the other is terminated (figure 5). They occur in two situations:

1. Occlusion -- In this situation, a T-joint implies that the boundary of one object has gone under another object, or else that a boundary that was "under" has reappeared. Thus, T-joints typically occur in pairs. The simple idea that finding the correct pair of T-joints provides information about region occlusion is used as the basis for a powerful pairing heuristic.

2. Abuttal -- contact with no overlapping.

Because of this second possibility, it would be too expensive to apply the pairing heuristic to all the T-joints in the scene. So it is only used when a region cannot match a SHAPE description and occlusion is the suspected cause of this failure. It is not always easy to decide which T-joints to pair on the basis of local clues, furthermore, the orientation of the T-joints (i.e. which line is the stem) is not always obvious in a curved line world.

We have devised heuristics for the proper pairing of T-joints to any depth of occlusion. These heuristics involve following both the stems and bars of the starting T-joint to the finishing T-joint according to certain rules. (Adler, 1975)

V. SYSTEM OVERVIEW

The system in block diagram form can be seen in figure 6. It has been written in the programming language CONNIVER (McDermott & Sussman, 1974), as implemented in MACLISP (Moon, 1974).

V.1 Input Data Processing

The process begins with a hand drawn frame of a cartoon (figure 1). This drawing passes through several stages of processing before it is presented to SCAN:

1. Digitisation -- The figure is traced by hand and the ordered X-Y coordinates for each line are recorded by the digitiser.

2. Computer Interpretation of Coordinates -- The data is catalogued: lines are given names, junctions are found and labelled, lines are ordered around each junction.

3. Region Formation -- Regions are formed by tracing lines to junctions and forming closed areas (Roberts, 1965). Lines that do not form regions are labelled as surface-markings. The region that is formed by the outline of the whole group of regions is labelled as the outer closure.

The outer closure is used to isolate each figure in a scene with more than one character or object. This does not always work. Luckily, in PEANUTS cartoons, HEADs are usually easily isolated and this gives us an alternative method of selecting a starting point.

V.2 Identification

SCAN takes control of the analysis. Its job is to postulate the identity of the regions in terms of the available models. For each region-to-model match an item is added to the data base of PARTIAL RESULTS. These items can trigger an identification of another region not being immediately considered since the data base is constantly monitored.

V.3 Model Implementation

The STRUCTURE model of PERSON, being invariant, is handled simply, using a MACLISP function, the other models are all implemented as CONNIVER IF-NEEDED methods. Each model uses this technique to call the next model type in the hierarchy. The pattern reduces the many possible models to a small group of eligible ones.

But all this is only part of the problem. Occlusion is a major source of difficulties. Again decisions must be made, but these are less straight-forward and more control is required after each decision node.

V.4 The TROUBLE-SHOOTER

The communication between models is straight-forward and is easily handled by CONNIVER's standard control primitives. The TROUBLE-SHOOTER must have more intricate control capabilities than the models because it must examine past decisions to find where a mistake has been made, and re-start SCAN at that point. The TROUBLE-SHOOTER mechanism is based on Fahlman's (1973) approach to a similar problem.

The TROUBLE-SHOOTER is not limited to retracing through bad decisions. It also has three other important functions:

1. Dealing with missing model parts -- When an entire substructure of a model is completely occluded or occluded to the extent of being unrecognisable, the TROUBLE-SHOOTER checks the partial solution to choose the next move.

2. Dealing with 3-D occlusion -- Some very curvy objects exhibit special characteristics since there is no junction evident where the line changes planes (Turner, 1974). This case cannot be handled by the simple T-heuristics and is thrown to the TROUBLE-SHOOTER.

3. Model suggesting -- Regions which are not part of the current model, or are in an unexpected position must still be recognised. The TROUBLE-SHOOTER suggests the appropriate model.

VI. ANALYSIS OF A SCENE

This section outlines the analysis of one particular scene (figure 1). The system has analysed this scene although several procedures must be made more general before further scenes are handled. Work is proceeding and the system should complete its second scene shortly.

1. As a first step all 21 regions, 7 surface-markings, 61 lines and 45 junctions are catalogued and entered into the data base. This figure is already isolated and the height/width ratio as well as area and orientation conform to the PERSON model.

2. A very good way to recognise a person is to study his/her face and this is the approach taken by the system. The STRUCTURE model suggests HEAD as a starting point and indicates that it is probably the uppermost region (relative to the orientation).

3. SCAN passes control to the HEAD COMPONENT model which tries to interpret the uppermost region (R0203) as an instance of someones's HEAD. There are several possible cases for PEANUTS HEADs depending on orientation and who the PERSON is. All HEAD COMPONENT models are placed on the POSSIBILITY-LIST by FETCHing:

(COMPONENT HEAD !?who R0203 !?view !>score-head)

4. We now step through this ordered POSSIBILITY-LIST trying each COMPONENT model in turn. The usual case for a HEAD is HAIR-above-FACE. This COMPONENT model initiates a search for its first component, viz. HAIR with the Fetch argument:

(DESCRIPTION HAIR !?who R0203 !?view !>score-hair)

All HAIR models are collected here, under the constraints:

who excludes CHARLIE BROWN (he has no hair)
view excludes BACK (no face region)

5. We apply each HAIR DESCRIPTION's test to region R0203 until the model for LUCY facing towards the FRONT-RIGHT matches, and adds:

(DESCRIPTION HAIR LUCY R0203 FRONT-RIGHT 85)

to the data base. The score is determined on the basis of test results and is in the range 0-100. A demon method is also added to monitor the data base and locate the remaining HAIR regions.

6. Control returns to the HEAD COMPONENT model. The model knows that FACE is below HAIR. Finding the correct FACE model is easier because the pattern has now been filled in: (DESCRIPTION FACE LUCY R0189 FRONT-RIGHT!>score-face)

7. The DESCRIPTION model for FACE fails since the lower part of the FACE is hidden by her hands resulting in a low score.

8. The TROUBLE-SHOOTER looks for an occlusion to account for this failure. It selects T0225 and T0235 as pair candidates, but the heuristic fails at T0227. Only T0228 can be paired with T0227, so its orientation is chosen to allow the pairing.

9. The sub-problem of pairing T0228 and T0227 is straight-forward. The item: (OCCLUDED-BY R0200 R0205) is added to the data base and a super-region S0300 is formed from the two regions.

10. Now the pairing of T0225 and T0235 is restarted. The bar trace is successful but now the stem trace is thwarted. The trace from T0235 stalls at T0232 and the trace from T0225 stalls on T0229.

11. The TROUBLE-SHOOTER cannot find a pair for T0229. This junction is re-oriented to allow the trace to continue until it stalls on T0231. Now T0231 and T0232 can be paired; The trace has shown that R0200 and R0205 together occlude the proposed FACE R0189.

12. The FACE returns successfully, adding THE ITEM:
(DESCRIPTION FACE LUCY R0189 FRONT-RIGHT 55)
This triggers the demons looking for more HAIR. The regions R0204, R0202, and R0201 are located, adding to the HAIR score.

13. The HEAD COMPONENT has found all its subparts. It gathers them into one super-region S0301, and exits adding:
(COMPONENT HEAD LUCY S0301 FRONT-RIGHT 78)

14. Next SCAN is guided by the STRUCTURE model to find TORSO below the HEAD. This fails because R0205 and R0200 are in the way.

15. The TROUBLE-SHOOTER must now suggest a model for R0200 and R0205 knowing their size and the fact that they occlude the FACE. An ordered POSSIBILITY-LIST is formed and the ARMs are recognised.

16. Now with ARMs out of the way, TORSO can be applied. We now know that the character is LUCY so we search for BLOUSE and SKIRT instead of SHIRT and SHORTS. The occlusion of the top of BLOUSE is handled by the TROUBLE-SHOOTER. The skirt is easily recognised, and the BOW is handled by demons.

17. Since ARMs have already been found, the STRUCTURE model suggests the final part: LEGS. The SKIN and SOCK parts are easily recognised. The SHOES cause two problems. First, each SHOE is composed of three regions. This is solved by the SHOE COMPOSITION model that builds the three regions into one recognisable one. Second, the SHOE on the right is also occluded. This is handled again by the TROUBLE-SHOOTER.

18. Finally, all the COMPONENTs have been found and all the regions have been identified. The analysis is complete.

VII. CONCLUSION

In our cartoon world of animate objects, we have to contend with shapes which are irregular, flexible, and partially occluded. Our analysis exploits the fact that this scene was drawn by a cartoonist who has certain conventions for recording significant events in an action sequence. In a sense, the cartoonist has pre-processed the scene, transforming it into a simpler line drawing. Our program begins by finding the outer closure which usually corresponds to a single person. This is no accident -- the cartoonist has arranged it for us. Simple tests for global shape information can be applied at this early stage to produce evidence for model selection.

The model hierarchy allows us to structure the knowledge of the system into groups of expert methods, each with a specific task to perform at a given level of the analysis. CONNIVER offers us a back-tracking facility with examination and ordering at decision nodes with the standard POSSIBILITY-LIST feature, as well as providing a more sophisticated control structure for re-examining past decisions. One drawback to our system is the number of different model METHODs that may be required for each new object to be recognised. For rigid objects, the number of models is small, but by applying different models for different orientations of each sub-part of a flexible object one builds up the size of the data base quickly.

ACKNOWLEDGEMENTS

I should like to thank Dr. Jim Howe, Dr. Sylvia Weir and my colleagues at the Department of Artificial Intelligence for their constructive critisism. This work was supported in part by the Social Science Research Council.

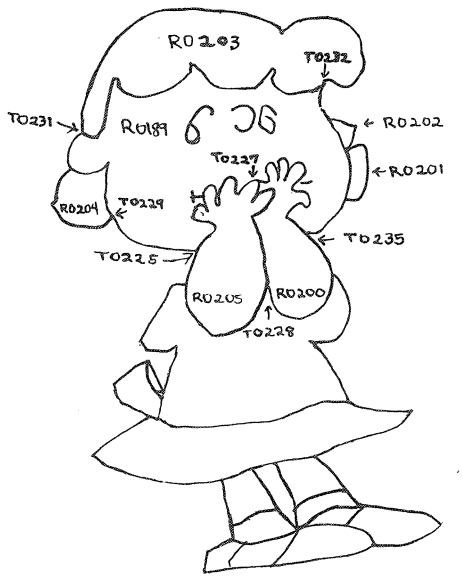


Figure 1



Figure 2

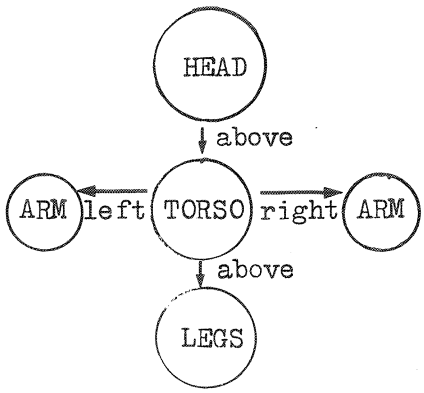


Figure 3. Person STRUCTURE model.

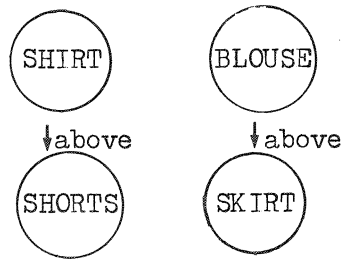


Figure 4. COMPONENT models for TORSO.

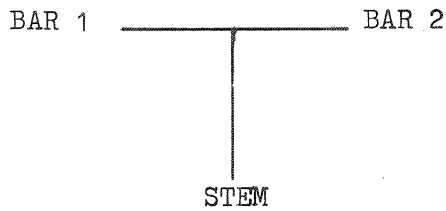


Figure 5. T-joint.

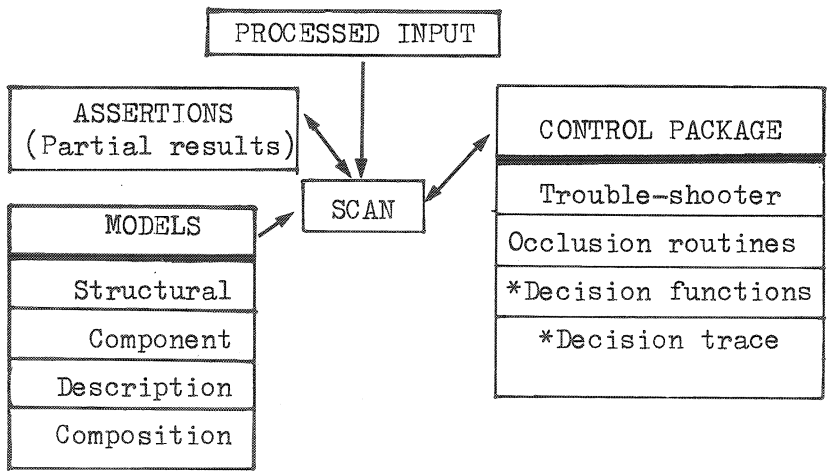


Figure 6

REFERENCES

Adler, M. R. (1973) An Analysis of Occluded Curved Line Drawings. B.S. Dissertation, Artificial Intelligence Laboratory, M.I.T.

Adler, M. R. (1975) Understanding PEANUTS Cartoons. Progress in Perception, D.A.I. Research Report No. 13, 39-73. Dept. of Artificial Intelligence, University of Edinburgh.

Fahlman, S. E. (1973) A Planning System for Robot Construction Tasks. A.I. Technical Report No. 2334, Artificial Intelligence Laboratory, M.I.T.

Guzman, A. (1971) Analysis of Curved Line Drawings Using Context and Global Information, in Machine Intelligence 6 (eds. B. Meltzer & D. Michie), 325-375. Edinburgh: Edinburgh University Press.

McDermott, D. & Sussman, G. (1974) The CONNIVER Reference Manual. A.I. Memo 259A, Artificial Intelligence Laboratory, M.I.T.

Minsky, M.A. (1975) A Framework for Representing Knowledge, in Psychology of Computer Vision (ed. P.H. Winston), 211-277. New York: McGraw Hill.

Moon, D.A. (1974) MACLISP Reference Manual, Project MAC--M.I.T., Cambridge, Mass.

Roberts, L.G. (1965) Machine Perception of Three-Dimensional Solids, in Optical and Electro-optical Information Processing (eds. Tiffet et al), M.I.T. Press.

Sussman, G. & McDermott, D. (1972) Why Conniving is Better than Planning. A.I. Memo 255A, Artificial Intelligence Laboratory, M.I.T.

Turner, K.J. (1974) Computer Perception of Curved Objects Using a Television Camera. Ph.D. Dissertation, School of Artificial Intelligence, University of Edinburgh.

Winston, P. H. (1975) The Psychology of Computer Vision. New York: McGraw Hill.

Bruce Anderson

A BRIEF CRITIQUE OF LISP

LISP is currently the default language for AI programming (and is often used for other non-numerical work) and it is also used in Computer Science teaching to illustrate various things, especially functional programming and heap storage allocation. Pedagogical use seems to be on the increase, with three new textbooks published or in press (Friedman 1974, Siklossy 1975, Allen 1976). While LISP systems may well be the best currently available for such tasks (I'm not sure), I think the language has serious deficiencies which this paper attempts to discuss. I do assume however that an incremental, interactive language is best for the experimental type of programming done in AI. Thus I think LISP is the right KIND of language, and will make criticisms from this point of view, rather than from the anti-interactive "Pascal is best" position. However one of the most exciting fields for research in programming at the moment is in the area between these views, the development of programming systems which are interactive yet can use knowledge about programs (e.g. type information) to do static checking and produce compact code.

I do not mention control-structures beyond recursion or their influence on implementations, though this is of course a vital current topic in AI.

A NON-PROBLEM

I feel that I have to mention this, though the topic is essentially orthogonal to the later discussion.

1. LISP's syntax. This is probably the aspect of LISP most disliked by non-users, and is the butt of many anti-LISP jokes, but doesn't seem to worry LISP users, and indeed in an interactive system for program preparation with the [] feature, a single-character quote, and a prettyprint, the problem goes away. These features are crucial: yet oddly enough they are made possible by the very fact which makes them essential, the simplicity of the syntax. There is no widely-used formatter for BCPL or POP-2. Of course one can provide a prettier syntax for LISP, as MLISP does, but the effort of reprogramming the debugging and editing systems to reflect this has not been made, and MLISP is not popular.

The structure-oriented editor is a very powerful tool, essentially forcing all edits to produce syntactically correct programs. Again, this is a relatively weak constraint in the sense that even "surface semantics" such as arity of system-provided functions is not checked. However it simplifies and encourages program transformations such as function folding:

```
(FOO X)
to
((LAMBDA (A) ... A ...) X)
to
((LAMBDA (X) ... X ...) X)
to
... X ...
```

which can be done concisely and accurately. It would be useful to try to implement such a system for a language with more syntax, such as BCPL, and see where the tradeoff is best, ease of editor implementation versus syntactic clarity.

It is interesting that the command language for the LISP editor is not LISP, nor is the command language for the Programmer's Assistant and other subsystems of InterLISP. This points out the defects of LISP as a command language, the fact that the shortest command involves two parentheses, and that in a command-oriented kinaesthetic interaction like editing, brevity is essential. Having more than one language in a system leads to problems however. For example the editor has its own "subroutine" method - edit macros, and while the editor can evaluate LISP, and LISP can call the editor, debugging and tracing does not extend to both, error handling is different, and edit macros cannot be compiled. The demands on command languages and ordinary programming languages are different: in the command case one is in a very well-defined context and sees immediately the results of an action, so that methods which would be dangerous in a general context can be used, such as short function (command) names, functions with several side-effects, and defaults for missing parameters. Thus I think it is wrong that InterLISP ALWAYS defaults missing parameters to NIL: in the command case this can be seen and remedied if it is wrong, but in the general case can lead to opague code and nasty bugs. Of course if people object to TYPING then the program-input system can fill in the spare NILs immediately the right parenthesis closing off a too-short argument-list is typed. The interplay between command-language and programming-language styles of interaction in a reactive system is an interesting area for research. Using syntax macros to create e.g. an editing

sublanguage (as in done in POP-2) could be the way to proceed, provided they are powerful enough to implement a complex enough syntax, and provided the debugging system interfaces with them. The latter is unlikely.

DIFFICULTIES IN THE LANGUAGE ITSELF

Of course LISP is rather MUDDLED in certain areas, and it's hard to describe a muddle as a set of separate errors since the very isolation implied by this description implies things can be put right one at a time when this may not be so. In this section I try to ignore the historical influence of implementations on semantics.

2. Functions are treated badly. Basically identifiers stand either for variables or for functions, so there is no real notion of function as value. Thus in (X X) the first X is evaluated by looking at X's function cell, but the second one by looking at its value cell. When the function is called the new environment contains new VALUE cells for the locals and parameters, but not new function cells. Thus we pass (QUOTE CAR) as an argument, not CAR, and the interpreter must go from the argument name to the function cell via this atom. This is inefficient, and can cause confusion e.g. if the value of FOO is (QUOTE CAR) then (FOO X) causes (CAR X) which may well be unintended! Notice also that the predicate ISFUNCTION must sometimes be a predicate on atoms, i.e. (ISFUNCTION CAR) requires ISFUNCTION to be an FEXPR, but at other times must evaluate its argument first e.g. in

```
(LAMBDA (X Y) (COND [(ISFUNCTION X) ...]...))
```

These two situations cannot be distinguished in general - there is complete confusion between functions and their representation by atoms. But why is it bad to treat functions a bit differently? It doesn't complicate the interpreter much and perhaps it's more efficient. The answer is that functions are essentially the ONLY tool for abstraction that LISP provides. If we create a complex data-structure to represent something then we must create functions to manipulate it if we are not to become bogged down in a sea of CDDADDDRs. It is then quite natural to want to pass these functions around as arguments to other functions which manipulate structures. Even in the case of MAPCAR we introduce inefficiency by passing atoms (i.e. representations) rather than functions.

3. Functions are puny anyway. The function is LISP's main tool for constructing new objects, but it is hard to return

interesting functions as results except by CONSing up new code. Basically the difficulty lies in not being able to form a closure of a function with a piece of environment to produce a new function. Consider the problem of writing FUNPROD so that

```
((FUNPROD F G) X) = (F(G X))
```

even neglecting that (QUOTE F) etc would be needed on the left along with some lookup mechanism. The usual answer given is:

```
(LAMBDA (F1 F2)
     (QUOTE (LAMBDA (X) (F1(F2 X )))))
```

but this fails as soon as the result is used in an environment where F1 is not bound to whatever it was when FUNPROD was called! The solution is to use the FUNARG device to refer F1 and F2 to the correct environment, but in LISP 1.6 this facility doesn't work if the FUNARG is returned as a result, for it is implemented by a pointer into the activation stack. InterLISP does allow an implementation, but this shows only too well what poor relatives functions are in that language. Here is the code:

```
(DEFINEQ (FUNPROD (F G)
(FUNCTION (LAMBDA (X)
(APPLY* F (APPLY*G X)))
(F G))))
```

and it is used e.g.

```
(DEFINE (QUOTE AB)(FUNPROD(FUNCTION A)(FUNCTION B)))
then
(APPLY* AB X)
```

As mentioned above, functional arguments must be quoted when passed, and explicitly called when used: note also that FUNARGS even at the top level must be called with APPLY or APPLY*. The best LISP 1.6 program I could think of to do the job is:

```
(DE FUNPROD (F G)
(LIST @LAMBDA @(X)
(LIST (GET F @EXPR)
(GET G @EXPR)) @X))))

(PUTPROP @AB (FUNPROD @A @B) @EXPR)
(AB X)
```

but see how the use of CONS is very unstructured (even for LISP!) and also delays until run-time the production of the code, so that there is no way the code produced can be compiled. Also, the functions bound in are not accessible as is the case in a closure or FUNARG. Flexible function-building is a powerful tool, and allows implementation of, and experimentation with, other structuring methods. As a simple example, take record structures. The obvious way to implement them is to have a function which takes a description of a record class and returns the functions for creating, recognizing and accessing components of the corresponding records, as is done in POP-2. For example we might like

```
(RECORDFNS @PAIR 2)
```

to return the functions for two-place records called PAIRs and (finessing the multiple-assignment and functions-aren't-objects problems !) use it

```
(SETQ (MKPAIR FRONT BACK ISPAIR)(RECORDFNS @PAIR 2))
```

so that

```
(ISPAIR (MKPAIR 1 2))
```

is true and so on. This can't be done except by consing up new code as there is no way to specialize the general record-functions to create the particular ones needed. Compare this with the INTERLISP "implementation":

```
declaration:
(RECORD PAIR (FRONT.BACK))
creation:
(CREATE PAIR FRONT_3 BACK_4)
access:
X:FRONT = (CAR X)
update:
X:FRONT_55 = (RPLACA X 4)
```

but the expansion is done essentially by load-time macro-expansion. FRONT and BACK are global constants, not functions, so for example they cannot be passed as arguments to MAPCAR or locally rebound. "Where there's a will, there's a kludge"! Of course most LISP programmers don't feel this lack of power any more than FORTRAN users miss recursion, but it does seem wrong that they should be cut off from well-tried techniques of good programming, let alone from experimenting

with more recent ones such as truly functional data-structures.

I have mentioned the magic word "closure". Quite what a closure is nobody really knows, other than a really rather general binding of an environment to a function body. It seems to depend on what you think the environment is i.e. what you think is important, tempered perhaps by your knowledge of what things cost! Thus POP-2 provides a clean call-by-value bindings-only closure via its partial application, a feature still little known outside POP-2 circles even after 8 years of use. Slipping into POP-2 syntax, we can solve the FUNPROD example by

```
FUNCTION FUNPROD FF GG;
LAMBDA X FFF GGG;
FFF(GGG(X))
END(%FF,GG%)
END;
```

where the (% %) brackets indicate partial application i.e. that the LAMBDA should become a function of ONE argument with FFF and GGG always bound to the values of FF and GG at the time of the partial application i.e. to FUNPROD's arguments. The record-structure example is too long to do here but would essentially involve partially applying functions such as

```
LAMBDA OBJECT COMPONENTNUM TEST;
IF TEST(OBJECT) THEN GET(OBJECT,COMPONENTNUM)
ELSE ERROR(...)
CLOSE
END;
```

Despite the elegance of this method, it can be somewhat difficult to use if frozen parameters are to be assigned to in such a way that the new value persists between activations, or if they are to be accessed from other functions. For example a function which just returns the number of times it has been called can be done either:

```
FUNCTION COUNT REFNUM;
CONT(REFNUM)+1 -> CONT(REFNUM);
CONT(REFNUM)
END(%CONSREF(0)%)
```

or

```
FUNCTION COUNT NUM SELF;
NUM+1 -> FROZVAL(1,SELF);
NUM+1
```

```
END(%0,UNDEF%);
COUNT -> FROZVAL(2,COUNT);
```

but both are a bit messy. Inability to assign to a parameter (in such a way as to change the value of the actual parameter) is of course correct for call by value. Just what access should be provided to the environment mechanisms and which closures should be provided explicitly is a current topic of discussion outside the topic of this paper as it involves consideration of control structures.

4. Poor assignment facilites. There are many different assignment functions [SET, SETQ, RPLACA, DE, DF....] so that it isn't easy to see where side -effects are taking place. Nor is there any notion of "place" so that assignment can be thought of as putting something in a place. One would prefer

```
(SETQ (CAR X ) 6)
to
(RPLACA X 6)
```

and would really like to be able to have read/update pairs so that

```
(FOO X)
```

could be used on th "left" of an assignment as well as the right:

```
(SETQ (FOO X) (FOO STEVE))
```

of course this is one of the classic problems in language-design. If (FOO X) can , at different times, denote a location and the value in that location then we get into the question of when it means which, L-value or R-value, to coerce or not to coerce. I don't find any of the "solutions" attractive, and the fundamental tension between "having a value" and "being a box", applicative versus imperative, is still unresolved. In fact it is now more complicated as we see the power of the idea that "being a box" is just a trivial case of "being a structure that can respond to an assignment message" so that the above assignment is "tell FOO to put (FOO Y) in her Xth component", with the possibility that FOO will complain because it is write-locked, or print out a message because it is being traced. Thus POP-2's approach, where (in LISP syntax)

```
(SETQ (FOO X) <expression>)
```

is syntactic sugar for

((UPDATER FOO) X <expression>)

is an improvement on the box approach, but not totally satisfactory.

5. FEXPRS are dangerous! They are confusing, much more so that call-by-name or call-by-reference which they are often used to implement. Arguments are easily evaluated in the wrong environment, or not dealt with in full generality. For example (FEX A) may work but not (FEX B) or (FEX (CAR A)), as in this SETZERO example:

```
(DF SETZERO (B)
(SET (CAR B) 0))
```

For (SETZERO A)/(SETZERO B)/(SETZERO (CAR A)) the internal SET is equivalent to (SETQ A 0)/(SETQ B 0)/(SETQ (CAR A) 0). The second of these is the wrong (i.e. inner) B, and the third isn't LISP. Part of the usage is caused by the lack of a good notation for constants, a problem partly cured by the single-character QUOTE macros (@ or '). again. Of course SOME uses of FEXPRS are essential, such as for SETQ and COND, at least if explicit references and coercion or call-by-reference are to be avoided. In fact we can look at them as the place where SYNTAX creeps into LISP, these special functions which must be declared globally, cannot easily be passed as arguments and must be known beforehand to the compiler. Notice also that, for COND at least, we wish to have a variable number of "arguments", always a potential source of programming errors.

DIFFICULTIES WITH IMPLEMENTATIONS

The problems we have mentioned above are all connected with the basic language (really LISP 1.5), though as we have noted current systems do differ at this level, whereas now we go on to criticise decisions made later, during implementation. Its interesting that the systems are still similar at this level, mainly for historical reasons.

6. Poor memory management. The different types of object(!) occupy different areas of memory and these areas cannot be dynamically expanded and contracted at all times. It can be quite serious to be part-way through a computation and have to restart the whole session because some area, typically a stack, is full. This is for "efficiency" reasons:

in the PDP-10 the address space is split up and the address of an object gives its type, thus allowing the use of the full 18(17?)-bit pointer without extra type bits in the word. Programs and stacks cannot be relocated easily because of the addressing structure of the machine. In fact it is rather suprising how little research has been done into hardware which would facilitate good engineering of LISP-like systems.

7. Input-output is a mess. Basically there is one port for input and one for output, and different channels set up by the user can be opened, closed and attached to these ports. The functions to set up channels are poorly designed, so that to write FOO so that (FOO X) reads from a file whose name is the value of X requires the use of EVAL! The reason should by now be clear: the functions were designed as FEXPRs so that in the common case of inputting a filename from the terminal no (QUOTE ...) had to be typed. The channels are not objects of the language, so atoms representing them are passed around instead. And because there are no closures, it is very hard to write programs that manipulate i/o streams e.g. concatenate them or run them in parallel.

8. Compilers and interpreters implement different languages! There are many assumptions that the compiler makes. Obviously it must know something of run-time bindings in order to produce good code, for example whether undeclared functions are EXPRs or FEXPRs, and must assume that functions do not modify themselves, but there is no systematic way to make any of this explicit.

FURTHER MORE "THEORETICAL" OBJECTIONS

9. LISP should really have call-by-name. This view has been put forward (by McCarthy and Weyhrauch at Stanford), to bring LISP more into line with the lambda-calculus, and to get some fixpoints which otherwise it does not compute (because their computations do not terminate). But although call-by-name CAN save computation, it is generally more expensive to implement, and its potential delaying of computation is usually a poor tradeoff. Why? Because programmers can understand the normal binding mechanism and understand what will be done and when. Thus if delayed access to some other environment is needed a function can be passed explicitly, provided of course that powerful-enough function-constructing facilities are available. Anyway programs that dont terminate usually do so because they have BUGS in them, and most people would like to find out SOONER rather than LATER!

10. LISP's functions are never really recursive! The way functions are normally defined is very dangerous. Ignoring for a moment the technical complication that functions are not values, the problem is that recursive functions make their recursive call via the NAME the function had in the DEFINING environment, so that if the value associated with this has changed, by assignment or by rebinding, the function no longer works. Tony Hoare calls this "recursion by punning". There is the LABEL facility but this is only for self-recursive functions (though this could be generalized), and anyway LABEL does not work properly in any current implementation. Thus:

```
(DE TRYFAC (N)
(COND [(ZEROP N) 0]
[T (* N (TRYFAC (- N 1)))]))
(TRYFAC 3)
** 6
(PUTPROP @FAC (GET @TRYFAC @EXPR) @EXPR)
{ (MOVD 'FAC 'TRYFAC) in InterLISP}
(DE TRYFAC (N) @LOST)
(FAC 6)
** LOST
```

Suppose we bound all free variables at the time of creation of the lambda-expression. There would still be a choice, the values current at the time, or the latest values in that environment. The latter is dangerous, but the former implies parallel assignment (at least for function definitions), which is difficult in an incremental system. For example, suppose we have FA and FB which call each other. A typical procedure is to define FA, but leave FB undefined and input its values through the error package, or define it as some trivial function, until FA seems to work. Then one goes on to define FB. We must either relink them each time, or (the LISP case) be faced with a situation where redefining can lead to errors. Normally in LISP functions are defined at the top level, so the problem does not usually arise, but as soon as we introduce a more powerful notion of function it becomes dangerous.

12. Scope rules. There are of course those who maintain that LISPs dynamic scope rule (for nonlocals) is just WRONG, that when you mention say, X, nonlocally you must surely have in mind which X you mean, and the only one which then exists is the lexically enclosing one. There certainly are cases where that is what you want e.g.

```
(PROG (SUM)
```

```
(SETQ SUM 0)
(MAPC @(LAMBDA (X) (SETQ SUM (+ X SUM)))
SOMELIST))
```

which could fail if MAPC had SUM as a local. The problem can be overcome by using closures appropriately, but it doesnt look very neat and of course it then mimics Algol-style binding. However in an interactive situation dynamic scope can be very useful e.g. if FOO does its output via PRINT then we can redirect output by evaluating calls to FOO in an environment where PRINT is rebound

```
(LAMBDA (PRINT) (FOO ...) NEWPRINT)
```

Also, dynamic scope makes it easy to link in new functions as they are created. Lexical scope "cures" the punning problem of the last section, but not in way which is really acceptable in an interactive and incremental system. It would be worth looking into programming style here and seeing what people "really do". To what extent are uses of nonlocals in actual LISP and POP-2 programs basically like the SUM example above? What reasonable notions of closure fit in with lexical scope?

WHY IS LISP STILL SO POPULAR?

The baroque nature of the InterLISP system illustrates precisely the low-level nature of LISP, in that almost none of the extensions is via the language's definition facilities, but are done by hanging things onto hooks in the LISP machine such as the error package. There are so many hooks that you can almost "hack around" many difficulties. Basically it isn't LISP that is popular, it's LISP SYSTEMS, the pragmatics of programming. The debugging and editing facilities are good, list cells fit nicely into one PDP-10 word, you can link to machine language and the compiler produces reasonable code. POP-2, which is linguistically superior (semantically at least - never argue syntax with LISP hackers!) has not caught on, precisely because its implementations are poor in these areas. People's choices of programming language are always interesting, especially in the academic domain where the choice is supposedly fairly free. For example BCPL is very popular at Essex for quite general programming, although it is a low-level language, for precisely the same reasons.

ACKNOWLEDGEMENTS

Many thanks to many people for interesting discussions, especially John Allen, Rod Burstall, Steve Hardy, Carl Hewitt

and of course the Essex AI Group.

REFERENCES

Allen J.R. (1976) **The anatomy of LISP.** New York: McGraw-Hill Publishing Co.

Burstall R.M., J.S.Collins & R.J.Popplestone (1971) **Programming in POP-2.** Edinburgh: Edinburgh University Press.

Friedman D. (1974) **The little LISPer.** Palo Alto: Science Research Associates.

Siklossy L. (1975) **Let's talk LISP.**

G. Attardi
C. Montangero
G. Prini

A HIGH LEVEL MACHINE FOR ARTIFICIAL INTELLIGENCE

The problem of defining abstract data types and realizing good representations for them is one of the major problems in computer science.

A first tentative solution to this problem was given in the late Fifties. It consists in defining several programming languages, allowing similar control regimes but containing different built-in 'problem oriented' sets of data types. Popular representatives of these first generation programming languages are FORTRAN, ALGOL60 and LISP. Usually a small set of basic data types (e.g. numbers and LISP atoms) and a few rudimentary structured data types (e.g. arrays and lists) are provided with such languages. The user is then supposed to select the most appropriate programming language for his problems.

This approach fails as soon as a problem does not fall into a restricted and well limited area (e.g. numerical and list processing), but deals with several types of arbitrary structures, as it often happens in A.I. or, in general, in non numerical applications (e.g. formula manipulation and compiler construction). First of all, the user is not explicitly invited to keep data definition clearly separated from data representation, and creates confusion between the logical design of data structures and their implementation, often leading to unstructured, unreadable and unreliable code. A good programing style which clearly distinguishes between definition and representation may considerably reduce these negative effects. However the user has to waste a lot of time in the routine work of implementing the standard primitives operating on his own data types. Even if this effort can be reduced by a careful choice of the programming language (e.g. trees can be better implemented in LISP than in ALGOL60), it cannot be completely eliminated. In any case, the newly implemented data types are not protected against misuse: functions intended to act on a specified type may be wrongly used for handling data of another type.

To overcome the drawbacks of 'problem oriented' programming languages, a second tentative solution has been proposed in the late Sixties. Programming languages have been proposed with both a set of built-in data types and a collection of powerful mechanisms for defining new data types. The most popular representatives of these second generation programming

languages are ALGOLW (Wirth and Hoare 1966), SIMULA67 (Dahl et al. 1970), ALGOL68 (vanWijngaarden et al. 1969), PASCAL (Jensen and Wirth 1974) and EL1 (Wegbreit 1974).
The user is supposed to define as many data types as he needs before (during in EL1) the execution of his programs.

Unfortunately, efficiency requirements impose that programs are (eventually in EL1) compiled into a low level machine language which does not allow fast type checking at run time, so type checking must be treated at compile time as far as possible (completely in ALGOLW and PASCAL). Smart compilers for such languages can be designed only at the cost of restrictions in the definition and treatment of data types. This is the case with PASCAL records, which do not allow unioning of types, and the severe limitations imposed on generic functions in EL1. Even though these restrictions may seem to be very weak, they strongly limit the user in many practical cases. On the other hand, SIMULA67 and ALGOL68 seem to allow a less restricted treatment of data types, but the necessary type checking at run time may intolerably compromise the efficiency of the object code.

The main reason why type checking is not efficiently performed at run time on standard computers is that data are not identified in the store, i.e. the same portion of memory may be interpreted to contain a number, a pointer, an instruction, etc. Identification of data may be easily obtained by tagging them, but efficient computations can be obtained only if tags and data are processed in parallel. This can actually be done in a microprogrammable computer, possibly with an ad hoc architecture.

With such an idea in mind, we have designed a machine, called SMOM (Structure Management Oriented Machine), having an architecture suitable to support some data management and control features. Note that the machine language of SMOM has not been conceived as a very high level programming language for immediate A.I. applications, but rather as a tool allowing fast and structured implementation of efficient programming systems for A.I. The machine language of SMOM is strictly recursive: however generalized control structures are easily programmable by implementing an appropriate model of multiple environments (Bobrow and Wegbreit 1975; Montangero et al. 1975), that is by defining suitable data types and functions operating on them. This may be done efficiently in SMOM because of the following features.

i) The powerful data type definition mechanisms of second generation programming languages are treated by the hardware.

ii) The type driven function call mechanism (see section III) allows control to be distributed over data in a uniform

and natural way.

iii) The machine language of SMOM does not contain primitives allowing concurrent programming: however SMOM code may be executed by the hardware with a higher degree of parallelism than is allowed by present computers (this issue deserves a deeper discussion, and is not treated in this paper).

SMOM has been inspired by SIMULA67 (Dahl et al. 1970), SMALLTALK (Kay 1975) and some recent papers on LISP (Deutsch 1973; Montangero et al. 1975). Some ideas underlying the machine language of SMOM have been tested by using an extension of MAGMA-LISP (Montangero et al. 1975) capable of supporting data definitions and type driven function calls. The full power and the relevance of these features for A.I. applications have been pointed out while implementing a proof generating system for a typed λ-calculus, and are discussed in (Aiello et al. 1976).

Section I contains an overview of the main features of SMOM. In sections II and III the data definition and control features of SMOM are presented in some detail. Section IV contains a brief discussion on type driven calls and distributed interpreters. Section V is devoted to the presentation of some ideas useful in a software simulation of SMOM. Finally, in section VI a possible hardware architecture of SMOM is sketched.

I - WHAT IS SMOM?

SMOM is a high level machine. Its language has been designed to allow both interesting hardware architectures to be studied and efficient software simulations to be realized on standard byte oriented computers.

The ideas underlying SMOM may be summarized as follows.

i) Each datum is tagged with its type.

ii) SMOM (like LISP but unlike almost all second generation programming languages) does not allow records and pointers to be treated as separate entities.

iii) ((Mutually) recursive) data types are definable at run time in SMOM by the powerful facilities present in second generation programming languages.

iv) Memory allocation and management is completely transparent to the user already at machine language level.

v) Programs are held very compact by adopting the reverse polish notation as presented in (Deutsch 1973).

vi) No sharp distinction is made between data and functions. Data of a specified type may be used as functions provided

that an interpreter has been programmed (in any other type for which an interpreter already exists) for that type. While used as functions, data of different types are allowed to be freely intermixed in SMOM, and the appropriate interpreter is automatically retrieved and activated each time.

vii) Fast execution of generic functions (i.e. functions which are defined on several types, possibly with a different behaviour) is obtained via the type driven function call mechanism discussed later (see section III).

In spite of vi), we find it convenient to keep the treatment of data-as-data (i-iv) separated from the exposition of data-as-functions (v-vii). Cross references between the two sections are used only if strictly necessary.

Many restrictions present in nowaday programming languages derive from the choice of handling data directly, instead of manipulating them through pointers. As an example, if a string has been pushed onto a stack (in the sense of ALGOL), and it is not at the top of the stack itself, it cannot be replaced by a longer string unless all data over it in the stack are moved to create enough room. Since such transformations of the stack cannot be easily and efficiently implemented on standard computers, designers generally prefer to eliminate this kind of nastry situations by imposing restrictions on the language. If pointers, instead of data, are pushed onto a stack, no restrictions have to be introduced provided that all pointers are of the same length.

Unfortunately, the following drawback seems to arise with tagged data handled through pointers: the frequent operation of retrieving the type of a datum without further processing the datum itself (for instance when testing the type of a datum to determine which one in a set of actions is to be undertaken), requires one storage access. The solution adopted in SMOM consists in tagging pointers instead of data.

All data are handled through data descriptors (dd's) in SMOM. A dd is a pair (T,P) where T is the type of the described object and P is a pointer to the datum itself. A dd (T,P) is also called a T descriptor, for short.

II - DATA TYPES IN SMOM

Data types may be either basic or structured. Basic data appear to have no inner structure: integer numbers, characters, etc. fall into this category. Since the amount of information needed to store data of a basic type is small, they can be directly stored into the pointer part of dd's, thus saving space and storage accesses. Structured data are made up of components: strings, LISP cons cells, etc. fall into this cathegory. They are implemented as ordered sets of dd's.

Both basic and structured data types may be either built-in or user defined in SMOM. Data type definitions are supported by the notion of class. A class is a structured datum which contains all information relative to a specific data type. So there is the class of numbers, the class of characters, etc. Since classes are data, a class of classes exists which contains all information relative to classes. The components of a class are the following.

i) An empty datum. Empty data are used as terminators for recursively defined data types.

ii) A template. It is a datum that records the structure of data belonging to that class, the name of the component fields and possibly the type restrictions imposed on them. The template is empty for basic data types.

iii) An unspecified number of registers. Registers contain information common to all elements of a class, such as, for instance, the definitions of functions operating on the data belonging to the class. Some standard functions (such as READ, PRINT, ALLOCATE, GARBAGE and the functions for selecting components of structured data) are implicitly defined when a class is defined. A function definition is stored in a register having the same name as the function itself. New registers may be created by giving their names and (possibly) their initial content either in the class definition or by editing the class.

In fact, a new data type may be defined simply by creating the class which contains all information about it. All objects, including classes, may be created either implicitly, by reading them, or explicitly, by means of a constructor function having the same name as the class associated with that data type.

The external representation of a datum is normally a list of the external representations of its components (if any), preceded by the name of the class to which the datum belongs. Special notation is used for some built-in data types, such as integer and real numbers. For instance, the complex number 3-4.25i is externally represented as

```
(COMPLEX 3 -4.25)
```

since 3 and -4.25 are the standard representations of the integer 3 and of the real -4.25 respectively. Obviously, the user may define his own I/O functions through primitives for manipulating characters and buffers.

Templates allow a great definitional power: in particular unioning of types, enumerative types (borrowed from PASCAL), structured types with a variable number of components, mutually recursive data types etc. are allowed without the restrictions

imposed by most second generation programming languages (dd's eliminate almost all the difficulties arising when implementing structured data in the usual way). Since the technique for managing type definitions through templates are fairly standard, they are not discussed here. We only show, as an example, the definition of the class COMPLEX.

```
(CLASS COMPLEX
       (TEMPLATE (REAL-PART (ONEOF INTEGER REAL))
                 (IMAG-PART (ONEOF INTEGER REAL))  )
       (REGISTER PLUS ...)
            .
            .
            .
       (REGISTER CONVERT-TO-POLAR ... )    )
```

III - CONTROL IN SMOM

In order to process a datum, a dd for it must be pushed onto the so called A(rgument)-stack.

Two special built-in data types, i.e. microprograms and programs, are present in SMOM.

Microprograms act as functions: they take their arguments near the top of the A-stack and push a value onto the A-stack itself after having popped their arguments. Several microprograms are built-in when SMOM is generated: they include functions acting on the A-stack, jump operators to be used in programs, etc. Many others are automatically generated whenever a class is defined: they include constructor functions, selector functions for the various components of data belonging to that class, standard I/O functions, etc. We have no room to describe all microprograms in detail here. The example near the end of this section may help in realizing what they do.

Programs are sequences of bytes, each one being interpreted as the internal name of a class register or as an operand. A program is executed in the following way: a byte is fetched, according to a program counter, and it is first interpreted as the internal name of a register in the special class ANY (several built-in microprograms are contained in registers of this class). If such a register exists, the datum it describes is called as a function according to the following rules.

i) If the datum is a microprogram, it is directly executed (it may fetch some bytes of the calling program and use them as operands.

ii) If the datum is a program, the continuation point (i.e. the current program counter) of the calling program is pushed onto the so called C(ontrol)-stack and the called

program executed.

iii) Otherwise the datum is pushed onto the A-stack and the standard function APPLY is called (after having pushed the continuation point of the calling program onto the C-stack). By the type driven call mechanism (see below), the function APPLY is searched in the class to which the datum itself belongs. Thus each datum may be used as a function, provided that an APPLY function (i.e. an interpreter) is defined for it.

If the class ANY does not contain a register with the same internal name as the fetched byte, such a register is searched in the class of the datum currently at the top if the A-stack. If such a register is found, processing goes on as in the previous case, otherwise an error is generated. The call mechanism just described is called type driven, since the type of the datum at the top of the A-stack may 'drive' the access to the code of the called function.

We show a sample program which performs addition of two COMPLEX numbers as previously defined. Two dd's for COMPLEX numbers are supposed to be at the top of the A-stack when this program is called.

```
(PROG (PUSH,2) (REAL-PART)
      (PUSH,2) (REAL-PART) (PLUS)
      (PUSH,3) (IMAG-PART)
      (PUSH,3) (IMAG-PART) (PLUS) (COMPLEX)
      (SQUEEZE,2)
      (RETURN)  )
```

Only few remarks are needed to understand the program. (PUSH,n) pushes the n-th (n>0) dd starting from the top of the A-stack onto the A-stack itself without popping arguments. (SQUEEZE,n) pops n dd's under the top of the A-stack, pushing the top itself n places down.

If such a program is stored in the PLUS register of the class COMPLEX, if the PLUS registers of the classes INTEGER and REAL contain the code for adding two INTEGER or REAL numbers, and if a PLUS register does not exist in the class ANY, the type driven call mechanism automatically retrieves the right PLUS function when a dd for an INTEGER, REAL or COMPLEX number is found at the top of the A-stack and PLUS is called (for simplicity, in the above program only the addition of two COMPLEX numbers has been considered: code for adding two numbers of different type may be written as well).

Let us remark that, in the above example, the code for adding two INTEGER and REAL numbers could be of any kind (microprograms as well as a datum of a user defined type) without influencing the structure of the program. In particular, data of a given type may be 'compiled' into data of

another type without influencing the programs which use them as functions.

Registers with the same name may exist in several classes, ANY included. In such a case, access to the class ANY may be bypassed and the corresponding function may be called directly by the type driven mechanism via the function (DRIVEN-APPLY, name), which requires the name of the function to be called as an operand. This allows programs to be efficiently implemented having a common initial section (to be stored in the class ANY), but having a different behaviour which depends on the type of some datum after that section has been executed. A (DRIVEN-APPLY, ...) will then appear at the end of the common section, the operand of DRIVEN-APPLY being the name of the calling function.

Finally, a remark on standard I/O functions. They are useful to do system dumps without making use of addresses, which are treated only at the hardware (i.e. microprogram) level in SMOM. Common substructures and circular ones may be correctly printed and read by these functions. For instance, the LISP structure resulting from the execution of the following LISP function

```
(LAMBDA ()
        (PROG (X Y)
              (SETQ Y (CONS NIL NIL))
              (SETQ X (CONS Y Y))
              (RPLACA Y Y)
              (RPLACD Y X)
              (RETURN X)  ))
```

would be printed as

```
(CONS (CONS (! (PUSH,1))
            (! (PUSH,3))  )
      (! (PUSH,2) (CAR))  ))
```

Standard input functions push a dd for a datum to be read onto the A-stack before the components of the datum are read, thus allowing backwards references. The symbol ! forces evaluation of the program which follows it during input. Such standard I/O functions are synthesized by SMOM, even if printing routines are a bit sophisticated in order to allow common substructures to be detected.

IV - TYPE DRIVEN PROGRAMMING AND DISTRIBUTED INTERPRETERS

By the notion of class and the type driven call mechanism, SMOM can be considered as a distributed extensible machine with a distributed hardware interpreter for built-in data types and allowing distributed software interpreters to be written for user defined data types. In fact, each data type can be seen as an independent module, having its own inter-

preter, its own functions specifying its behaviour and even its own memory. The built-in function call mechanisms automatically retrieve the right pieces of code each time according to the type of a datum, thus forbidding functions written for a data type to act on data of a different type.

The treatment of data types provided with SMOM and its sophisticated function call mechanisms allow structured programs to be developed by stepwise extension of the data and control structures (Aiello et al. 1976). Actually the data definition and control aspects of programs may be developed and implemented in parallel in SMOM. Programs may be easily extended, since adding a new data type to a program implies only addition of new functions: no changes are to be done to make the new features compatible with the old code. As an example, adding a new data type to a programming language for which an interpreter has been already written does not require modifications to be done to the interpreter which handles, say, assignment operations: it is just necessary to define the new assignment primitives for that type.

As an example of this kind of structuring, consider how a LISP interpreter may be implemented. The classes ATOM and LIST are defined first: SMOM actually extends itself by synthesizing a number of microprograms implementing the standard primitive functions. An APPLY function is then defined in the machine language of SMOM for both classes, thus realizing a distributed LISP interpreter (of course, a number of utility functions are to be defined in addition to the APPLY functions: they are defined in the appropriate class). LISP functions may now be called freely by SMOM code, since the APPLY function for LISTs or ATOMs is automatically retrieved and invoked if such a data type is found in the class register corresponding to an instruction in a SMOM program. This way, LISP code is fully compatible with SMOM code, and the same will be true for any further level of software being built over LISP, provided that the appropriate interpreters are defined each time.

V - IDEAS FOR A SOFTWARE SIMULATION

We have not yet built a piece of hardware called SMOM. Only a software simulation is in progress. An efficient SMOM interpreter may be realized on any byte oriented computer (in this case, microprograms are programs in the language of the host machine). We present some ideas adopted in our implementation, which seem to be useful if the host machine operates in a paged environment.

Memory is divided into pages (not necessarily coincident with the pages of the host machine) which are initially held on a global free list. Pages may be allocated to increase the size of either the private memory of a structured data type or

a common heap (which is treated as a linked list of pages). Basic data types do not require private memories, since their descriptors are not pointers and are directly stored as components of structured data.

Each structured data type (except classes which are treated differently) is associated with a partially free page containing only data of that type. Allocation of new data is performed using that page. When the partially free page is full and a new datum is to be allocated, a new page is taken from the global free list. The allocation algorithm for new data depends on whether all data of the type under consideration have the same number of components (fixed length data) or not (variable length data). In the first case, the page is considered as a collection of equal blocks, each one capable of containing as many dd's as needed, possibly in a packed form. Allocation takes places by using one of these blocks. In the second case, allocation takes place as if the data type had a single component, called indirect reference. When a datum is to be created, it is put into the common heap, possibly in a packed form, and an indirect reference is allocated which points to it. By this technique, the task of compactifying the whole memory after a garbage collection is greatly simplified.

As for classes, they are treated in a special way with respect to storage management, even though they are variable length data. Class registers are to be accessed fast, since their main purpose is to hold function definitions. Since functions are indexed by both a register name and a class name, and since it is likely that there will be a number of 'general purpose' functions defined for almost every class (e.g. I/O routines) and a number of 'specific' functions defined for few classes (e.g. the function CAR is defined only for ATOMs and LISTs in the MAGMA-LISP interpreter), the following organization allows rapid access in a reasonable amount of space. First the register name is used to access a bucket array: each bucket is either a pointer to an array indexed by the class name and containing the class registers (a trap if the expected register is not found), or it is a small table of pairs class-dd, which is searched according to the key class name. The choice between array and small table is performed according to the number of classes which contain a register with the name under consideration. In our software simulation the threshold is three.

VI - CONCLUDING REMARKS

We have here described the machine language of SMOM and we have discussed some ideas employed in a software simulation, currently in progress. A hardware realization may be even better organized. For instance, the type driven function call mechanism and access to class registers may be implemented by

algorithms having a high degree of parallelism and using small associative memories. The memory of SMOM may be made less passive. It may be built out of microprocessors (having small associated memories) which operate in parallel under the control of a supervisor processor. Each microprocessor is allocated for a single data type and contains all functions relative to that type. The supervisor activates the microprocessor containing the datum to be processed and sends it the name of the function to be computed. After computation, the dedicated processor returns the result to the supervisor. Since the supervisor does no significant computation, it can be used to prefetch instructions following the current one, thus allowing actual parallel computations of the various dedicated microprocessors. This issue deserves a more accurate discussion, since the full power of this approach can only be reached if the A- and C-stacks are made more flexible. In fact, if these stacks are implemented by actual stacks (i.e. contiguous memory cells), prefetching cannot go on very far, since many modifications near the top of the stacks are to be completed before the execution of another function. If the A- and C-stacks are implemented by linked trees, a sophisticate algorithm may be defined which allows an unlimited number of instructions to be prefetched without restrictions.

ACKNOWLEDGMENTS

We should like to thank all of our colleagues for many stimulating discussions and suggestions. Special thanks are due to Dr Franco Sirovich, who participated in an earlier phase of this research, and to Dr Marco Vanneschi, who communicated us his exciting philosophy about hardware.

REFERENCES

Aiello,L., et al. (1976) Recursive data types in LISP: a case study in type driven programming. Proceedings of the Second Symposium on Programming. Paris.

Bobrow,D. & Wegbreit,B. (1973) A model and stack implementation of multiple environments. Comm. ACM 16, 591-603.

Dahl,O., et al. (1970) The SIMULA67 common base language. Norwegian Computing Centre Publication 5-22. Oslo.

Deutsch,L.P. (1973) A LISP machine with very compact programs. Proceedings of the Third International Joint Conference on Artificial Intelligence. Stanford.

Jensen,K. & Wirth,N. (1974) PASCAL user manual and report. Lecture Notes in Computer Science 18. Springer Verlag.

Kay,A. (1975) Personal Computing. Proceedings of the Meeting on Twenty Years of Computer Science. Pisa.

Montangero,C., et al. (1975) MAGMA-LISP: a 'Machine Language' for Artificial Intelligence. Proceedings on the Fourth

International Joint Conference on Artificial Intelligence. Tbilisi.

Wegbreit,B. (1974) The treatment of data types in EL1. Comm. ACM 17, 251-264.

vanWijngaarden,A., et al. (1969) Report on the algorithmic language ALGOL68. Numerische Mathematik 14, 79-218.

Wirth,N. & Hoare,C. (1966) A contribution to the development of ALGOL. Comm ACM 9, 413-432.

R. Bornat

REASONING ABOUT HAND PRINTED FORTRAN PROGRAMS

This paper describes current work on a part of the Essex Vision project [2]. As part of a study of how knowledge can intervene in the process of visual perception I am investigating how knowledge about FORTRAN programming can best be used in a program which will read hand printed FORTRAN programs. Work to date has concentrated on the plausibility and limitations of the approaches which have suggested themselves and this paper outlines one example protocol ([1] gives further examples). Work on building a program to emulate human protocols is, at the time of writing, in the early stages but it follows the lines sketched out below.

THE PROBLEM

The system* is presented with a description of the layout of a FORTRAN program in terms of the position and size of 'blobs' of writing - see figure 4 for an illustrative example. The only differentiation provided is between 'alphanumeric-and-bracket' and 'operator' ('+', ', ', '*', '=' etc.) blobs. This representation is produced from digitised input by a program described in [3].

The system has to use this information as the basis of a dialogue with a character reader (current work on this is described in [5]). It must build up an understanding of the program it is reading in order to be able to make useful predictions or adjudications about character identities. It can use knowledge - 'facts' about FORTRAN - to generate hypotheses about the program and to link features of separate areas of the program via consistency arguments [8], [10], [6], [2].

I have chosen at this stage of development to ignore the constraint that the system must undergo a useful dialogue and instead I have investigated how much can be discovered from the representation sketched above. The protocol below assumes that the reasoner can ask simple questions of a character recogniser, but not vice-versa, and flights of fancy based on slight evidence are therefore possible.

---

*I distinguish between my 'system' and the 'program' which it is reading. I might have said 'data', but I would rather emphasise the restricted nature of our visual input.

HUMAN COMPETENCE

We first chose our problem because it seemed to us to illustrate very neatly how consistency arguments could help in perceiving visual scenes. If we look at a FORTRAN coding sheet, we can see ways in which we can consciously use consistency arguments to discover the identity of (say) some or all of the keywords on the sheet without reading them at all - for example (at least in the very short programs we are reading) the last line has to be 'END', the line above it (if it is an executable statement) has to be 'GOTO', 'STOP' or 'RETURN'. Less trivially, we can reason about statements from their shape

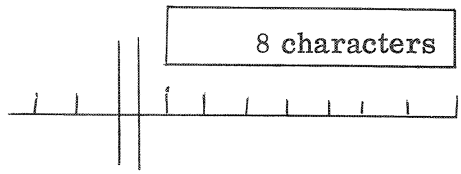


Figure 1

might well be 'CONTINUE' because it is an eight-character blob - if we see

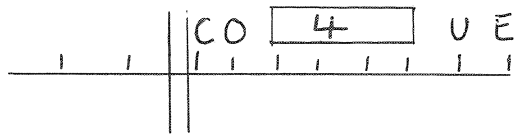


Figure 2

in a position in which we are expecting the terminal statement of a DO we would probably look no further. An illuminating anecdote to show how this sort of thing happens in some situations is the following:

I was shown the program:

```
A = some expression
B = some other expression
IF A=B GOTO 1
(a statement followed which printed out the
values of A and B)
```

Figure 3

The evidence was overwhelmingly that 'A' was equal to 'B', yet, as the indignant programmer protested, the 'GOTO' wasn't executed. At the time I was extremely familiar with FORTRAN syntax, yet it was several minutes before I noticed that the apparent 'IF' statement was in fact a pathologically legal assignment statement!* Context had swamped the local evidence.

Shown a diagram like that in figure 4, a human can assign roles and meanings to most of the blobs on the sheet. When reading programs directly, humans build up a representation of the program which is often quite complex - asked to justify a character they might reply 'I know it is 'R' because the word is 'RED'. It's not 'AED' because it's obviously 'RED' in the READ statement and this line is checking that it is in bounds'. Conversations with local punch operators show that they use extensive informal knowledge of FORTRAN and of programming practice - the best evidence perhaps is how difficult they find it to punch programs with deliberate errors.

Like Weir [9] I deny that perception can be divided into stages of (i) input, (ii) value judgement. Assignment of purpose to visual cues, which is what my system attempts, is an essential condition for perception. There follows an example protocol of (a human simulation of) the activity I am hoping to implement. It would perhaps be instructive if the reader paused at this point to consider whether she can assign roles to the blobs illustrated before reading the argument which follows. The knowledge (FORTRAN 'facts') which I have used to construct these arguments appears in full in [1]. After the protocol I discuss some features which it has illustrated and I report on the current state of work on the program which I am writing to achieve some of these inferences as part of the overall Essex Vision project.

PROTOCOL OF A PROGRAM ABOUT RECIPROCALS AND SUMS OF SQUARES

1: Line 1 could be a comment or a labelled FORMAT (everything else is rejected by lack of punctuation). (Visually, the arrangement of labels in columns 4 and 5 on every other line suggests that line 1 is a comment, but it would be over-ambitious to ask the program to infer that at this stage, I think.) If a character reader can tell the difference between a 'C' and a digit, we can discover that line 1 is a comment anyway. If not, it should be able to see that the first blob doesn't read 'FORMAT' (it reads 'PLEASE').

*Spaces are ignored by most FORTRAN compilers, so it is an assignment of 'BGOTO1' to 'IFA'.

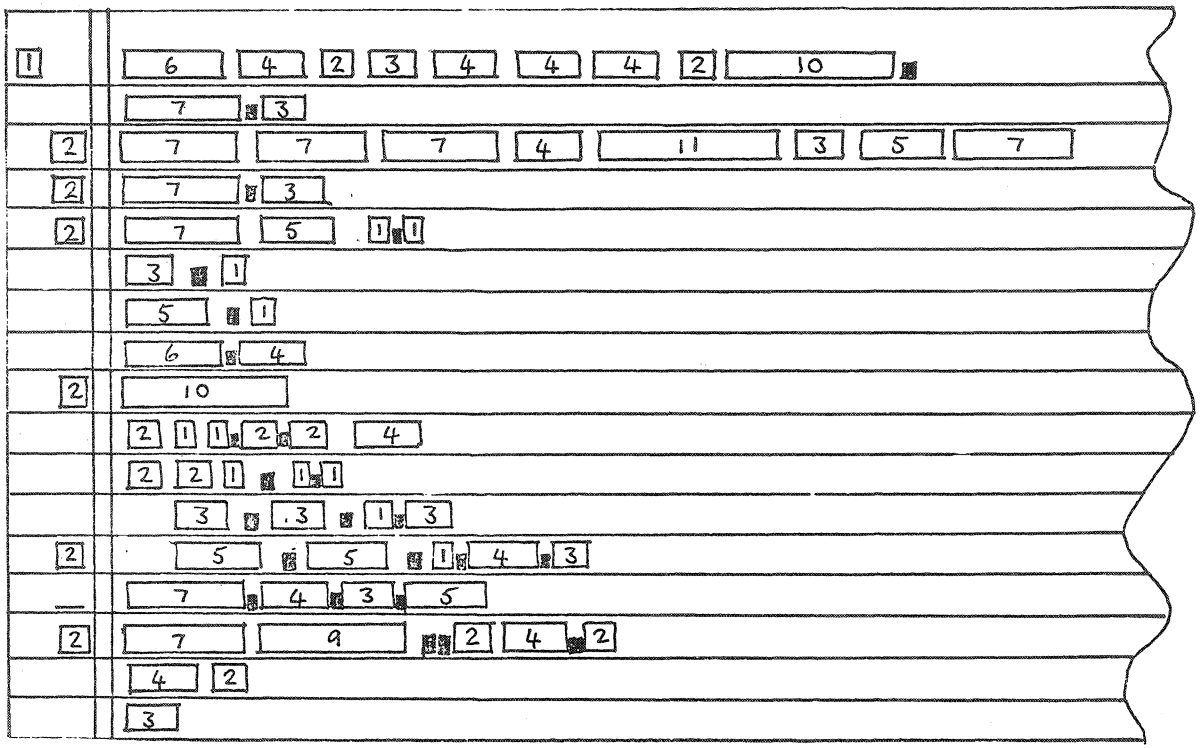

1 6 4 2 3 4 4 4 2 10
7 3
2 7 7 7 4 11 3 5 7
2 7 3
2 7 5 1 1
3 1
5 1
6 4
2 10
2 1 1 2 2 4
2 2 1 1 1
3 .3 1 3
2 5 5 1 4 3
7 4 3 5
2 7 9 2 4 2
4 2
3

Figure 4

2: Line 2 could be COMMONx, WRITE(x, GOTO(xx, READ(xx or an assignment statement involving an array. No array has been declared, so the assignment statement is out. A character reader should be able to distinguish the possibilities above - in fact it is WRITE(x. The remaining blob could be 'y)z' or 'yy)'. The first possibility demands a previously defined identifier - there isn't one so the statement must read WRITE(x, yy).
3: There must be a FORMAT statement labelled with the 'yy' on line 2.
4: Line 3 must be 'FORMAT(' - once again the lack of punctuation rules out anything else. The adjacency to line 2 suggests that this line has the label mentioned there.
5: Line 4 is labelled. The same reasoning as for line 2, and the same adjudication by a character reader, tells us that this is 'WRITE(a, bb)'.
6: Line 5 is another FORMAT*. Once again it probably has the same label as that mentioned on the line above.
7: Line 6 is an assignment statement. The right-hand side must be a constant. The left-hand side must be a variable name (we know that there are no arrays in the program).
8: Line 7 is another assignment statement. Once again, the right-hand side must be a constant (it is too long to be the same name as that on the previous line). The left-hand side must be a different variable name - we now have a four-character name and a three-character name.
9: Line 8 is 'READ(x' or an assignment statement. In fact it is READ - the identity of the punctuation mark as a comma is enough to tell us. The rest of the statement must be 'cc)d' because we only have two-character labels. Thus we have another identifier, this time a single character one.
10: Line 9 could be 'BACKSPACEi', 'ENDFILEii', 'FORMAT(hh)', 'REWINDiiii', 'PAUSEiiiii', 'WRITE(....' or 'READ(.....'. Since every I/O statement so far has been immediately followed by its own FORMAT, we might hypothesise that it is a FORMAT

---

*The full list of possibilities, going just by blob length, is COMPLEX ENDFILE INTEGER LOGICAL ASSIGN COMMON DOUBLE (precision) FORMAT RETURN REWIND BLOCK (data) PAUSE WRITE CALL DATA GOTO READ REAL STOP END DO IT. Most are eliminated by position in the program, impossibility of fitting the syntax to the blob shapes or by applications of the fact that alphanumeric words are almost always separated by spaces. There is a surprising amount of visual redundancy in FORTRAN, when you look for it.

statement associated with line 8. If we have read the identifier on line 8, we might further be able to hypothesise that the format specifier is 'I' since the identifier is 'N'.
11: Line 10 is a logical IF. The pattern in the middle of the statement is a logical operator - .EQ., .LE. etc. - and the statement which follows is STOP. Thus the statement is

IF (x.oo.y) STOP

- one of x or y must be the single character identifier on line 8.
12: Line 11 is a DO statement. If we look for the terminating label, we notice systematic indentation of lines 12 and 13. Line 14 isn't labelled or indented, so probably line 13 is the terminating statement.
13: Line 12 is an assignment statement. The three-character identifier on the left is probably that from line 6. The right-hand side can only contain constants, function calls or previously-initialised values, so the three-character blob on the right is probably that identifier as well.
14: Line 13 is an assignment statement, terminates the DO, and probably involves the five-character identifier from line 7.
15: Line 14 is either a WRITE or a READ. Given that we have so far only seen single character unit numbers, it is probably WRITE. We can say that the statement reads

WRITE(u, ff)x, yyy, zzzzz

- each one of the identifiers which has so far occurred in the program.
16: Line 15 is FORMAT. Once again, we may hypothesise that it links up with the previous line. Within the statement, we can see an initial Hollerith constant and then what may be three format specifiers. The final one is only a single character, so it must be '/'. The previous one is 'Innn' or 'nInn' and the one before that, if it isn't part of the Hollerith, is 'In'.
17: Line 16 is 'GOTO xx'. So far we only know one labelled executable statement - the WRITE on line 4 - so it is probable that line 4 is the destination.
18: Line 17 is END. If we haven't already confirmed it, the destination of the GOTO on line 16 must be line 5.

DISCUSSION OF THE PROTOCOL

Despite the artificiality induced by using such a small example, several points are illustrated. I have had to linearise the protocol for reasons of space, destroying evidence of how attention 'skips

around' the sheet. However it is important to emphasise that my system will not operate in 'hypothesise-and-test' mode but rather more like the 'wait-and-see' mode of Marcus [7]*. The hypothesise-and-test mode produces a top-down search: for example finding 'END' causes us to hypothesise a RETURN in the line above. A test for counter-evidence to this hypothesis might cause us to look above for an earlier 'END' - if we don't find one then the line above the current line can't be 'RETURN'. Such a visual wild-goose chase is counter to my intuition and I prefer to operate in terms of 'expectations': finding a cue 'α' raises our expectation of finding 'β' elsewhere (as well as denying the possibility of 'γ' perhaps). Expectations which are overwhelming may be investigated immediately - thus, for example, it might be reasonable to look for the END line as soon as we find the first line of a subprogram.

The reasoner's internal representation structures will be some kind of tree - a hierarchical description of the program components in terms of the roles they play. Some of this will be like that of a compiler (program-subprogram, DO loop etc.) but some will reflect program schemas useful to the reasoner's analysis. Having recognised

```
READ (x, yy)a
IF (b.oo.c) STOP
```

Figure 5

in lines 8 and 10 of figure 4, we may hypothesise that we have found a piece of program which is reading something and checking a termination condition. Thus 'b' or 'c' is likely to be the same as 'a'. The internal representation will reflect the (supposed) purpose of these two lines.

Most inter-line consistency information is generated from knowledge of control (like narrative flow in a story). The schema of figure 5 is an example. Another is the restriction which we can place on use of uninitialised identifiers before the first unconditional transfer of control. Generating inter-line consistency checks makes it more essential to use a structured internal representation rather than a 'soup'of assertions because counter-evidence to some hypothesis may be generated from other locations on the sheet - the

---

* The difference is like that between deterministic and non-deterministic parsing in a compiler.

reasoner then needs to look again, via the structure, at the evidence which generated the original hypothesis.

THE MARK1 PROGRAM

The program is written in EVIL [4] and currently uses CONNIVER-like demons to recognise and assess evidence. It generates an initial estimate of statement identity from initial blob-size and filters it using knowledge of syntax. To date use of flow of control knowledge has been implemented.

In order to discriminate between statements, MARK1 proposes 'acid tests' for a character reader. It is surprising how easy it is to tell statements apart: for example an assignment statement can usually be picked out just by checking the '=' sign. Pathological CALL and FORMAT statements with '=' signs in embedded Hollerith constants confuse MARK1 just as they confuse humans! For example:

```
FORMAT(12H)  =  (AB*CDE+10)
```

Figure 6

At present MARK1 treats some knowledge of probabilities as certainties. For example, humans usually separate alphabetic words with a space - you rarely see 'COMMONABC' for example. Later versions must be able to retract hypotheses based on such an assumption.

Use of CONNIVER-like demons allows insertion of additional knowledge relatively easy, of course, but an assertional data-base makes it difficult to trigger demons on complex conditions such as (A and B butnot C). The structured (frame-like) data-base described in 5 seems more appropriate for such decisions and I anticipate that later versions (MARK2 ... MARKoo) will use it.

REFERENCES

Bornat, R. FORTRAN knowledge and FORTRAN reasoning, University of Essex Computing Centre memo (in preparation).

Bornat, R. & Brady, J.M. Using knowledge in the computer interpretation of FORTRAN coding sheets (to appear in IJMMS, 1976).

Bornat, R. & Brady J.M. Finding blobs of writing in the FORTRAN coding sheets projects (submitted to AISB Conference 1976).

Bornat, R. & Wielinga, B.J. The EVIL programming system, University of Essex Computing Centre memo (in preparation).
Brady, J.M. & Wielinga, B.J. Seeing a pattern as a character, (submitted to AISB Conference 1976)
Freuder, E. MIT Ph.D. Thesis (1975).
Marcus, M. Diagnosis as a Notion of Grammar. Proc TINLAP Conference, MIT, 1975.
Waltz, D. MIT Ph.D. Thesis (1973)
Weir, S. Symbolic description and the constructive nature of perception, DAI research report 13, Department of Artificial Intelligence, University of Edinburgh.
Winston, P. The MIT robot, Machine Intelligence 7, Edinburgh University Press.

R. Bornat
J.M. Brady

FINDING BLOBS OF WRITING IN THE FORTRAN CODING-SHEETS PROJECT

The Fortran coding-sheets project is an AI vision experiment which was inspired by the experience of a number of AI projects, particularly those carried out at MIT (Minsky and Papert 1972; Winston 1975) and Sussex (Clowes 1971, 1973). For a general introduction to the project see (Bornat and Brady 1976). We view visual perception as a process of constructing an interpretation of a picture of a scene, not only from visual 'cues' within the picture, but also by using knowledge of what it is expected that the picture represents. Similar views about perception, again substantially influenced by Clowes (1973), are expressed by Weir (1975, 1976). Viewing perception as a process makes the organisation of that process, in terms of the sub-processes which make it up, of paramount importance. In this respect, we have been influenced by descriptions of 'heterarchical control' (Minsky and Papert 1972; Winston 1972; Freuder 1975; Bornat and Brady 1976), which connotes a system in which processes genuinely coexist and interact.

Our particular 'toy world' is that of coding sheets on which are hand-printed Fortran programs. There is a great quantity of knowledge which we can deploy about our world, which seems to fall naturally into two sections: knowledge about Fortran programming and knowledge about the formation of writing. It is convenient to divide our collection of processes into two large groups and to allow the groups to interact (figure 1).



Figure 1. Top level view of the program

The processes based on writing (the 'character experts') make detailed examination of the actual hand-blocked characters, while the Fortran processes (the Fortran reasoner) at present work on uninterpreted 'blobs' which correspond to strings of characters. The Fortran processes are described in Bornat (1976), and the writing processes in Brady and Wielinga (1976a, 1976b).

This paper describes the design and implementation of a program which produces a compressed view of the picture to serve as initial data for the Fortran reasoner. It also isolates the regions to be worked on by the character experts. The compressed view describes the input in terms of the natural coding sheet coordinates (lines and character positions) indicating where the 'blobs' of writing are, what length the 'blobs' are and which of them might be operator symbols ('+', '*', '=', etc.). At present this amount of information appears to be sufficient to suggest hypotheses about the writing to the rest of the program - in particular we don't seem to need to identify brackets or particular alpha-numeric characters at this stage.

The paper concludes with a discussion of why the authors are dissatisfied with the blob finding program except as an existence proof that it is possible to build one. The problem of when to involve knowledge in a vision program is raised.

THE PICTURE IMAGE

The principles of the organisation of our program, sketched in (Bornat and Brady 1976; Brady and Wielinga 1976a, 1976b; and Bornat 1976), mean that it cannot be constrained to an ordered sequential scan across the picture. Many of our processes require access to any part of (a representation of!) the entire image. Conversely the 'character experts' need to be able to investigate the strokes making up the writing in some detail. Thus we need a high-resolution representation of all of the picture at once. These twin requirements put a great strain on our computer service (a DEC KI-10 with 256K memory and limited backing store) and we have had to compromise. We work with a digitised representation of photographic transparencies, developed to give as faithful a range of light levels as possible. Each character space on the coding sheet corresponds to about 20 x 20 pixels - about one-sixth the resolution of conventional OCR systems. Intensity resolution is 256 light-levels at each point. The entire picture of a coding sheet is about 1.5M bytes (375K words of DEC-10 storage).

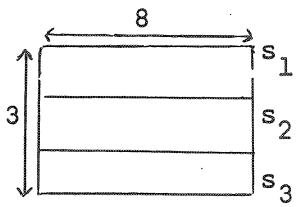
Finding the blobs of writing is like glancing quickly over the sheet, or looking at it without bothering to focus attention on the individual characters. For this purpose, we don't need all the spatial resolution we have available, so we shrink the picture to one-sixteenth of its original size. We pick the 'blackest' element of each 4 x 4 window in the picture to produce the shrunken version. This method preserves the black/white contrast better than an averaging process which would smooth the data too much. The small picture is then about 100K bytes or 25K words - a more reasonable size for

processing. At the shrunken resolution each character position is about 5 x 5 pixels. Subjectively, the impression is like that of a coding sheet 10 or 20 feet away - it's easy to see lines and the location of the writing, but little else. (Actually, you can read the writing still if you have a good idea what it says! Handwritten areas - like signatures - are easiest, surprisingly.)

We proceed in stages - first we find the coding-sheet lines, then we estimate the character position 'grid' and finally we find the blobs of writing.

FINDING THE LINES

The line finding is based on a program which is first applied to discover the horizontal lines and then to discover the vertical lines. The line finding has a similar overall structure to the Binford-Horn line finder (1971) in that it first pushes a 'trolley' to find the line points, and then group the line points into lists. The 8 x 3 trolley is roughly equivalent to using two edge masks (Marr 1974) of different panel width. However the kinds of problems we encounter in fitting the lines when there is writing on the sheet are rather different to those involved with plane-faced polyhedra. We can assume, first of all, that the horizontal lines are roughly horizontal. An 8 x 3 trolley (figure 2) is pushed over the picture, to find points at which the gradient $|g_{21}|$ or $|g_{31}|$ exceeds a threshold gradthresh. If it does, the fact is recorded by setting a corresponding bit in a matrix and the blacker of the s_i's contributing to the large gradient is recorded in a histogram. Typically, this histogram exhibits a sharp peak, which provides an accurate initial estimate of the contrast, and may be used later in finding blobs (see below).



s_1 = sum in 1st row $s_i/8 = sav_i$, $1 \leq i \leq 3$

$g_{21} = (s_1 - s_2)/8$,

$g_{31} = (s_1 - s_3)/8$

Figure 2.

The histogram may not exhibit a peak however - for example when there is a steady contrast gradient across the sheet. In such cases, the blob finder is 'warned' not to rely on global measures of contrast. The gradient is always recorded, even if it does not exceed gradthresh. (This is to enable a check to be made on the initial threshold estimate, which can be measured from a few responses of the trolley. In the case of

disagreement, the trolley pusher is simply re-run.) A tiny fragment of the output produced by this stage of processing is shown in figure 3. Since even the reduced picture occupies about 25K words of store, the trolley-pushing routine is written in BCPL to maximise efficiency.

The next stage of processing is more complicated and is written in EVIL (Bornat and Wielinga 1976). This in fact is a fairly typical application of EVIL, which was designed to have a Bobrow-Wegbreit spaghetti-stack (Bobrow and Wegbreit 1973) and be an interactive record-processing system, which, nevertheless, enabled fairly routine processing of huge amounts of data to be carried out highly efficiently.

```
419 ØØØØØØ111ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ 3
420 ØØØØØØØ1111111ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ 7
421 ØØØØØØØØØØØ1111111ØØØØØØØØØØØØØØØØØØØØØØØØØØ 7
422 ØØØØØØØØØØØØØØ1111111ØØØØØØØØØØØØØØØØØØØØØØ 7
423 ØØØØØØØØØØØØØØØØØØØ1111111ØØØØØØØØØØØØØØØØ 7
424 ØØØØØØØØØØØØØØØØØØØØØØØ1111111ØØØØØØØØØØØØ 7
425 ØØØØØØ1ØØØØØØØØØØØØØØØØØØØØØ1111111ØØØØØ 8
426 ØØØØ1ØØØ1Ø1Ø1ØØØØØØØØØØØØØØØØØØØ1111ØØØØ 8
427 ØØØØ1ØØØ1Ø1Ø11Ø1ØØ1ØØØØØØØØØØØØØØØØØØØØ 7
428 ØØØØØØØØØØØØØ1Ø1111Ø11ØØØØØØØØØØØØØØØØØØ 7
429 ØØØØØØØØØØØØØØØØØØØ1111ØØØØØØØØØØØØØØØØ 4
430 ØØØØØØØØØØØØØØØØØØØØØ111Ø1ØØØØØØØØØØØØØ 4
431 ØØØØØØØØØØØØØØØØØØØØØØØ1Ø11ØØØØØØØØØØØØ 3
432 ØØØ1ØØØØØØØØØØØØØØØØØØØØØØ1Ø11Ø11ØØØØØØ 4
433 ØØØ1ØØØØØØØØØØØØØØØØØØØØØØØ1ØØ1Ø1ØØØØ 4
434 ØØØØØØØØØØØØØØØØØØØØØØ1ØØØØØØØØØØ1ØØØØ 2
```

Figure 3.

The next stage of processing is to convert the data in figure 3 into a list of coding sheet lines - whose number can be predicted, as can the fact that they are long. Quantisation effects mean that what is accepted as a coding sheet line appears (figure 3 again)
in the data as a list of segments (runs of 1's with occasional 'small' holes). The segment lists are similar to Clowes and O'Gorman's (1973) 'putative line lists". Details are given in a technical report available from the authors.

Since the number k of lines in a coding sheet is known, the longest k lines (if there are that many) are hypothesised as the coding sheet lines. Each line "ideally" has an equation $y = mx + c$. Since the lines are parallel, we expect that every horizontal line will have the same slope m_h to within an allowable error; failure here indicates that one of the 'real' lines has been missed and a spurious one picked up. Suppose the horizontal lines have equations $y = m_h x + c_i$, $1 \leq i \leq k$. The lines can be assumed to be sorted on the constant c_i, then

because the horizontal lines are known to be equal distances apart it may be expected that the differences $|c_i - c_{i-1}|$ are nearly constant. (The spacing of the first three lines on our sheet is different to the rest: our program takes note of this.) If there is too large a discrepancy in the differences, a line is missing and, so long as a sufficient number have been found, its equation can be predicted and it can be searched for in the rejected linelists. This can happen for example if a segment is missed in the middle of a long line. In practice however, this rarely occurs, since the right side of a coding sheet is usually empty, so that even if some segments are missed the longest lines are the right ones. Even on an artificial example of a sheet covered with writing, all the horizontal lines were found.

It is possible to gauge a rough estimate of m_h and c_i for the ith line from the end points; but this places far too great a reliance on the first and last segments in the line-list, which are in fact likely to be most suspect because of interactions with the vertical lines. Horn (Binford and Horn 1971) discusses similar problems caused by vertices of polyhedra. Instead, we allow every segment to contribute to the m_h and c_i by making a least squares fit to the line.

Similar considerations apply to the vertical lines, except that they are not equidistant. Their spacing is predictable, though - see figure 4. They are however usually easier to find because they do not have writing crossing them. There is also an additional consistency check because m_v and m_h are perpendicular to each other.

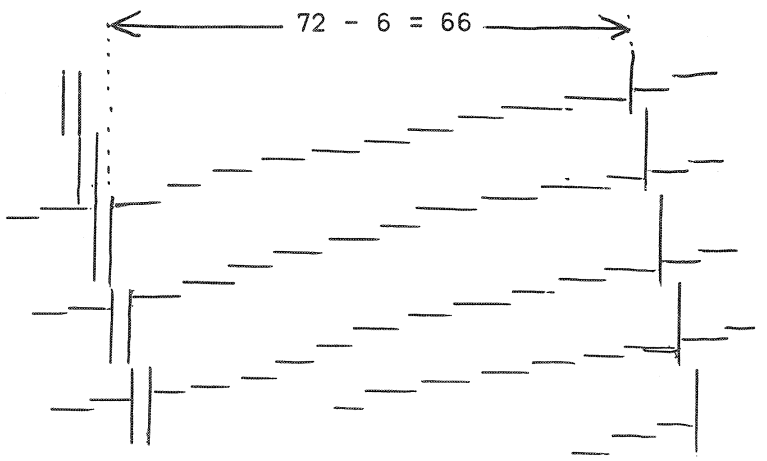


Figure 4.

FINDING THE GRID

The spacing of the vertical lines, positioned at columns 1, 5, 7, 72 and 81, means that we know where the character grid is without looking - coding sheets are printed accurately enough. At the shrunken resolution we can still just tell where the 'blips' on the line are by looking at the data, but at present we prefer to use dead reckoning using the horizontal line-lists to give us 'y' displacements to correspond to the 'x' displacements calculated from the positions of the vertical lines.

FINDING THE BLOBS

The inter-blip gap is more important than the precise location of the character grid. Handprinting uses the size of the gap as a guide, but often doesn't align at all with the grid:

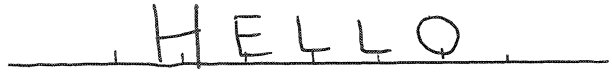


Figure 5.

To identify the number of characters in a blob all we need to know is the size in blip-gaps. Only in column 1 (comment) and column 6 (continuation) is it critically important that we detect exactly where a blob starts or ends.

Far more important is the length and height of a blob. At the time of writing, work on this section of our program is just commencing. We are using histogram techniques to distinguish between 'white' (blank) and 'black' (line) spaces. An enclosing rectangle then gives us length and height. Many writers vary the height of letters, but investigations in our data so far show that operator symbols are usually shorter than a blip-gap and smaller than the surrounding characters. Thus we hope to be able to dinstinguish between 'blobs' and 'operators', which at present seems sufficient distinction to satisfy the Fortran reasoner (Bornat 1976).

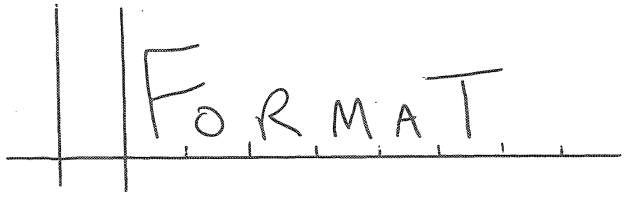


Figure 6.

WHAT IS WRONG WITH OUR PROGRAM

By now the reader will no doubt have noticed the conflict

between our expressed belief in 'heterarchical' program organisation, expressed in the introduction, and the actual reality which is a program organised as a linear sequence of 'stages'. We are dissatisfied with this situation but at present we don't know what to do about it - however the existence of this program in its present form enables us at least to say that our project operates with real input, not just a simulated idealisation. We feel that other sections of the project are more intrinsically interesting and will more effectively repay our efforts so that we may never find it useful to make the changes outlined below. Nevertheless it is interesting, we feel, to discuss the problem of process organisation as we see it at present.

The difficulty is externally obvious in terms of the behaviour of the program. It accumulates evidence uncritically, filters it mechanically and only then uses information about spacing and parallelism as a final check. The problem would still exist if we used a 'better' uncritical line-finder or a 'better' gradient operator (Bob Wielinga built a version of the program which used a line-finder like that in (Clowes and O'Gorman (1973) for example) - we would still be able to characterise the operation of the program in the same way. By contrast, the behaviour we would wish to produce would be one in which the program discovered cues in a section of the input which enabled it to say how the rest of the sheet should be laid out: finding a piece of a horizontal line should enable it to say where the rest of the line is: predicting and searching for the corners of the coding sheet or the end-points of the lines is a good way of anchoring the coding-sheet grid, and so on.

Another approach is to regard the horizontal lines as a texture of black (writing) and white (blank) bars. The texture is strongly periodic. Finding some blobs would enable us to predict the lines, rather than the other way about. The reader may like to introspect on the strength of cues available on lined pages of handwriting where there is (i) the whole page covered with writing (ii) the same but the page has a margin (iii) only alternate lines contain writing (iv) only the margin contains writing. Instead of relying on finding lines we should be swayed by dominant visual cues. (Freuder 1975) and Sloman (1975) seems to be steps in this direction.

The problem is one of the use of knowledge rather than its possession. Our program 'knows' in some sense, a great deal about the coding sheet. For example - the design of the long, thinggradient operator embodies the knowledge that we are looking for along straight lines which are fairly precisely horizontally and vertically aligned; we look for sharp contrasts, using knowledge that the lines are printed; at one stage or another we use the fact that the lines are long, parallel and evenly spaced. The items of knowledge in themselves do not produce

the desired behaviour. The problem of course is how and when does such knowledge intervene in the processing of visual information?

The work of David Marr (Marr 1974a, 1974b, 1975) suggests that our intuitions about heterarchy are leading us astray. Although we are convinced by Marr's argument that re-direction of the primal input processes (bar and edge detection) by 'high-level' knowledge is computationally unrealistic we are unconvinced that subsequent investigation of this information can be independent of knowledge of the scene being viewed. Thus we could accept that our gradient operator (or any other one) is analogous to those which build Marr's 'primal sketch' but we would still want to insist that processing these primitive cues is a task in which knowledge of the particular scene should constructively intervene; further it should intervene in a manner which would not preclude other knowledge, of different scenes, to intervene if we were to expand the system's 'possible worlds'. Thus we would want, in theory, to be able to change our program so that it could view the sheet in perspective, or view a sheet printed so that the lines were in a 'fan' shape. Unfortunately we have to admit that programming this 'modular' intervention of knowledge in mid-level visual processing is beyond the state of the art at present. Until it can be achieved, or until it can be convincingly shown to be unnecessary, it must remain unresolved how far knowledge can direct this kind of processing.

Reasoning about Fortran and handwriting seems to be a more tractable problem as well as a better way of investigating just how knowledge can be used in the way suggested above. It is reasonable, for example, that the character experts might as well interface with an Algol or LISP reasoner as with Fortran. To a large extent the entire project is exploring the control structures necessary to produce this kind of knowledge interaction; thus we have contented ourselves, for the moment, with our program as an "existence proof" that the blobs can be found so that other processes don't have to work with simulated environment.

REFERENCES

Binford, T.O. and B.K.P. Horn (1971) The Binford-Horn Line finder. MIT AI lab working paper 16.

Bobrow, D.G. and B. Wegbreit (1973) A model and stack implementation of multiple environments. Comm ACM, 16 October.

Bornat, R. (1976) Reasoning about hand-blocked Fortran programs. Submitted to AISB Summer conf.

Bornat, R. and B.J. Wielinga (1976) Live was I ere I saw EVIL. University of Essex, dept. memo.

Bornat, R. and J.M. Brady (1976) Using knowledge in the computer interpretation of Fortran coding sheets. Int. J. Man-Machine Studies.

Brady, J.M. and B.J. Wielinga (1976a) Seeing a pattern as a character. Submitted to proc. AISB conference.

Brady, J.M. and B.J. Wielinga (1976b) Seeing a pattern as a character. In preparation.

Clowes, M.B. (1971) On seeing things. Artificial Intelligence 2, 79-116.

Clowes, M.B. (1973) Draft notes on vision (privately circulated). University of Sussex.

Freuder, E. (1975) Ph.D. thesis. MIT AI lab.

Marr, D. (1974a) The purpose of low-level vision. MIT AI memo 324.

Marr, D. (1974b) The low-level symbolic representation of intensity changes in an image. MIT AI memo 325.

Marr, D. (1975) Analysing natural images: a computational theory of texture vision. MIT AI memo 334.

Minsky, M. and S. Papert (1972) Progress Report. MIT AI memo 252.

O'Gorman, F. and M.B. Clowes (1973) Finding edges through collinearity of feature points. Proc. IJCAI 3, p. 543-555, Stanford.

Sloman, A. (1975) Proposal to SRC.

Weir, S. (1975) Symbolic description and the constructive nature of perception. Final report to SSRC to grant HR2071/2, Edinburgh.

Weir, S. (1976) Paper submitted to AISB Summer Conference, Edinburgh.

Winston, P.H.(1975) The psychology of computer vision. McGraw-Hill.

R. Bornat
B.J. Wielinga

DOES AI PROGRAMMING REALLY HAVE TO BE LIKE KNITTING WITH SPAGHETTI?

AI programming is sophisticated - often exploring the edge of what is known to be possible - and the hope of the AI programming language designer is to provide a language in which any kind of computational process behaviour may be described, with an implementation in which execution of that process is as efficient as may be. AI programs attempt to model intelligent processes in terms of the behaviour of complex computational processes and it is not desirable to restrict future advance by prohibiting certain kinds of behaviours; at the same time, as research progresses, it is necessary to provide clearer representations of behaviours which have been proven effective.

AI programming languages preserve flexibility by providing a collection of primitives which are formally adequate to describe a large range of control behaviours* (e.g. early LISP's conditional expression, function call and return, FUNARG). Formal adequacy isn't enough, though, or we'd all be programming Turing Machines. The primitives provided are biased, then, towards the description of certain kinds of behaviour and as Bobrow and Wegbreit (1973) have shown, this permits efficient execution of these behaviours. Other behaviours may be 'hacked up' in terms of the primitives provided but, in the limit, there always seem to be situations in which the primitives provided hinder as much as help.

In this paper we investigate the difficult synthesis between generality, efficiency and linguistic power and we critically examine some of the tools proposed for the implementation of AI programming languages. In particular we discuss the work of Bobrow and Wegbreit because of their contribution to the efficiency of execution of AI programs. We do not propose a new set of basic primitives or an adequate collection of control behaviours - we believe that it isn't possible to do so. We aim to point some directions for AI programming language development, though, so that we can write our programs more naturally and run them more efficiently as well.

The research which gave rise to this paper originated partly from the Fortran-reading project at Essex University (Bornat & Brady 1976a, 1976b; Brady and Wielinga 1976; Bornat

---

* Not any behaviour, but usually those which the designers know and approve.

1976) in course of which we implemented a system called EVIL* (Bornat & Wielinga, 1976). EVIL is based on the Bobrow and Wegbreit 'spaghetti-stack' model and provides such features as lists and record-structures similar to LISP and POP-2, linguistic constructs based on BCPL. All of this embedded in a highly interactive environment (interpreter, in-core editor, interactive debugging package). In the course of implementing EVIL and during subsequent experiments with implementation of a CONNIVER-type database and a FRAME-system, we discovered some deficiencies of the linguistic schemas we originally chose for the language and of the spaghetti-stack mechanisms. We found it hard to extend beyond the capabilities of LISP and POP-2: this paper explains why.

CONTROL BEHAVIOURS, EFFICIENCY AND STRUCTURED PROGRAMMING

A system can only implement efficiently those behaviours which it can recognise. The spaghetti stack model of Bobrow and Wegbreit (1973) provided efficiency by means of a reference count mechanism which was capable of recognising simple recursive behaviour. More complex behaviours were provided via label values (ENVIRON), access environment manipulation (ENVIRON again) and GOTOs (ENVEVAL using a saved label value). The user could help the system a bit by explicitly destroying label values, signalling the use of multiple simply-recursive processes or backtracking. Bobrow and Wegbreit proposed their implementation as a general model of process behaviour and contended that it should handle other languages as well as it does LISP. It isn't clear that this is so: while some features of other languages might easily be modelled, others present real difficulties.

'Access closures' (modelled by FUNARG in LISP) are the only form of closure mentioned by Bobrow and Wegbreig. 'Control closures', a closure of a function and a control environment** (example: the result of applying Landin's 'j' operator), are ommitted from LISP but it is easy to see how they might be modelled. 'Partial application' a la POP-2 seems to be outside the scope of the model. So are 'process closures' (generator functions, Landin's streams, SIMULA 67 classes etc.), which are not adequately modelled by label values. There are other

---

* The gullible may like to believe that this is an acronym for Essex Vision Language; it is really an excruciating pun on EVAL.

** 'Access closures' may easily be modelled as a pair: function code, name-value mapping . A 'control closure' is, by analogy, a pair: function code, control environment . A 'control environment' specifies what should happen at function exit - in many implementations, just a 'return link'.

examples: actor-like behaviour in particular seems very difficult to model.

The point is not that these features can't be 'hacked-up' out of the spaghetti-stack primitives; they can be, though with difficulty, (just as they can in FORTRAN!). These behaviours, though, ought to be low-level primitives of the machine and ought to be directly describably in the language for two reasons: first we can program directly in terms of the same control structures we are using conceptually, second we make our intentions clear to the machine, so that it can recognise the behaviour and implement it efficiently. Clarity is esential to avoid bugs: for instance the coroutining example in Bobrow and Wegbreit (1973) doesn't release every stack frame as soon as possible and so is unnecessarily inefficient (see (Bornat & Weilinga 1976) for details). We learnt the lesson, too, when using the spaghetti-stack primitives in EVIL - it is hard to keep track of the allocation and de-allocation of stack frames when the behaviour is anything other than simply recursive.

PSDs - A Reformulated Model

Despite our belief that no model of programming languages can adequately account for all conceivable behaviours, use of a (locally adequate) model can help to make discussion of behaviours more precise. If we rewite the Bobrow-Wegbreit model, slightly, removing the implementation details, we arrive at a slightly more general model which can account for some (though not all) of the behaviour we have mentioned. A Process State Descriptor (a PSD) is like a spaghetti-stack 'frame'. It contains five sections:

A: A description of the current activity of the process (local storage, a record of interpreter stages, etc)*
B: A description of processing yet to be done
C: Aname-value mapping for variables local to the process (declared in the program section which control it)*
D: A name-value mapping for other variables accessible by the process
E: A description of what to do when section B is exhausted

Using this model, we are able to define EVIL implementations of some interesting programming values. A functional value is a PSD with only section B filled; an access closure a PSD with sections B and D; a control closure a PSD with sections B and E; a partially-applied function has section B and some of section C filled in. A label-value is a PSD with every section filled in, and so on.

---

* These bracketed remarks are merely examples, of how these sections are filled in many implementations.

This model allows us to model some behaviours more elegantly then, and it also exposes some design choices. Our model isn't the end of the line, though, as we shall see, and we emphasise that it isn't a viable model for all the program behaviours we want to implement. We introduce it because it strips away the implementation details of the spaghetti-stack model (we see spaghetti stacks as an implementation of PSDs) and so enables us to discuss the capabilities and drawbacks of modelling behaviour in terms of recursive functions. In particular, although most implemented languages can be easily accommodated in the PSD model, it is clear that the number of sections in a PSD, the information to be kept in them and the ways PSD's are linked together are all essentially arbitrary choices. Thus the PSD model is an aid to an expansion of possibilities as well as a delineation of existing reality (which isn't surprising if we reveal that it started life as an attempt to explain 'actors'!).

PROCESS CLOSURES: COROUTINES AND SEMI-COROUTINES

Landin's streams (Landin 1965), SIMULA-67's classes (Dahl and Hoare 1972) and CONNIVER's generators (Sussman and McDermott 1972) are examples of programming constructs in which control is not constrained to function call and return. Milner's 'process' (Milner 1973) and, to a lesser extent, Hewitt's 'actors' (Hewitt 1973, 1974) provide modern theoretical back ground, allowing us to emphasise the behavioural view of programming and de-emphasise the functional. More and more, AI programs are coming to look like multiple co-operating processes and it is time we developed implementations and languages using these ideas which are as tidy and as structured as those we already have for hierarchical behaviours.

The simplest implementation of coroutines is via global label values, as in GEDANKEN (Reynolds 1970) or in Bobrow and Wegbreit (1973). This isn't adequate for semi-coroutines (Dahl and Hoare 1972), for streams or for generators, in which the activation of a co-process is like a function call and the de-activation is like a return, except that the next activation starts from where the last one left off.

As an example of co-processing, consider CONNIVER's TRYNEXT applied to a possibilities list whose first member is an IF-NEEDED method. We might regard this element as a co-process descriptor: TRYNEXT activates the co-process. At some point it reaches an AUREVOIR: now the co-process returns a value to TRY-NEXT (actually it splices it into the possibilities list, but draw a veil over that!) and the descriptor is updated so that the next time it is used the co-process will continue from that point after the AUREVOIR. SIMULA's CALL and DETACH have much the same effect.

This notion is very clear, and to rescue it from the innards of TRYNEXT, raising it to its proper status as a process structuring aid which is as important as hierarchical function calls, would be very rewarding. The problem is the scope of the control environment - each AUREVOIR or DETACH must know which co-process it is stopping and which it is starting. Normal function returns use the CLINK, or section E of the PSD, for tne corresponding purpose. To get a bug-free, manageable implementation we have to extend our virtual machine to give the same clarity to co-process communication. Implementations in the PSD model or the spaghetti-stack model, as we know to our cost, have to search the control chain for the value of a 'secret' variable which identifies the co-process descriptor. This is only reasonable if you are implementing TRYNEXT-AUREVOIR as a minor hack - it won't do in a proper language.

Clearly, for the system to be able to recognise what is going on so that it can implement it properly, each PSD should indicate what co-process it belongs to. An indication might perhaps be (a reference to) a co-process descriptor of some kind, updated on every DETACH to contain a label value representing the current state of the co-process. A difficulty here is that we can't see how label values mesh with this: normally a GOTO restores an old control environment, but co-process descriptors would be values in the stack and thus difficult to reset.

Another feature of SIMULA 67 is the ability to access variables of a dormant co-process - so we need some way to describe the way we handle the access environment of a co-process. Up to now AI languages haven't included this kind of inter-process communication, so it isn't clear what is needed or whether what SIMULA 67 provides will do.

The fundamental point is: if our programs are to use co-processing as a major control structuring mechanism, then our languages and our mechanisms should provide it as a primitive. If we hack it up out of function calls we get inefficiency of execution, opacity in our programs, bugs in programs and in implementations and general confusion - besides losing sight of the methodological fact that co-processing is a process structuring device we can build on, not merely a clever trick.

CONCLUSION

We have tried to show that some very well-structured and straightforward control behaviours are difficult to describe in terms of the models of program behaviour currently available. As a result AI programs are harder to write (the damn systems won't let us write what we mean!) and we are doubly discouraged because when we do hack up what we want it runs too damn slow. The introduction of the PSD model allowed us to account for

some control constructs fairly elegantly (and hence efficiently but failed to account fully for such a well-known behaviour as SIMULA 67 classes.

Currently we are working on an extension of the EVIL implementation and of the PSD model, which we hope will enable us to handle co-processing properly. Already, though, we know of some behaviours which our new model won't be able to handle appropriately - for example event-driven processes, dynamic lists, actors. Although we are aware that we could never reach the ultimate AI programming language, it is important that everybody keeps trying. The pioneering work of Dijstra (1972) has shown that if you can Say What You Mean then you'll probably Get It Right - a big boost to programmer productivity and that of Bobrow and Wegbreit has shown that if the system knows What You Want it can Do It For You Fast (and Get It Right!) - which frees us all from worrying about the efficiency of our basic program designs. The whole effort of developing AI programming languages is one of bringing new structuring ideas into the reach of every programmer and thus expanding everybody's range of possibilities.

Although we may appear to have stressed the question of efficiency, what is at issue here is a representational point. If we are to represent our theories in terms of processes we must be able to program in terms of processes. Restricting ourselves to function calls is just that: restrictive! Perhaps talking about AI programming 'languages' is misleading - we have been talking about drawing out and clarifying ways of thinking about processes. The structuring aids we use must help us to think about our programs in a variety of ways.

REFERENCES

Bobrow, D.G. & Wegbreit B. (1973) A Model, and Stack Implementation, of Multiple Environments. Comm. ACM 16, 10.

Bornat, R & Brady J.M. (1976a) Using Knowledge of the Computer interpretation of coding sheets. (to appear in IJMMS, 1976)

Bornat R. & Brady J.M. (1976b) Finding Blobs of writing in the FORTRAN coding sheets project (submitted to AISB conference)

Brady, J.M. & Weilinga B.J. (1976) Seeing a Pattern as a character (submitted to AISB conference).

Bornat, R. (1976) Reasoning about Hand Printed FORTRAN programs (submitted to AISB conference).

Bornat, R. & Weilinga B.J. (1976) The EVIL programming system University of Essex, Computing Centre memo (in preparation).

Landin, P.J. (1965) A correspondence between Algol 60 and Church's Lamda-notation. Comm. ACM 8.

Dahl & Hoare (1972) Hierarchical Data Structures, in 'Structured Programming', Academic Press.

Sussman, G.J. & McDermott, D.V. The CONNIVER reference manual, MIT-AI memo 259.

Hewitt, C. et al. (1973) A Universal Modular ACTOR formalism for AI, proc. IJCAI 3.

Hewitt, C et al. (1974) Behavioural Semantics of ACTOR systems, MIT working paper.

Milner, R. (1973) An Approach to the Semantics of Parallel Program, Proc. Convegno de Informatics Teorica, PISA.

Reynolds, J.C. (1970) GEDANKEN - A simple typeless language Comm. ACM 13, 5.

Dijkstra E.W. (1972) Notes on Structured Programming, in Structured Programming, Academic Press.

J.M. Brady
B.J. Wielinga

SEEING A PATTERN AS A CHARACTER

The work described in this paper is part of a project to develop a program capable of reading a handwritten Fortran coding sheet (see Bornat and Brady, 1976a). The emphasis of the project is on the organisation of processes and knowledge necessary to facilitate the interpretation of the massive amount of digitised input data as a Fortran program on a coding sheet. A companion paper (Bornat and Brady, 1976b) describes a preprocessor capable of isolating the coding sheet grid and the length and position of the "blobs" of writing. Bornat (1976) discusses a program based on knowledge of Fortran which can construct a tentative interpretation of the blob data as a Fortran program. Although this turns out to be possible to an extent beyond expectations, it is still necessary to investigate in some detail (the regions corresponding to) the individual characters on the sheet. This paper describes a program which is designed to deploy a lot of knowledge about characters and the way they are written, to read handprinting, and which is intended to interact with the Fortran reasoner.

Unlike most previous work in optical character recognition, and in pattern recognition, we do not consider the task of our program to be solely to classify an input. For example, the Fortran reasoning processes may hypothesise that a particular blob is "FORMAT', and hence that a particular character is 'F', and ask our program for verification. Indeed, we view character reading as a perceptual task not essentially different from other picture perception tasks. In particular, the arguments presented in (Bornat and Brady, 1976a) apply in microcosm to the character reading portion of the project. The effect of working within the context of Fortran is in the extra knowledge which can be brought to bear in trying to see a pattern as a character. Waltz's (1975) work suggests that extra knowledge, if used appropriately, can make interpretation more straightforward since there are more consistencies to exploit.

The remainder of the paper discusses some of the ideas which have affected the evolution of our present program, example behaviours of which are sketched in sections 4 and 5. In (Brady and Wielinga, 1976), we present a fuller account of a number of issues, such as segmentation, low level processes, and our representation of characters as 'frames', which for reasons of space, we here touch on lightly or not at all.

2. CHARACTER DESCRIPTIONS

Any character interpretation system embodies, albeit usually in a highly inflexible form, some knowledge of characters, and at least a description of each character model in the alphabet. The elements from which such descriptions are constructed must be computationally discoverable; practical pattern recognition systems generally also insist that the description elements should be cheaply discoverable. Most conventional character recognition systems are based essentially on feature extraction (Balm, 1969). Characters are described in terms of their topological features, such as: number of line endings, junctions, cusps and curvature, concavities etc. Shillman describes an extensive set of features ('phenomenological attributes') based on a study of ambiguous characters, rather than concentrating on a classical archetype or model (Shillman, 1974, Blesser et. al., 1973). Although this technique has been fairly successful in the recognition of machine printed charaters, it seems to break down for casually hand-printed letters. Furthermore, like most other suggestions, it would probably fail completely in the presence of ligatures or in situations where segmentation is difficult and gives rise to spurious "evidence". Our program has to deal with both of these problems.

Our approach has been to forget cheap processing, and use descriptions which we find intuitively satisfying, and which seemingly apply even for casual printing. Thus our descriptions of characters speak of strokes and curves, and their positional and relational properties. The presence of strokes and stroke relations (like intersections and junctions) often provides strong cues for characters. When trying to verify a particular character we expect to find the prominent strokes but not always more detailed features. Particularly in the presence of ligatures or when segmentation is difficult, features such as the number of line endings or curvature might provide misleading cues. Although this descriptive mechanism is rather simple, it suffices for the vast majority of our needs, particularly given the context of Fortran. Moreover, it can be extended to include general knowledge about the way strokes are written, and the distortions which commonly occur, for example when writing quickly. Thus we believe that we can tentatively put together a description of the process by which a character was produced. This seems to be a powerful way to get to grips with ligatures and ambiguous characters (figure 1)



figure 1 (TTNISH or FINISH?)

Finding strokes is essentially the same process as line finding which has been considered in several vision projects. The technique we use is a mixture of "trolley pushing" and the Clowes-O'Gorman (1973) description of a stroke in terms of its (set of) constant (quantised) gradient direction(s) at points where the gradient intensity exceeds a threshold. A particular set of quantised gradient directions is selected either by analysing a histogram taken over the whole region, or on the basis of contextual hints about the character's identity. A stroke is defined to be a region consisting of all points having a particular small set of gradient directions (and with certain proximity relations fulfilled). Curves are described as a piecewise-linear sequence of (straight) strokes with a changing direction between successive strokes (generalising chain coding, Freeman 1961). In this way we can derive an estimate of the curvature of a particular curve; this can be useful for example in distinguishing between 'C' and '('.

There are a number of problems with the stroke/curve finding process. Strokes which are intersected tend to be found as two small strokes rather than as a single big one. Curved flourishes at the end of strokes are generally not identified as part of the stroke, and are usually missed altogether. The system's preference for straight strokes means that shallow curves are often picked up as two individual straight strokes. These problems are compounded by the fact that for reasons outlined in (Bornat and Brady, 1976b) the spatial resolution of our characters is about 25 x 25 pixels, which is considerably smaller than most conventional character recognition systems use.

Instead of trying to improve the performance of the feature finding process, we rely on the flexible process structure of the program, which allows us to refine the data when this appears necessary. Several operations are provided for this purpose. Strokes can be joined, e.g. when their direction is similar and they are collinear. Strokes can be extended in a chosen direction. Junctions can be verified or searched for. We can also lower various thresholds to look for tiny strokes like the crossbar of an 'A'. All these operations can be triggered by high-level hypotheses or by specific knowledge in demon-like form.

3. WORKING WITH PARTIAL KNOWLEDGE

We have designed our program to adapt its behaviour to provide the various kinds of information which are asked for by the other parts of the system, and to take advantage of any advice which they can offer. Most character reading systems behave only as recognisers: in terms of the way in which a vision program uses knowledge to see, we can define recognition as the problem of interpreting an input given minimal knowledge, for

example, merely knowing that it is a character. In section 1, we proposed that our program should at least additionally be capable of verifying the hypothesis 'the character is F'; in terms of the use of knowledge, verification if the task of interpreting an input given total knowledge, that is, we know what character to expect.

The interpretation of an input, and the deploying of know ledge, develops gradually over a number of interactions between our program and the Fortran reasoner. Thus the program works in a context of partial knowledge, minimal and total being just two extreme cases. The Fortran reasoner may advise our program that: 'This character is probably I or L', 'I expect an arithmetical operator', 'this character is likely to be the same as the third one on the previous line', and so on. Alternatively, it may not demand exhaustive recognition, but might ask 'is there any evidence for F?', 'are there any brackets?'. Similarly, our program may respond 'I can confirm A, but I prefer H', 'I deny A, but suggest H, U or V', 'there is some evidence for E, but troublesome evidence also'. It might ask of Fortran, 'is E possible here?', 'which is more likely, M or N?'.

An important consequence of this rich interaction is that the control structure of our program must be very flexible, and must adapt to take advantage of what information is wanted and what is available. In the next section we describe why we abandoned a prototype written in a Conniver-like language. Apart from the problem of control structure, there is the problem of the confidence level of the information provided by other parts of the system. For example, when the Fortran reasoner asks us to distinguish between say A and C, our program may try to find a sensitive test to discriminate between these characters (e.g. by looking for a diagonal stroke in the right hand side of the picture), but such tests fail completely when the original hypothesis is faulty. So, the character reading program must, to a certain extent, constantly be critical of any information provided to it. A likely source of this kind of trouble is the segmenter, and our program should be at least aware of possible problems of this type. We return to a brief discussion of these issues later.

4. OBITUARY FOR A CONNIVER-BASED PROGRAM

In an earlier stage of the development of the character reading program, we built an (unpublished) program in POP2, using the POPCORN package developed by Steven Hardy (1976) for his thesis. POPCORN essentially provides the same primitives as CONNIVER (McDermott and Sussman, 1972/1974). The basic strategy of the program was to wait until all 'cheap' cues (such as strokes, stroke relations) had been found, before deciding what to do. The program used IFADDED and IFEVERADDED (Hardy, 1976) (CONNIVER'S HANG) methods to note strong evidence for particular

characters, and sprung appropriate traps when all cheap cues had been discovered, to hypothesise the most strongly supported character(s). Thus finding three horizontal strokes suggests 'E', and causes the downstroke to be hypothesised and searched for even if it is not cheaply available. Similarly, a junction can by hypothesised and verified on the basis of the strokes alleged to comprise it, or on the basis that the character identity is known. It can be seen that the program alternately works in recognition mode, in which it assumes nothing is known and verification mode, in which it thinks it knows everything. This early disloyalty to the methodology of partial knowledge proclaimed in the previous section is just one of a number of problems we encountered in the course of developing the POPCORN program, and which we believe are intrinsic in CONNIVER-like programs.

Firstly, the primitives IFADDED and IFNEEDED 'naturally' suggest data driven and goal driven control structures. We found it difficult to express with the Conniver primitives a control structure which was sufficiently flexible to make efficient use of partial evidence available. Of course it is possible in theory to circumvent this problem, but we agree with McDermott and Sussman (1972, page 1) that the primitive control structures provided in a programming language are closely inter-related with the problem solutions which a programmer effectively considers.

The second problem concerns the (lack of) structure of the CONNIVER data base, and its implications for the structuring of control. By way of illustration, consider a character model as a typical package of related facts and procedural knowledge. Such models must be available in situations where partial information is being considered. If the character program is trying to distinguish between two characters (in response to Fortran's 'is it I or L?'), or between a confirmed instance of a character and its model (to derive a notion of style such as sloping forwards, which might aid later reading), it must be able to structure its knowledge appropriately. However, the CONNIVER database is inherently flat, so that it is rather difficult (even using contexts) to impose structure on available information. In addition, it is difficult to make the program 'aware' of its own behaviour. For example if knowledge is represented procedurally, it is usually nontrivial to discover what evidence is available in support of a particular character, what contrary evidence has been found, and what has not yet been confirmed. Typically, demons are triggered as evidence is added to the database, but they leave only indirect evidence of their effect in the form of new assertions. However, the implications of a piece of evidence can usually only be assessed in the light of what has already been discovered. Figure 2 shows a pathologically written F, where the stroke

normally assigned to the role of lower horizontal stroke is a very strong contender for the role of topstroke. In such cases we require our program to contemplate the structure it has built, and compare it with an F-model. It should notice the unassigned lower stroke and accept the lower candidate for the role of topstroke as a reasonable filler on the basis of a common, and known, distortion. This kind of behaviour is difficult to achieve in CONNIVER, where character models are not very explicit and the program is relatively unaware of the reasons supporting a particular interpretation.



figure 2



figure 3

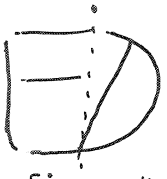


figure 4

Figure 3 illustrates a similar problem, which occurs when contradictory evidence is discovered. In a data-driven, demon-based system, it is all too easy to ignore contradictory evidence. Thus, proceeding in a purely topdown manner, any 'E' can be confirmed as an 'F'. We have designed our program to account for all the data, even if it is contradictory. In figure 3, the 'F' model should realise that the bottom stroke is strong evidence against 'F', and indeed that it very strongly suggests 'E'. Figure 4 is intended to show that while it very strongly suggests 'E' rather than 'F', it is not totally convincing.

The examples shown in figures 2 and 3 are within our current program's capabilities. Figure 5 shows how such problems often result from 'wrong' segmentations. Suppose that the program would happily accept the first region as 'A', but would get into trouble with the second region. L is most plausible, since, for example, I would be expected to have produced a vertical stroke about the centre of the horizontal. Since the horizontal stroke extends beyond the left edge of region 2, a segmentation problem may be suspected, and a vertical stroke to complete the 'L' hypothesised to the right of region 1. This is discovered and region 1 reinterpreted as 'F'. We have not tried our program on this example, because of our difficulties in producing data, but we believe that our current program can straight-forwardly be extended to bring it within range.

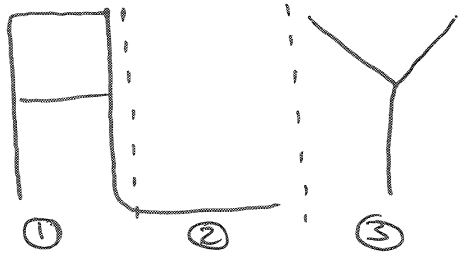


figure 5

5. AN IMPLEMENTATION OF FRAMES AND A SKETCH OF THE PROGRAM

To overcome the problems discussed in the previous section, we have based our current program on an implementation of Minsky's (1975) frames. In a wide-ranging discussion of various aspects of intelligence Minsky reminded A.I. workers that knowledge, as well as control, needs to be carefully, and richly, structured, and he proposed structures called frames, consisting of related facts and procedures, as a representation of knowledge. If Minsky's discussion was intuitive, it was also sufficiently attractive to inspire a number of groups to implement their understanding of the suggestion, thereby making the ideas more precise. Our implementation (in the language EVIL (Bornat and Wielinga, 1976) developed for the project) is based on the discussions of Winograd (1974a, 1975a, 1975b) and Bobrow et. al. (1974), tuned to the needs of our character program.

Frames are characterised by a name, type, slot-descriptions and actions. The purpose of type is to link the frame in an ISA-nierarchy (Winograd 1974, McDermott 1975), i.e. to make certain properties of a class of frames generally available for each instance of the TYPE. For instance, when we are considering a frame of type STROKE we may directly access general properties of STROKE-type frames, such as: how we can find strokes, what are the general attributes of strokes.

Slots basically provide a means to describe a frame or its applicability. Slots are filled either with an invariant description of an important feature of the frame, or contain a template description of the entity which can fill the slot when the frame is instantiated (see below). For example a frame for the letter A may contain three slots providing descriptions of three strokes and several slots to describe the relations between the strokes.

Actions form the procedural part of frames. They typically consist of a precondition and an action specifier. The action specifier describes the action to be performed when the precondition becomes true. Preconditions generally become satisfied when certain operations on the frame get performed, e.g. instantiation, filling a slot, confirmation.

The central operation in the system is frame instantiation. Instantiation of a frame occurs in a context. A context contains frames which have been instantiated in earlier stages of the computation either by hypothesising them or by being created by low-level operations such as finding strokes. When a frame is instantiated, it can be mapped onto the evidence that is already present in the current context. This means that frames in the context are matched against the slot-descriptions of the frame that is being instantiated. During the process of instantiation a frame might 'grab control' in a demon-like way, when the preconditions for an action get

fulfilled. An instantiated frame remains in a context until it is explicitly removed. When a frame is only partially instantiated, any new evidence will be matched against the slots which have not yet been filled. This technique provides a means to create demons which will only trigger on a combination of several preconditions.

These ideas are now illustrated with reference to a simplified frame for the character A (figure 6). The A frame has 5 slots, corresponding to the leftdown, rightdown and horizontal strokes, the intersection of the leftdown and rightdown strokes and the crossbar formed by all three strokes. It also has 3 actions, with,preconditions TO. ACHIEVE, WHENFILLED, and WHENCONFIRMED. The action TO. ACHIEVE is of use when the system operates top-down, for example in response to a request from the Fortran reasoner to verify A. The WHENFILLED action is a demon, which fires when the left and right down strokes of the A are discovered (it is expected that they will be if the character is A), and tries to verify that they form an appropriate junction. If they do, A is confirmed. (This is only one of the ways the WHENCONFIRMED action may be fired; an alternative might be in the course of trying to verify an 'H' which turns out to have a junction at the top). If the junction is not found, the strokes are checked to see if they converge towards the top. If not, A is denied, and H, U or V proposed as more likely. If they do converge, H is preferred to A, and the frame suspends itself by executing the routine LEAVE(). This means that the frame has difficulty in establishing A, but if necessary, could 'try harder', for example by computing the angle between leftdown and rightdown, in an attempt to distinguish the cases displayed in figure 7. This idea is based on Hardy's (1976) 'research director'. Finally, the WHENCONFIRMED action sees if all troublesome evidence can be explained away, perhaps as segmentation problems, and then tries to assimilate A into the current context, by matching the instantiated A frame against hypothesised slots in the context. Besides the instantiation operation there are operations on frames like store, retrieve and matching operations. Frames are indexed in a multi-level associative memory, i.e. they can be retrieved by name, type and by any part of a slot description. Thus we can retrieve for example all character frames with, say, a big vertical stroke and an L-type junction as components.

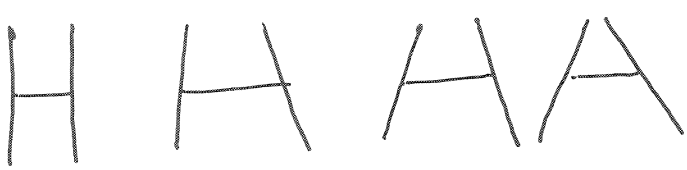


figure 7

```
[A ISA LETTER
   WITH.SLOTS
       CHAR ← A
       RIGHTDOWN  ← [? ISA STROKE
                     WITH.SLOTS
                         SLOPE ← <ANYOF LEFTV VERT>
                         SIZE  ← <ANYOF BIG MEDIUM>/NOINDEX
                                 //Note: NOINDEX means can't
                                 //be searched for under size
                         POS ← RIGHT]
       LEFTDOWN   ← [ ? ISA STROKE
                     WITH.SLOTS
                         SLOPE ← <ANYOF RIGHTV VERT>
                         SIZE  ← <ANYOF BIG MEDIUM>/NOINDEX
                         POS   ← LEFT]
       HORIZ      ← [ ? ISA STROKE
                      WITH.SLOTS
                          SLOPE ← HORIZ
                          SIZE  ← <ANYOF SMALL MEDIUM>
                          POS   ← MIDDLE] /NOINDEX
                    //i.e. we don't expect to find Horiz, but
                      are delighted if we do
       INTLR      ← [ ? ISA INTERSECTION
                     WITH.SLOTS
                         STROKE1 ← !RIGHTDOWN
                         STROKE2 ← !LEFTDOWN
                        RELANGLE ←  ACUTE
                         POS     ← TOP]
       CROSB      ← [ ? ISA CROSSBAR
                      WITH.SLOTS
                          SLEFT ← !LEFTDOWN
                         SRIGHT ← !RIGHTDOWN
                           SBAR ← !HORIZ] /NOINDEX

with.actions
   [TO.ACHIEVE]
            $( achieve (leftdown)
               achieve (rightdown)
            $)]
   [WHEN FILLED   <ALLOF LEFTDOWN RIGHTDOWN>]
          [test verified (INTLR)
           then confirm (A, context)
           or   //the junction can't be verified
                test converge.at.top (leftdown, rightdown)
                then $( prefer (H, context)
                        leave ()
                test checktouch (leftdown, rightdown)
                     then confirm (A,context)
```

```
                    or confirm (H,context)
                       possible (A,context)$)
               or deny (A,context)
                  propose ([H U V], context)]]
    [WHENCONFIRMED
          [handle.troublesome.evidence ()<>
           assimilate (A)]
    ]
```

figure 6. A frame for A

6. APOLOGY

The preceding discussion raises more problems than it solves. We are very well aware that our implementation of frames is very sketchy and that a lot of issues have not been faced. Crucially, we have only hinted at the way our program actually works, although more details are provided in (Brady and Wielinga, 1976). The most glaring omissions in this account are to do with the issue of control. The basic strategy of our current program is to investigate the input data in a bottom-up way until enough knowledge is gathered to proceed in a top-down manner. So, when enough contextual information is available to make at least a few (but not too many) hypotheses, the information in the available character frames will be used to guide the low-level operations. Alternatively, when no information whatsoever is available in an initial stage, the strategy will be first to add a few demon-frames to the context which will watch for junctions, intersections and similar cues, and give control to the low-level stroke finding operations. When one or more strong pieces of evidence become available (like a junction or intersection), character frames will be retrieved from the database. If this retrieval operation results in a sufficiently small number of hypotheses, a top-down approach will be tried.

The implementation of this type of control structure poses many severe problems. In section 3 we stressed that the overall strategy must be flexible and must change drastically according to context or the type of request made. This raises the following question: where should control reside in the system? In the example of the A frame (fig. 6), quite a lot of control was concentrated in the A frame itself, such as the attempt to explain away troublesome information. This type of locally concentrated control is easy to program. From a global viewpoint however, it might be an entirely inappropriate strategy. Certainly, these are the issues which we expect will occupy most of our attention in the near future.

REFERENCES

Balm, G.J. (1970) An Introduction to optical character reader considerations, Pattern Recognition, (1970), 151.

Blesser, B. et al. (1973) A theoretical approach for Character recognition based on phenomenological attributes. Proc. First Joint Conf. on Pattern recognition 1973, p. 33.

Bobrow, D.G. et al. (1974) Steps towards language understanding. Xerox PARC, October 1974.

Bobrow, D.G. & Collins, A.M. Representation and Understanding: Studies in Cognitive Science, Academic Press.

Bobrow, D.G. & Wegbreit, B (1973) A model and stack implementation of multiple environments. Comm. ACM, 16, p. 591.

Bornat, R (1976) Reasoning about hand-blocked Fortran programs submitted to AISB conference 1976.

Bornat, R. & Brady, J.M. (1976a) Using knowledge in the computer interpretation of Fortran coding sheets. Int.J. Man-machine Studies 1976.

Bornat, R & Brady J.M (1976b) Finding blobs of writing in the Fortran coding-sheets project. Submitted to the AISB conf.

Bornat, R & Wielinga B.J. (1976) The EVIL programming system. University of Essex, Dept. memo, to appear.

Brady, J.M. & Wielinga, B.J. (1976) Seeing a pattern as a character, in preparation.

Freeman, H. (1961) On the encoding of arbitrary geometric configurations IEEE Trans Electronic Computers, vol EC-10 p260.

Hardy, S (1976) Synthesis of LISP functions from example. Ph.D thesis, University of Essex.

McDermott, D.V. (1975) Symbol mapping. Sigart Newsletter No. 53

McDermott, D.V. & Sussman, G.J. (1972/1974) The Conniver reference manual, MIT AI-Memo 259/259A.

Minsky, M. (1975) A framework for representing knowledge, in Winston (1975).

O'Gorman, F. & Clowes, M.B. (1973) Finding edges through collinearity of feature points. Proc. IJCAI 3 p. 543

Shillman, R.J. (1974) Character recognition based on phenomenological attributes: theory and methods, Ph.D thesis, MIT.

Sussman, G.J. & McDermott, D.V. (1972) Why Conniving is better than Planning. MIT AI-memo 255A.

Waltz, D. (1972) Generating semantic descriptions from drawings of scenes with shadows, Ph.D. thesis MIT.

Winograd, T. (1974) Five lectures on AI, Stanford AIM-246.

Winograd, T. (1975a) Frames and the procedural declarative controversy, in Bobrow and Collins (1975)

Winograd, T. (1975b) et al. A knowledge Representation Language talk presented at the conf. on Machine Representation of knowledge, Nato Advanced Studies Institute, Santa Cruz.

Winston, P. (Ed) (1975) The Psychology of Computer Vision. New York: McGraw-Hill, 1975.

F. Malloy Brown

THE ROLE OF EXTENSIBLE DEDUCTIVE SYSTEMS IN MATHEMATICAL REASONING

We argue that significant improvement in deductive ability of automatic theorem provers presupposes the construction of deductive systems which make use of domain dependent mathematical knowledge and which can justify the results of using such knowledge in terms of the axioms of the given domain. We then briefly describe an implementation of a prototype deductive system of this nature for the domain of elementary set theory.

1. INTRODUCTION

We first describe two possible objectives for research on the automation of deductive reasoning:

1. The first objective is to construct a program to determine if a sentence can be proven from the axioms of a theory.
2. The second objective is to construct a program to explain why, in terms of the given axioms of a theory, a sentence can be proven from those axioms.

Consider for example a theory consisting of the following axioms:

A1: $(x+y)+z = x+(y+z)$ associativity

A2: $x+y = y+x$ commutativity

A3: $z+y = x+z \leftrightarrow y=z$ cancellation

We could build a very efficient theorem prover for this theory, which instead of using those axioms to prove theorems, uses the following procedure:

P1: Cancel like variables on opposite sides of the equality sign.

with the strategic information that whenever P1 is used the old equation is to be erased.

Such a theorem prover would satisfy objective 1 because this procedure is essentially a derived rule of inference of the theory consisting of axioms A1, A2 and A3.

For example, a proof that $a+(b+c) = c+(b+a)$ is a theorem using this procedure could be obtained as follows:

```
a+(b+c) = c+(b+a)
        ↓                    by P1 (cancel a)
   b+c  = c+b
        ↓                    by P1 (cancel b)
     c  = c
        ↓                    by logic
        ■
```

This theorem prover does not satisfy objective 2 because the inference steps in this proof are not justified in terms of the primitive axioms of the theory, but only in terms of P1. However, a theorem prover for this theory which also satisfies objective 2 could be obtained by substituting equals using the axioms A1, A2, and A3. In this case a proof of a+(b+c) = c+(b+a) would be:

```
a+(b+c) = c+(b+a)
        ↓                    by A1
a+(b+c) = (c+b)+a
        ↓                    by A2
a+(b+c) = a+(c+b)
        ↓                    by A3
   b+c  = c+b
        ↓                    by A2
   b+c  = b+c
        ↓                    by logic
        ■
```

Now it is easy to verify that a deductive system which embodies extra mathematical knowledge in terms of lemmas, procedures like P1 and the strategic knowledge as to how to use those lemmas and procedures, has the potential for being more efficient at proving theorems than systems which do not. For example, the first theorem prover we described which embodies the procedure P1, proves a+(b+c) = c+(b+a) without any search. On the other hand, the second theorem prover which uses only the axioms A1, A2, and A3 involves search due to superfulous applications of the axioms. For example, all of the following seven sentences, six of which are superfulous, are obtainable from the theorem a+(b+c) = c+(b+a) by a single application of one of the axioms:

```
    (a+b)+c = c+(b+a)        by A1
    a+(b+c) = (c+b)+a        by A1
    (b+c)+a = c+(b+a)        by A2
    a+(c+b) = c+(b+a)        by A2
    a+(b+c) = (b+a)+c        by A2
    a+(b+c) = c+(a+b)        by A2
x+(a+(b+c)) = x+(c+b+a))     by A3
```

Note that if the second theorem prover had some fairly sophisticated search strategic knowledge which restricted the use of the associativity and commutativity axioms so as to directly lead to the use of the cancellation axiom, then, there might very well have been no search. The problem with such strategies in a more complex theorem proving system, is that out of all the many possible strategies how does the theorem prover know which one to apply to which expression? Suppose for each strategy we had some very efficient method of testing whether it should be used on the given expression, then it follows that our problem would be solved. The question remains as to just what are the efficient methods? The answer is that these methods are simply items of extra mathematical knowledge. For example in the case of the sophisticated search strategy that we gave for the second theorem prover, an efficient method to determine whether it should be used is simply the procedure P1.

In the remainder of this paper we shall continue to speak of items rather than complex strategies for what is important is not the mere existence of various strategies but rather the methods of determining when a particular strategy should be used. Since every search space including the one produced by adding various items, involves some strategic considerations, we shall reserve the word 'strategy' for this purpose. However, these strategies will not be very complex and will have simple, even obvious methods of determining when they should be used. An example of such a strategy was the erase strategy used in conjunction with P1.

Given then, that the most efficient deductive systems embody extra mathematical knowledge such as the procedure P1; and that while such systems trivially satisfy objective 1, a certain amount of extra effort must be made if such systems are also to satisfy objective 2; one may well wonder if there is any purpose at all in requiring objective 2 to be satisfied. That extra programming effort must be made in order for such systems to satisfy objective 2 can be seen from the fact that extra mathematical knowledge such as lemmas and procedures such as P1 would have to be justified in some sense from the axioms of the theory. For example, lemmas could be justified by proving them using only the axioms of the theory, but what about procedures written in LISP or machine code? We leave this technical question until section 2 and now state why deductive systems should satisfy objective 2 in the first place.

The reason is this: any theorem prover not continually able to increase it's mathematical knowledge, such as lemmas, procedures, and strategies will quickly reach a plateau in the level of difficulty of theorems it can prove, after which a combinatorial explosion must occur in it's search space.

Consider, for example, some logical system with equality, such as resolution with paramodulation; with a two sorted theory consisting of Tarski's axioms of real numbers and Peano's axioms of number theory. A proof procedure based on this system would have little trouble proving:

$$(a+b)+c = c+(b+a)$$

but would find:

$$\exists x\, \exists y\, \exists z\, \exists u\, \exists v\, \forall n\, \left(\sum_{k=1}^{n} k^3 = x \cdot n^4 + y \cdot n^3 + z \cdot n^2 + u \cdot n + v\right)$$

very difficult indeed.

On the other hand, it is not difficult to implement a more knowledgeable algebra-number theory theorem prover, embodying about 70 lemmas, strategies, and procedures which could easily prove the above theorem. In fact our algebra number-theory system [1] proves this theorem in about 24 seconds. Of course, since this theorem prover cannot itself further increase it's mathematical knowledge, it too soon reaches a plateau in the level of difficulty of theorems it can prove. For example, it is unable to prove the theorem:

$$\forall p\, \exists x\, \forall n\, \sum_{k=1}^{n} k^p = \sum_{j=o}^{p+1} (Xj) \cdot n^j$$

Having now exemplified the fact that a mathematical system must be able to increase it's mathematical knowledge, it is easy to see why it's deductive component must satisfy objective 2. For consider, if objective 2 were not required to be satisfied how would our program itself be able to tell if the mathematical knowledge it was adding to it's deductive component, were correct or even consistent with the original axioms of the theory. For example, in the theory consisting of axioms A1, A2, and A3, what would there be to stop a procedure such as P2 from being acquired:

P2: Cancel unlike variables on opposits sides of the equality sign.

Note that whereas P1 was essentially a derived rule of inference of this theory, P2 is not. For whereas a+b = a+b is true in this fragment of arithmetic, applying P2 to it gives the sentence a = b (i.e. $\forall a \forall b\ a = b$) which is false in arithmetic.

In summary, the goal of our research programme, in respect to deductive reasoning, is to construct deductive systems which

(1) make use of extra mathematical knowledge (eventually to be created by other components of the mathematical system). (That is to make use of efficient methods of calculating which strategies should be

used on which expressions.)

and

(2) satisfy objective 2.

For example, we would like to have a theorem prover as knowledgeable, and hence efficient, as our algebra-number theory program, but which could justify it's proofs in terms of the Peano and Tarski axioms.

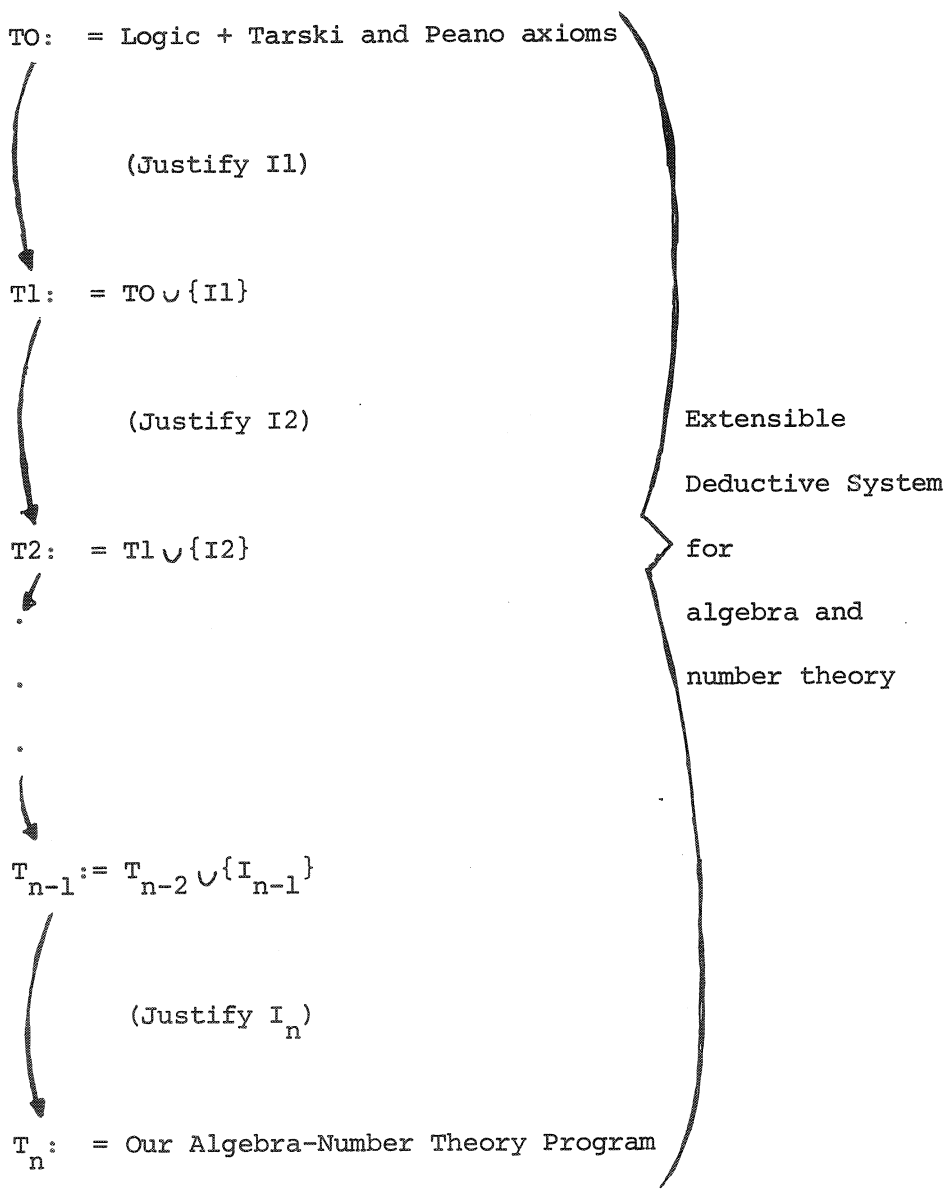
2. EXTENSIBLE DEDUCTIVE SYSTEMS

How is it possible to construct deductive systems which use extra mathematical knowledge, and yet which satisfy objective 2? We intend to realize this goal by constructing extensible deductive systems:

An extensible deductive system is a theorem prover which starting from some system of axioms justifies more and more items of extra mathematical knowledge such as lemmas and procedures, one after another, creating ever more knowledgeable theorem provers. That is, after each such item is justified, it itself is organized into the deductive system, and is used in justifying further items.

For example, an extensible deductive system resulting in our algebra-number theory theorem prover could be obtained as follows: (see Fig. 1) let $I_1 \ldots I_n$ be all the items of extra mathematical knowledge in our algebra number theory program. Each Ij will be representable in our mathematical language and hence by definition is a lemma, not a procedure. However, for the purposes of justification procedures are easily represented in our mathematical language. For example, the procedure P1 could be represented as being the lemma which is the conjunction of the axioms A1, A2, and A3. Let the theorem prover T_{j+1} be obtained from the theorem prover T_j by organizing into the system the item I_{j+1} after justifying I_{j+1} using T_j. It is important to understand that organizing an item into the system means a lot more than merely adding another sentence to some data base. In particular it means that knowledge about how and when the item is to be used is created and added to the system. For example, in the case of the lemma version of the procedure P1 (i.e. the conjunction of axioms A1, A2, and A3) what we want to create and add is an efficient LISP function to do the cancellation along with relevant information expressed in LISP as to when the system should attempt to use this function. Note that by allowing the full power of a language like LISP, it is not very difficult to represent strategic knowledge as to how to use items. Then by letting T_o be a theorem prover for logic including equality with the Tarski and Peano axioms, it follows that T_n will be

Figure 1: Example of extensible deductive system



our algebra-number theory program. Furthermore, the spectrum of theorem provers $\langle T_o, \ldots, T_n \rangle$ will form an extensible deductive system resulting in T_n. That is any further theorem proven by T_n will be justifiable, via $T_1 \ldots T_{n-1}$, eventually in terms of the primitive axioms to T_o.

In order to experimentally test these ideas, last year we implemented [2] a prototype extensible deductive system for a small portion of the set theory given in Quine's book Set Theory and its Logic [3]. The initial theorem prover T_o of of this extensible system consisted of a sequent calculus with equality, and about 30 axioms and definitions. A few of these definitions are listed below: (Note that $\{u:\phi u\}$ is the abstract of all things: u such that ϕu holds; $\{x\}$, $\{xy\}$, $\langle xy \rangle$ are the unitset, pairset, and ordered pair respectively; and $\{\langle xy \rangle : \phi xy\}$ is the abstract for all ordered pairs such that ϕxy holds.)

D1: $x \in \{u:\phi u\} \leftrightarrow \phi x$

D2: $\{x\} = \{z:z=x\}$

D3: $\{x\ y\} = \{z:z=x \vee z=y\}$

D4: $\langle x\ y \rangle = \{\{x\}\ \{xy\}\}$

D5: $\{\langle xy \rangle:\phi xy\} = \{z: \exists x \exists y\ \langle xy \rangle = z \wedge \phi xy\}$

D6: $x=y \leftrightarrow \forall z.z \in x \leftrightarrow z \in y$

Succeeding theorem provers T_{j+1} were obtained by organizing into the system certain given lemmas I_{j+1} that had been proven using the preceeding theorem prover T_j. All items in the system were lemmas not procedures. Furthermore, the actual process of organizing was not automated. Instead the organized LISP representation of each lemma was initially given, but not used until automatically activated by a proof of that lemma. These lemmas are used in a manner similar to definitions by replacing what matches the left side of the equivalence by that instance of the right side. Furthermore, when several items might be applied to a sentence, the system prefers to use the the least primitive item, usually the one justified last. Thus, definitions are used only if no other item could be used. A few of the lemmas in our set theory system are given below:

I1: $x = x \leftrightarrow \blacksquare$

I2: $\{x\} = \{y\} \leftrightarrow x = y$

I3: $\{xy\} = \{z\} \leftrightarrow x = z \wedge y = z$

I4: $\{z\} = \{zy\} \leftrightarrow z = x \wedge z = y$

I5: $\{xy\} = \{uv\} \leftrightarrow x = u \wedge y = v) \vee (x = v \wedge y = u)$

I6: $\langle xy\rangle = \langle uv\rangle \leftrightarrow x = u \wedge y = v$

I7: $\langle xy\rangle \in \{\langle uv\rangle : \phi uv\} \leftrightarrow \phi xy$

Consider now the proof of lemma I7 using the theorem prover T6. From Figure 2 we see that two of the inference steps in this proof were obtained by using the lemma I6. Thus this proof is not immediately in terms of the primitive axioms of this set theory. However since I6 was proven from T5 using I2, I3, I4, and I5, and since each of these lemmas was proven using only T1, and since I1 was proven from the primitive axiom, namely T0, it follows that the lemma I7 has actually been justified in termsof the primitive axiom of this theory.

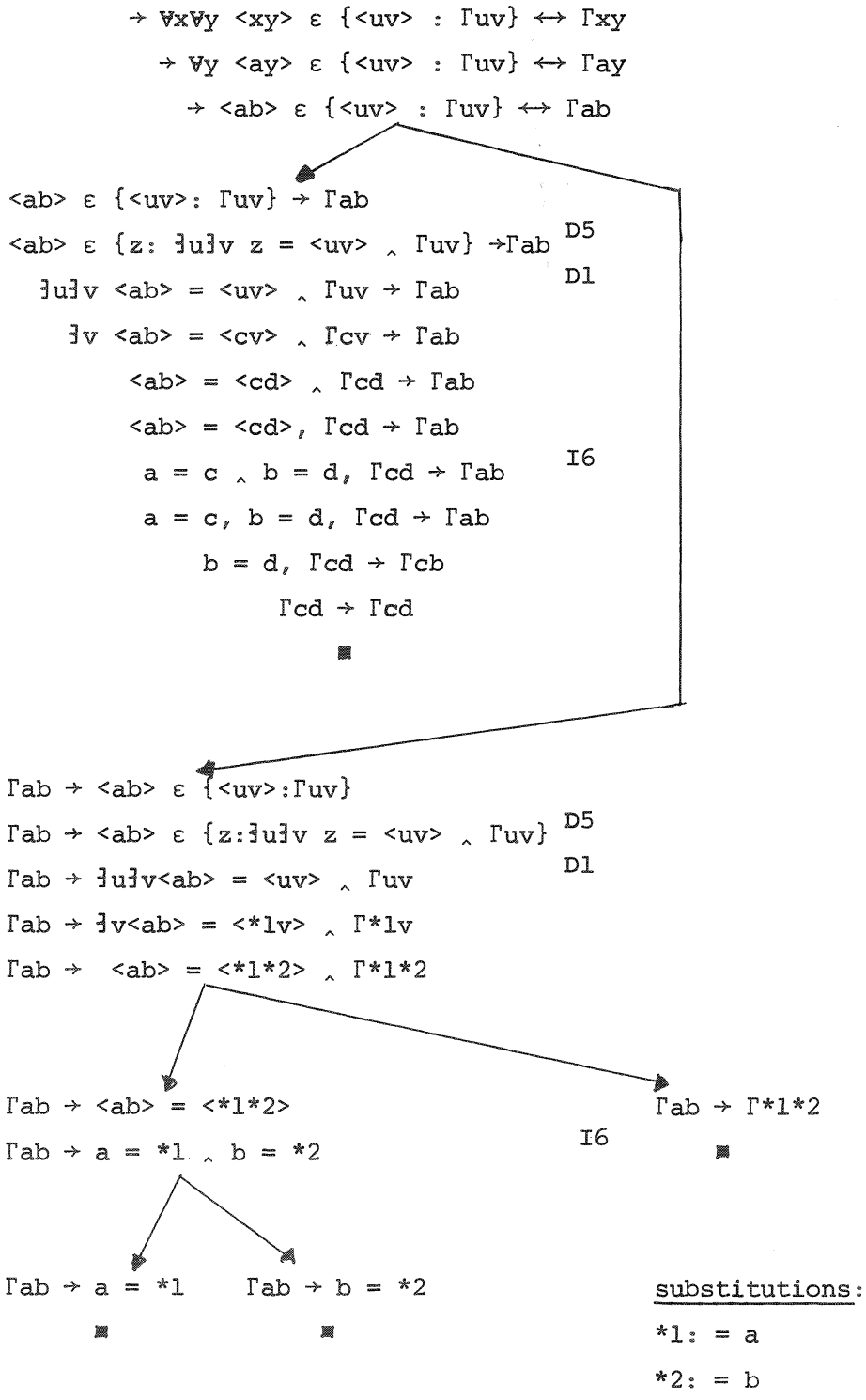
Note that items of extra mathematical knowledge such as lemmas I1, ..., I7 play an indespensible roll in the deductive process not only because they cut down the sheer length of the proof of a theorem, but also because they cut down the possibilities of interaction between various subformulas of a theorem. For example, in the proof of I7 given in Figure 2, if the lemma I6 had not been used then the theorem prover would have had to show that various instances of I6 held; namely that $\langle ab\rangle = \langle cd\rangle$ was equivalent to $a = c \wedge b = d$ and that $\langle ab\rangle = \langle *_1 *_2\rangle$ was equivalent to $a = *_1 \wedge b = *_2$. Not only does this make the proof of I7 much longer but also the proofs of those instances of I6 must be carried out in the presence of sentences such as: Γab, Γcd, $\Gamma *_1 *_2$. If Γ were a large complex sentence, it is easy to see that there would be many possibilities of interaction between it and the intermediate formulas produced when proving the instances of I6. Continuing in this fashion, if the use of further lemmas is disallowed, say for example lemmas I2, I3, I4, and I5 which were used in proving I6 then the resulting proof must be even longer, and there become more and more possibilities of interaction, eventually creating a combinatorial explosion. On the other hand, by using items of extra mathematical knowledge such as the lemmas I1...I7, the proofs of further theorems in elementary set theory do not appear to be any longer, or any more difficult to obtain than the proofs of earlier lemmas.

For these reasons, it is plausible to suggest that by using items of extra mathematical knowledge in extensible deductive systems, that efficient theorem provers satisfying objective 2 can be constructed.

3. CONCLUSION

If our goal is to study mathematical reasoning, then we must study the ability of a mathematical system to improve its deductive capabilities by the acquisition of more sophisticated mathematical techniques. If we are to know that such tech-

Figure 2: Proof of lemma I7 using T6



niques are correct, then our deductive systems must be able to formally justify them. Thus it is not enough to construct theorem provers which fail to satisfy objective 2; we must construct extensible deductive systems.

ACKNOWLEDGEMENTS

I wish to thank Bernard Meltzer for encouraging this research, and Martha Stone for reading the draft. This research was supported by a scholarship awarded to the authors by the University of Edinburgh, and is currently supported by a grant BRG 94431 from the Science Research Council.

REFERENCES

1. F. Malloy Brown. "Towards the Automation of Set Theory and its Logic". To appear in 1976.

2. F. Malloy Brown. "A Theorem prover for elementary Algebra". To appear in 1976.

3. W.V.O. Quine. "Set Theory and its Logic". Revised Edition 1969. Oxford University Press, London; Library of Congress Catalogue Card Number 68-14271.

F. Malloy Brown

A DEDUCTIVE SYSTEM FOR ELEMENTARY ARITHMETIC

We describe a theorem prover for elementary number theory based on truth value preserving transformations, and present some theorems which it has proven.

1. INTRODUCTION

This is a report of work completed in seventeen days during the fall of 1974. It describes an automatic theorem prover for the elementary theory of natural numbers which represents theorems as lists and proves them by applying truth value preserving transformations.

2. DESCRIPTION OF THE THEOREM PROVER

Our theorem prover consists of an interpreter for mathematical expressions and many items of mathematical knowledge. This interpreter is a fairly complex mechanism, but for the purpose of describing this theorem prover it may be viewed as applying items of mathematical knowledge of the form: $\phi \leftrightarrow \psi$ or $\phi = \psi$ to the theorem being proven, in the following manner: The interpreter evaluates the theorem recursively in a call by name manner. For each sub-expression that the interpreter evaluates, in turn it tries to match the ϕ expression of each item to that sub-expression. If ever it finds a sub-expression $\phi\theta$ which is an instance of ϕ of some item, then it replaces that expression by the corresponding instance $\psi\theta$ of ψ. At this point all memory of the sub-expression $\phi\theta$ is immediately lost and the interpreter now evaluates $\psi\theta$. If no items can be applied to a sub-expression then the sub-expression is not evaluated again but is simply returned.

For example if $x+o = x$ and $x+o = o+x$ are the only items and if they are listed to be used in that order then evaluating the theorem $A+(B+o)\geqslant A$ will cause the sub-expression B+o to be replaced by B resulting in $A+B\geqslant A$. All memory of the sub-expression B+o is immediately lost upon its replacement by B and thus the interpreter does not attempt to apply the second item to B+o.

This theorem prover has a great deal of logical and arithmetic knowledge. We first describe the items of logical knowledge, and then the items of arithmetic knowledge.

2.1 Logical Knowledge

Our theorem prover has knowledge about twelve logical symbols which are listed below with their English translations:

$\wedge$	and
$\vee$	or
$\sim$	not
$\blacksquare$	true
$\square$	false
$\supset$	implies
$\leftrightarrow$	iff
$\exists$	there exists
$\forall$	for all
$=$	equal
$\rightarrow$	sequent arrow
and	implicit and between sequents

The sequent arrow is defined as follows:

$$\phi 1,\ldots,\phi n \rightarrow \psi 1,\ \ldots,\psi n = \text{df}\ (\phi 1 \wedge \ldots \wedge \phi n) \supset (\psi 1 \vee \ldots \vee \psi n)$$

where ϕi and ψj are sentences. Thus a sequent may be thought of as being a database of true statements $\phi 1, \ldots, \phi n$ called assertions which occur before the sequent arrow, and false statements $\psi 1, \ldots \psi n$ called goals which occur after the sequent arrow. The implicit conjunction of different sequents may be thought of as being a group of different databases.

The items of logical knowledge, which are all schemata because they involve ellipses, are listed below:

Assertion schemata:

$\blacksquare \rightarrow$: $(\ldots,\blacksquare,\ldots\rightarrow\ldots)\leftrightarrow(\ldots,\ldots\rightarrow\ldots)$

$\square \rightarrow$: $(\ldots,\square,\ldots\rightarrow\ldots)\leftrightarrow \square$

$\sim \rightarrow$: $(\ldots,\sim x,\ldots\rightarrow\ldots)\leftrightarrow(\ldots,\ldots\rightarrow x,\ldots)$

$\wedge \rightarrow$: $(\ldots,x\wedge y,\ldots\rightarrow\ldots)\leftrightarrow(\ldots,x,y,\ldots\rightarrow\ldots)$

$\vee \rightarrow$: $(\ldots,x\vee y,\ldots\rightarrow\ldots)\leftrightarrow(\ldots,x,\ldots\rightarrow\ldots)$ and $(\ldots,y,\ldots\rightarrow\ldots)$

$\supset \rightarrow$: $(\ldots,x\supset y,\ldots\rightarrow\ldots)\leftrightarrow(\ldots,\ldots\rightarrow x,\ldots)$ and $(\ldots,y,\ldots\rightarrow\ldots)$

$\leftrightarrow \rightarrow$: $(\ldots,x\leftrightarrow y,\ldots\rightarrow\ldots)\leftrightarrow(\ldots x,y,\ldots\rightarrow\ldots)$ and $(\ldots,\ldots\rightarrow x,y,\ldots)$

$\forall \rightarrow$: $(\ldots,\forall x\phi x,\ldots\rightarrow\ldots)\leftrightarrow(\ldots,\forall x\phi x,\phi_*,\ldots\rightarrow\ldots)$

where * is a new Prawitz dummy variable

$\exists \rightarrow$: $(\ldots,\exists x\phi x,\ldots\rightarrow\ldots)\leftrightarrow(\ldots,\phi a,\ldots\rightarrow\ldots)$

where a is a new skolem function.

$= \rightarrow$: $(\Pi a,\ldots\{{}^{a=t}_{t=a}\}\ldots,\Gamma a\rightarrow\phi a,\ldots,\psi a)\leftrightarrow(\Pi t\ldots,\ldots,\Gamma t\rightarrow\phi t,\ldots,\psi t)$

where a is a skolem function not occurring in t.

This is our extentionality schema, for example:

$$(A = B+C \rightarrow C\geq A) \leftrightarrow (\rightarrow C\geq B+C)$$

Goal schemata:

→ ■: (...→...,■,...)↔ ■
→ ¤: (...→...,¤,...)↔(...→...,...)
→ ∿: (...→...,∿x,...)↔(...,x→...,...)
→ ∧: (...→...,x∧y,...)↔(...→...,x,...)and(...→...,y,...)
→ ∨: (...→...,x∨y,...)↔(...→...,x,y,...)
→ ⊃: (...→...,x⊃y,...)↔(...,x→...,y,...)
→ ↔:(...→...,x↔y,...)↔(...,x→...,y,...)and(...,y→...,x,...)
→ ∀: (...→...,∀xϕx,...)↔(...→...,ϕa,...)
where a is a new skolem function.
→ ∃: (...→...,∃xϕx,...)↔(...,ϕ*,∃xϕx,...)
where * is a new Prawitz dummy variable.

Other logical schemata:

Atom: (...,x,...→...,x,...) ↔ ■
And : (...and ■ and...) ↔ (...and...)
Unify:(...,x_1,...→...,y_1,...)and...and(...,x_n,...→...y_n,...)
↔
[(...,x_i,...→...,y_i,...)and...and(...,x_n,...→...,y_n...)]θ
where θ_i is any one of the substitutions which makes tautologous the greatest number of sequents starting with the first sequent and progressing towards the nth sequent. For example, an instance of this schema is:

[(P * → Pa,Pb) and (Q* → Qb) and (R* → Ra)] ↔
[Rb → Ra]

Note that the Prawitz dummy variable * was instantiated to b, not to a, because the substitution (*:=b) unifies the first two sequents,whereas (*:=a) unifies only the first sequent.

Finally we note that if all sequents can be made tautologous by some substitution then ■ is returned.

The schemata ∀ → and →∃ are applied only as last resorts. Furthermore, all other schemata are applied before the unify schema.

The fact that only assertion equations are ever eliminated by = → schema implies a preference, for sequents containing only equations, not to have more than one goal equation. The reason for this preference is that there is no logical items which allows two goal equations to interact to produce ■. Thus one of the goals would probably be irrelevant.

An informed reader will recall that these propositional rules are used in Wang's algorithm in the LISP 1.5 manual [1], and that Wang [2] used the other rules restricted to the decidable case of where only skolem constants, but not skolem functions, were necessary. The general idea of unification is due to Prawitz [3] who used rules similar to all the rules given here except for = → and unify. His unification rule

leads to a complete logic, ours does not.

2.2 Arithmetic Knowledge

Our theorem prover has knowledge about ten arithmetic symbols which are listed below with their English translations:

o	zero
1	one
S	successor
P	funny predecessor (PO = O)
+	plus
$\dot{-}$	funny minus ((o $\dot{-}$ x) = O)
•	times
<	less than
$\leqslant$	less than or equal to
|	divides

The items of arithmetic knowledge, some of which are lemmas because they do not involve ellipses, and some of which are schemata are listed below. It should be noted that certain items, as indicated, are applied only to goals, or only to assertions. Such items have been carefully chosen with these restrictions in mind, so as to minimize the introduction of Prawitz dummy variables.

Definitions:

A1 : S x = 1 + x
A2 : P x = x $\dot{-}$ 1
A3 : x<y $\leftrightarrow$ $\sim\exists$u y+u = x if it is a goal
A4 : x<y $\leftrightarrow$ $\exists$u x+(u+1) = y if it is an assertion
A5 : x$\leqslant$y $\leftrightarrow$ $\sim\exists$u y+(u+1) = x if it is a goal
A6 : x$\leqslant$y $\leftrightarrow$ $\exists$u. x+u = y if it is an assertion
A7 : 2 = S1
A8 : 3 = S2
etc.

Items for equality:

A9 : x+y = o $\leftrightarrow$ x = o $\wedge$ y = o
A10: o = x+y $\leftrightarrow$ x = o $\wedge$ y = o
A11: x•y = o $\leftrightarrow$ x = o $\vee$ y = o
A12: o = x•y $\leftrightarrow$ x = o $\vee$ y = o
A13: x+y = 1 $\leftrightarrow$ (x = o $\wedge$ y = 1) $\vee$ (x = 1 $\wedge$ y = o)
A14: 1 = x+y $\leftrightarrow$ (x = o $\wedge$ y = 1) $\vee$ (x = 1 $\wedge$ y = o)
A15: x•y = 1 $\leftrightarrow$ x = 1 $\wedge$ y=1
A16: 1 = x• y $\leftrightarrow$ x=1 $\wedge$ y=1
A17: o = 1 $\leftrightarrow$ ¤
A18: 1 = o $\leftrightarrow$ ¤
A19: o = o $\leftrightarrow$ ■
A20: 1 = 1 $\leftrightarrow$ ■
A21: ... + x + ... = ... + x + ... $\leftrightarrow$ ... + ... = ... + ...

This last item is the cancellation schema of addition, which can be stated as follows: "The sentence containing two occurrences of the same term each connected via any number of plus signs to an outermost equality sign is equivalent to the sentence obtained by cancelling those two occurrences of that term". For example:

$$A + (B + C) = D + (E + B) \rightarrow A + C = D + E$$

Items for divides:

A22: $x|1 \leftrightarrow x = 1$ if it is a goal
A23: $o|x \leftrightarrow x = o$ if it is a goal
A24: $1|x \leftrightarrow \blacksquare$ if it is a goal
A25: $x|o \leftrightarrow \blacksquare$ if it is a goal
A26: $x|\ldots x\ldots \leftrightarrow \blacksquare$ if it is a goal
A27: $x|y \leftrightarrow \exists u\ x \cdot u = y$ if it is an assertion

Lemmas for plus:

A28: $x+o = x$
A29: $o+x = x$
A30: $(x+y)+z = x+(y+z)$
A31: $x+y = y+x$ on certain occasions*

Lemmas for times:

A32: $x \cdot o = o$
A33: $o \cdot x = o$
A34: $x \cdot 1 = x$
A35: $1 \cdot x = x$
A36: $x \cdot (y+z) = (x \cdot y)+(x \cdot z)$
A37: $(x+y) \cdot z = (x \cdot z)+(y \cdot z)$
A38: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
A39: $x \cdot y = y \cdot x$ on certain occasions*

Items for minus:

A40: $o \dot{-} x = o$
A41: $x \dot{-} o = x$
A42: $(x \dot{-} y) \dot{-} z = x \dot{-} (y+z)$
A43: $(\ldots + x + \ldots) \dot{-} (\ldots + x + \ldots) = (\ldots + \ldots) \dot{-} (\ldots + \ldots)$

for example: $A + B \dot{-} C + A = B \dot{-} C$

Lemmas for case analysis:

A44: $\phi[x \dot{-} y] \leftrightarrow (\forall z, x = z+y \supset \phi z) \wedge (\exists u\ x+u = y \supset . \phi o)$ it it is a goal
A45: $\phi[x \dot{-} y] \leftrightarrow (\exists z\ x = z+y \wedge \phi z) \vee (\exists u\ x+u = y \wedge \phi o)$ if it is an assertion

*Merely an implementation hack to save conses in applying associativity lemmas A30 and A38.

Factorization schema:

A46:

$$(\ldots\cdot x\cdot\ldots)\{\overset{+}{\cdot}\}\cdots\{\overset{+}{\cdot}\}(\ldots\cdot x\cdot\ldots)=(\ldots\cdot x\cdot\ldots)\{\overset{+}{\cdot}\}\cdots\{\overset{+}{\cdot}\}(\ldots\cdot x\cdot\ldots)$$

$$\leftrightarrow (\ldots\cdot\ldots)\{\overset{+}{\cdot}\}\cdots\{\overset{+}{\cdot}\}(\ldots\cdot\ldots)=(\ldots\cdot\ldots)\{\overset{+}{\cdot}\}\cdots\{\overset{+}{\cdot}\}(\ldots\cdot\ldots)\vee x=0$$

This last item is simply the operation for factoring out x. An example is:

$$A\cdot B + C\cdot A = D\cdot A \mathbin{\dot{-}} A \leftrightarrow B+C = D\mathbin{\dot{-}}1 \vee A = 0$$

These lemmas are used only when all else fails.

Proof by contradiction:

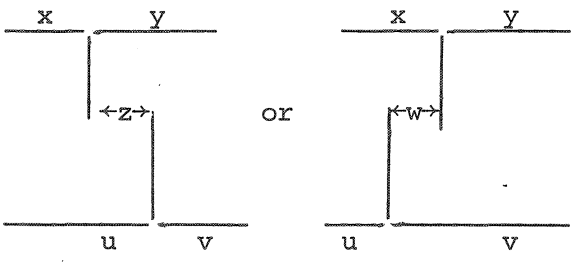
A47: $x = y \leftrightarrow \sim\exists u\ x+1+u = y \wedge \sim\exists w\ y+1+w = x$
if it is a goal and all else fails

The proof by contradiction heuristic is essentially a method of allowing interaction between two goal equations. It does this by transforming one of the equations into assertion equations, thus allowing the extensionality schema = → to be used.

Magnitude heuristic:

A48: $x+y = u+y \leftrightarrow (\exists z\ x+z = u \wedge v+z = y) \vee (\exists w\ u+w = x \wedge y+w=v)$
if it is an assertion and all else fails

The justification that this last lemma is indeed a theorem can be seen by observing:



3. EXAMPLE

We give below a protocol produced by our theorem prover.

Example AM32: No even number equals any odd number.

$\rightarrow \sim\ 2\cdot A = (2\cdot B)+1$
$2\cdot A = (2\cdot B)+1 \rightarrow$
$A+A = (2\cdot B)+1 \rightarrow$
$A+A = (B+B)+1 \rightarrow$
$A+A = 1+(B+B) \rightarrow$: the commutativity hack A31
$\downarrow$: magnitude heuristic A48

$[\exists u\ A+u=1 \wedge (B+B)+u=A] \vee [\exists w\ 1+w=A \wedge A+w=B+B] \rightarrow$

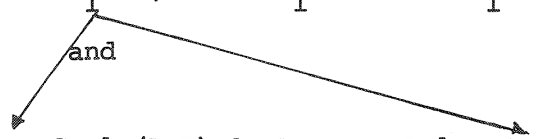


$\exists u\ A+u=1 \wedge (B+B)+u=A \rightarrow$

$A+C_1=1 \wedge (B+B)+C_1=A \rightarrow$

$A+C_1=1,\ (B+B)+C_1=A \rightarrow$

$(A=o \wedge C_1=1) \vee (A=1 \wedge C_1=o),\ (B+B)+C_1=A \rightarrow$



$A=o \wedge C_1=1, (B+B)+C_1=A \rightarrow$

$A=o, C_1=1, (B+B)+C_1=A \rightarrow$

$C_1=1, (B+B)+C_1=o \rightarrow$

$(B+B)+1=o \rightarrow$

$1+(B+B)=o \rightarrow$

$1=o \wedge B+B=o \rightarrow$

$1=o, B+B=o \rightarrow$

$\square,\ B+B=o \rightarrow$

$A=1 \wedge C_1=o, (B+B)+C_1=A \rightarrow$

$A=1, C_1=o, (B+B)+C_1=A \rightarrow$

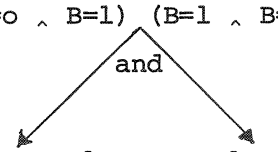
$C_1=o, (B+B)+C_1=1 \rightarrow$

$(B+B)+o=1 \rightarrow$

$B+(B+o)=1 \rightarrow$

$B+B=1 \rightarrow$

$(B=o \wedge B=1)\ (B=1 \wedge B=o) \rightarrow$



$B=o \wedge B=1 \rightarrow$ $\quad$ $B=1 \wedge B=o \rightarrow$

$B=o,\ B=1 \rightarrow$ $\quad$ $B=1,\ B=o \rightarrow$

$o = 1 \rightarrow$ $\quad$ $1 = o \rightarrow$

$\square \rightarrow$ $\quad$ $\square \rightarrow$

$\exists w\ 1+w=A \wedge A+w=B+B \rightarrow$

$1+C_2=A \wedge A+C_2=B+B \rightarrow$

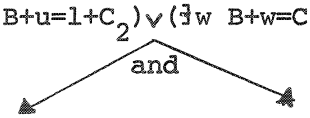
$1+C_2=A, A+C_2=B+B \rightarrow$

$(1+C_2)+C_2=B+B \rightarrow$

$C_2+(1+C_2)=B+B \rightarrow$: again the commutativity hack A31

$\downarrow$: magnitude heuristic A48

$(\exists u\ C_2+u=B \wedge B+u=1+C_2) \vee (\exists w\ B+w=C_2 \wedge (1+C_2)+w=B) \rightarrow$



$\exists u\ C_2+u=B \wedge B+u=1+C_2 \rightarrow$

$C_2+C_3=B \wedge B+C_3=1+C_2 \rightarrow$

$C_2+C_3=B, B+C_3=1+C_2 \rightarrow$

$(C_2+C_3)+C_3=1+C_2 \rightarrow$

$C_3+(C_2+C_3)+1+C_2 \rightarrow$

$C_3+C_3=1 \rightarrow$

$(C_3=o \wedge C_3=1) \vee (C_3=1 \wedge C_3=o) \rightarrow$

and

$C_3=o \wedge C_3=1 \rightarrow$

$C_3=o, C_3=1 \rightarrow$

$o=1 \rightarrow$

$\square \rightarrow$

$C_3=1 \wedge C_3=o \rightarrow$

$C_3=1, C_3=o \rightarrow$

$1=o \rightarrow$

$\square \rightarrow$

$\exists w\ B+w = C_2 \wedge (1+C_2)+w = B \rightarrow$

$B+C_4 = C_2 \wedge (1+C_2)+C_4 = B \rightarrow$

$B+C_4 = C_2, (1+C_2)+C_4 = B \rightarrow$

$(1+(B+C_4))+C_4 = B \rightarrow$

$C_4+(1+(B+C_4)) = B \rightarrow$

$C_4+(1+C_4) = o \rightarrow$

$C_4 = o \wedge 1+C_4 = o \rightarrow$

$C_4 = o, 1+C_4 = o \rightarrow$

$1+o = o \rightarrow$

$1 = o \rightarrow$

$\square \rightarrow$

time = 432 millisec

4. RESULTS

In order to determine whether our theorem prover was very capable of proving arithmetic theorems, we had it attempt to prove a test batch of 87 arithmetic theorems which was used by the SUMS project [4,5]. The overall result was that 86 of the theorems were proven in a total time less than 14 seconds (using a DEC 10 computer with a KA10 central processing unit). The only theorem of the test batch that the theorem prover failed to prove was AEV5: 200+200 = 400 which caused an overflow on an internal stack of the LISP system. This was caused by the fact that the theorem prover represents numerals in unary notation.

A selection of the theorems which were proven is given below:

Theorem		time (ms)
AE12:	$(A \dot{-} B) \dot{-} C = (A \dot{-} C) \dot{-} B$	303
AE13:	$(A + B) \dot{-} C = (A \dot{-} C) + (B \dot{-} (C \dot{-} A))$	151
AF9 :	$C \leqslant B \supset A \dot{-} (B \dot{-} C) = (A + C) \dot{-} B$	75
AF10:	$A \leqslant C \wedge B \leqslant C \supset A \dot{-} B = (C \dot{-} B) \dot{-} (C \dot{-} A)$	76
AF14:	$A \leqslant B \leftrightarrow A \leqslant SB \wedge \sim A = SB$	284
AM7 :	$1 \dot{-} (A + B) = (1 \dot{-} A) \cdot (1 \dot{-} C)$	189
AM15:	$2 \leqslant A \wedge 2 \leqslant B \supset A + B \leqslant A \cdot B$	208
AM28:	$A \mid B \wedge A \mid C \supset A \mid B \dot{-} C$	94
AM32:	$\sim 2 \cdot A = (2 \cdot B) + 1$	432
AEV8:	$2 \leqslant A \wedge A < 4 \leftrightarrow A = 2 \vee A = 3$	496

5. IMPLEMENTATION

Our interpreter for mathematical symbols is implemented in 342 lines of LISP [1] source code. The items of mathematical knowledge are encoded as either formal sentences of our mathematical language or as LISP functions. The logical items are implemented in 531 lines of LISP source code and the arithmetic items, with the exception of a few definitions encoded as formal sentences, are implemented in 336 lines of LISP source code.

In the case of items implemented as formal sentences, our mathematical interpreter, using its pattern matching facilities, works more or less as described in section 2. But in the case of an item encoded in a LISP function the interpreter passes the expression it is evaluating to that LISP function, and expects an equivalent, or equal, expression to be returned.

The run time structure of the theorem prover is as follows:

(1K is 2^{10} words of core memory).

LISP interpreter:	9K
Mathematical Interpreter and Logical items:	6K
Arithmetic items:	2K
The 87 test theorems:	2K
Free cells:	11K
	30K

Most of the LISP functions are compiled.

6. CONCLUSION

We have experimented with our theorem prover and have found it to be capable of proving a fairly wide range of theorems in elementary arithmetic. This provides reasonable evidence that the techniques used by our theorem prover, such as the many truth value preserving transformations, are quite useful for this domain. A more detailed discussion of this theorem prover is given in [6].

ACKNOWLEDGEMENTS

This research was supported by a scholarship awarded to the author by the University of Edinburgh.

REFERENCES

1. J. McCarthy et al. LISP 1.5 Programmers Manual, MIT Press, Cambridge, Mass., 1965.

2. Hao Wang. "Toward Mechanical Mathematics". IBM Journal of Research and Development, v.4, 1960, pp 2-22.

3. D. Prawitz. "An Improved Proof Procedure". Theoria, v. 26, 1960, pp 102-139.

4. Alan Bundy. "Doing Arithmetic with Diagrams". DCL Report No. 61, 1973.

5. Alan Bundy. "Doing Arithmetic with Diagrams". In Adv. Papers - 3rd International Joint Conference on Artificial Intelligence, 1973, pp. 130-138.

6. F. Malloy Brown. "Doing Arithmetic without Diagrams". Dept. of Artificial Intelligence Research Report No. 16. University of Edinburgh, January 1976.

A. Bundy
G. Luger
M. Stone
R. Welham

MECHO[1]; YEAR ONE

This is a progress report on the MECHO project originally announced in D.A.I. Working Paper No. 8, Bundy, Luger and Stone, 1975. The project is to write a computer program which can solve mechanics problems stated in English. This is motivated by a desire to understand how it is possible to form a mathematical model of a real world situation. A problem, typical of those solved by our program, is given and the natural language analysis, equation extraction and the solution of the problem discussed.

0 MOTIVATION

This work is motivated by a desire to understand how it is possible to change the representation of a problem in order to make its solution easier. In particular how it is possible to go from the natural language statement of a problem to a mathematical model from which the problem can be solved. Mechanics seems a suitable area to study this because it provides a rich source of problems which are stated in English using a limited domain of discourse and which are hard enough to be interesting without being intractable.

Secondary motives for studying mechanics problems are that they (a) provide an opportunity to study how semantic knowledge (of physics) can be used to guide the search for the solution to a problem in pure mathematics (equation solving) and (b) there may also be educational spinoff from our formalization of the intuitive physical knowledge required in problem solving. This knowledge is not normally stated in text books, but is essential for solving the problem, and is often overlooked by the unsuccessful problem solver.

1 METHODOLOGY

Our initial approach to solving mechanics problems has been to

---

1 As all the best projects seem to need a silly name (HACKER, DENDRAL, PARRY ...) we have called ours MECHO, short for MECHanics Oracle.

divide the task into parts and to tackle each of the problems relatively independently. The reasons for this approach are explained in Bundy, Luger and Stone (1975). Accordingly we have written programs to

(a) Translate the syntactic parse of a problem statement into a surface-level meaning representation,

(b) Extract equations from a deep level meaning representation,

(c) Solve the resulting simultaneous equations.

This preliminary work has enabled us to build up our descriptive theory. That is, we are developing an ontology for each of the meaning representations and are getting a better idea of the kind of inferences required to go from one representation to another. We also have a better understanding of how the inferencing should be controlled (i.e., about the search strategy).

The second section is divided into three parts. These describe the natural language analysis (A), equation extraction using the "Marples Algorithm" (B) and the solving of the equations (C). The third section discusses the merits of PROLOG in this project.

2 A MECHANICS PROBLEM AND ITS SOLUTION

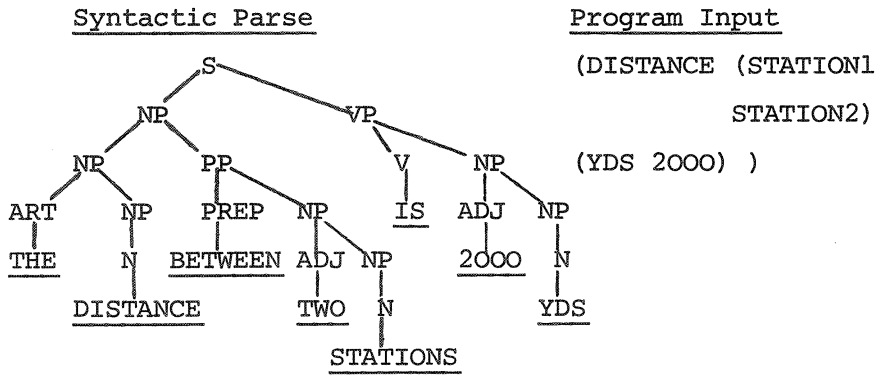
The following mechanics problem is used throughout this paper to illustrate the action of our programs:-

> The distance between two stations is 2000 yards. An electric train starts from rest at one station with a uniform acceleration of A1 ft/sec^2; it comes to rest at the other station with a uniform retardation of A2 ft/sec^2. The speed for the intermediate portion of the journey is constant. Find the constant velocity if the journey is to be completed in three minutes.

A. Natural Language Analysis

The existing natural language program is written in POP-2 and consists mainly of a hierarchical (Bundy & Stone, 1975) database and a set of functions that make use of the database. The database is very important as it contains the information used to guide the parse. At the moment the program only partially performs the task outlined below.

We depend on semantics to dis-ambiguate any parsing problems, and we presently concentrate on developing mechanisms to deal with semantics rather than syntax. The program as it stands accepts as input an embedded list structure that is similar to the output from a simple syntactic parser.



The basic strategy is straightforward. Each clause is examined sequentially. The key word in each phrase (noun for noun phrase, verb for verb phrase) is looked at first. These words, generally representing entities or relationships, will each have at least one major entry in the database. This entry contains information such as which entities can be objects of relationships and which have certain attributes or entities associated with them. For example, acceleration is a quantity, and quantities are known to have a measure, a unit, and a direction. In the case of an entity, the words on the same phrase level are checked against the database to see how they can be used to expand the definition of the entity. For example, in the clause

((TRAIN ELECTRIC) (STARTS(REST) (STATION ONE))
(ACCELERATION UNIFORM FT/SEC^{2}A1)),

ACCELERATION as a quantity is first recognized as an attribute of a particle. UNIFORM is defined as "CONSTANT", an attribute of a quantity; and FT/SEC^2 A1 is recognized as a particular instance of a quantity. When it was recognized that a particle (the TRAIN) had been put into motion, it was noted that there should be a corresponding acceleration. If one assumes that A1 is the appropriate acceleration, the following assertions can be made:

ACCEL (TRAIN,AQ1,PERIOD1)
MEASURE (AQ1,A1)
UNIT (AQ1,FT/SEC^2)
CONSTANT (AQ1)

(PERIOD1 refers to a period of time at the start of which the motion of the train begins)

For a relationship, any previously mentioned entities are examined to see if they qualify as arguments. As the rest of the clause is parsed, new entities are also tested for suitability as arguments.

In the third clause: ((IT) (COMES-TO-REST (STATION OTHER)
(RETARDATION UNIFORM (FT/SEC^2 A2)))),

"IT" is recognized as a pronoun referring to a solid object. "COME-TO-REST" is a verb that indicates the completion of a period of motion of a solid object. This would allow the following assertion:

MOTION (IT,PERIOD2).

But before making this assertion, the data structure is searched for a referent for "IT". The "TRAIN" is the only solid object presently described as being in motion. "IT" and "TRAIN" can be equated, and the following assertion made:

MOTION (TRAIN,PERIOD2)

After the first pass a focal point for the problem is established, in this case the journey of the train. Any isolated pieces of the data structure are examined in a second pass to see if they can be linked to this overall structure. The first clause implied these assertions:

PATH (DØ,STATION1,STATION2)
MEASURE (DØ,2ØØØ)
UNIT (DØ,YDS)

These assertions were not directly related to anything. On the second pass STATION1 and STATION2 are seen as endpoints of PATHØ, the path taken by the train, and

EQUAL(PATHØ,DØ) can be asserted.

The structure is carefully checked for contradictions, and eventually will be passed on to the equation extractor in the form of PROLOG clauses:

JOURNEY (TRAIN,EPISODE,PATHØ)
SEGMENT (EPISODE,PERIOD1.PERIOD3.PERIOD2.NIL)
INITIAL (PERIOD1,DEPARTURE)
INITIAL (PERIOD3,CHANGE1)
INITIAL (PERIOD2,CHANGE2)
FINAL (PERIOD2,ARRIVAL)
etc.

B. The Marples Algorithm

All programs for solving applied mathematics problems employ some device for extracting equations from a semantic database. Often this is very simple as in Charniak's program which extracted all possible equations. It is possible to do better than this by avoiding irrelevant equations. The best explanation we have seen of how to do this is Marples' description (Marples 1974) of the behaviour of engineering students. Our algorithm is based on his description.

Our algorithm distinguishes between symbolic quantifiers for which solutions are required (sought unknowns such as the "constant velocity" in the example above) and those which can legitimately appear in such solutions (givens such as accelerations A1 and A2). The first sought unknown is removed from the list of sought unknowns and an equation is formed which contains it. In general this can be done in several ways. We prefer equations which involve only sought unknowns or givens and do not introduce any further intermediate unknowns. Unfortunately we may be forced to introduce some intermediate unknowns in which case these are added to the end of the list of sought unknowns. The sought unknown for which we have just solved is added to the list of givens and a record of the equation we have just formed is remembered. The process is repeated recursively until there are no further sought unknowns to be solved for. The equations are then subjected to an "independence" check, to make sure the same equation does not appear twice and that only two (of a possible five) constant acceleration equations are used.

Equations are actually formed by the PROLOG procedure "MAKEEQN", each clause of which corresponds to a certain physical formula. For example, consider the clause for constant acceleration, v=u+a.t (present velocity equals the initial velocity plus the acceleration multiplied by the time).

```
+MAKEEQN (*V = *U + *A. *T, (CONSTACCEL.1).(*P. *OBJ),*US)
     - ACCEL (*OBJ, *A, *P)    - UNUSED (CONSTACCEL. 1,
       *P. *OBJ, *US)
     - ISVINVAR (*A)      - DIFF (*A, *ZERO)
     - PERIOD (*P)      - DURATION (*P, *T)
     - INITVEL (*OBJ, *U, *P)      - FINVEL (*OBJ, *V, *P).
```

"MAKEEQN" is the name of the procedure; *V = *U . . . . , *US is the calling pattern. The rest of the clause is the body, where - ACCEL (*OBJ, *A, *P) etc. are sub-routine calls. "CONSTACCEL.1" is the name we give to the resulting equation. *US, the list of equations already produced, is the only input to the procedure. These are both used by the independence checking procedure "UNUSED".

The clause can be read as follows:

"We can make an equation V = U+A.T provided: A is the acceleration of some object, OBJ, during some period of time, P; CONSTACCEL.1 in situation P.OBJ is independent of all equations used so far; vector A is constant; A is different from ZERO; P is a period; T is the duration of P; U is the initial velocity of OBJ in period P and V is the final velocity."

These facts are usually checked in the semantic database, although the last two might involve some trivial inferences. The equations extracted from the example considered above are:

```
EQUATIONS - EXTRACTED
V = ZERO + AQ1. TI1 &
T1Ø = TI1 + (TI2 + (TI3 + Ø)) &
V . TI2 = D2  &
ZERO = V + AQ3. TI3 &
DØ = D1 + (D2 + (D3 + Ø)) &
D1 = ZERO . TI1 + 1 / 2 . AQ1 . TI1 : 2 &
D3 = V . TI3 + 1 / 2 . AQ3 . TI3 : 2 &
TRUE
(':' is exponentiation)
```

The units (of velocity, time, etc.) are then standardized and the equations are simplified. The following PROLOG clause, including the list of sought unknowns, is sent to the equation solver:

```
-SOLVE(

Z=A1.T1                                    &
3=T1.6Ø:-1+T2.6Ø:-1+T3.6Ø:-1               &
Z.T2=X2                                    &
Ø=Z+A2.T3                                  &
6ØØØ=X1+X2+X3                              &
X1=2:-1.A1.T1:2                            &
X3=Z.T3+2:-1.A2.T3:2                       &
TRUE,
Z.T1.T2.T3.X2.X1.X3.NIL).
```

The string at the end of the seven equations is the list of "sought unknowns".

C. The Equation Solver

This program is capable of symbolically solving sets of simultaneous algebraic equations for a given list of unknowns. The first step is to select one of the equations and one of the unknowns and to solve the equation for the unknown. The solution is then substituted in the remaining equations and the process repeated until the last equation is solved for the last unknown. At the moment the equation to be solved and the unknown to be solved for are selected by stepping sequentially through the lists of each until a pair is found for which the unknown is solvable by the program. The order in which the equations were extracted (previous section), determines the order within the lists. Later it is hoped to make this process more intelligent by having the program formulate an overall plan or optimum order for the solution of these equations.

The bulk of the program is concerned with solving one equation for one unknown. The strategy consists of successively applying members of a set of rewrite rules to the equation. At present there are 61 such rules in the program, but

since the program is written in PROLOG, an approximation to predicate logic, additional rewrite rules may be added at any time. The rules are not applied at random and the computation is guided by indexing them into sets labelled as useful for a particular strategy. For example, a strategy known as isolation is applied to an equation as soon as there is just one occurrence of the unknown in that equation. The idea behind this strategy is to change the equation to one of the form X=T where X is the unknown and T is a term not containing the unknown. Only the rewrite rules marked "useful to isolation" are invoked by isolation. A typical rewrite rule useful for isolation is "Replace log(U)=V by U=e:V".

Other strategies used by a super-strategy called the basic method are known as collection and attraction. Collection has the job of collecting together occurrences of the unknown and thus reducing the number of occurrences of the unknown in the equation. Thus a typical rewrite rule labelled as useful for collection is "replace 2.sin U.cosU by sin 2U".

Attraction brings occurrences of the unknown "closer together", thus preparing the way for collection. Typically, to attract U and V in the expression U.W+V.W we use the rewrite rule "replace U.W+V.W by (U+V).W".

The basic method implemented in the program tries to apply the strategies of isolation, collection and attraction, recursively. In addition to the basic method, the program has the capability to recognize certain special classes of equations such as linear or quadratic, and can also make a change of unknown. Thus when solving the equation a(sin x):2 + bsinx +c = 0 for x the program first substitutes y for sinx and then recognizes that the resulting equation a.y:2+by+c = 0 is a quadratic in y.

Technical features of the program include a pattern matcher which knows about the commutativity of addition and multiplication,and a package for regarding terms dominated by addition or multiplication function symbols as "bags".

3 EXPERIENCE WITH PROLOG

In most of our mechanics work to date we have used the experimental programming language PROLOG, first developed at Marseille (Roussel, 1975), and currently being improved at Edinburgh (Warren; 1975, 1976). We have written two programs in PROLOG and are in a position to draw some conclusions based on our experiences.

We were very pleased with both the expressive power and speed of PROLOG. It offered all the normal facilities of functional language like LISP or POP-2, with no significant loss of speed. The provision of pattern directed invocation and non-determinism resulted in smaller, more transparent sub-

routines in our programming and a consequent reduction in programming effort. The search mechanism was faster than anything we could have written in a short time. We found PROLOG very easy to learn.

The biggest drawback was the space requirements. At the present time PROLOG is available at the University of Edinburgh in two sizes, 50K and 75K. The 75K PROLOG can only be used in unsocial hours. Earlier versions of our programs exhausted the 50K PROLOG. The latest versions now exhaust 75K. Because the PROLOG default is to be prepared to backtrack to every choice point unless specifically told not to, it uses a lot of space at run time recording these choice points. These choice points are kept even if PROLOG has been told not to backtrack to them. Warren plans to correct this fault (Warren, 1976).

The second major drawback is that the debugging aids are primitive, especially in the early version of PROLOG (SVI) which we are using. Warren has now issued an improved version (SVW) which, together with some further planned improvements (Warren, 1976) seems to meet most of our criticisms. In general, we feel that PROLOG is an exciting new language, fully justifying further development to make it a viable alternative to other A.I. languages.

4 SUMMARY AND CONCLUSIONS

The achievements of the MECHO project so far include three separate programs that respectively:

1) partially implement the task outlined in the natural language section,

2) implement the Marples algorithm for the extraction and independence checking of equations and the conversion of these into a uniform set of units, and

3) implement the "basic method" (Bundy, 1975) for successfully solving sequences of simultaneous equations.

The most important tasks for the future are concerned with providing intelligent links between these programs. The first desirable link would be between a syntactic parser and the existing natural language programs. For instance, contextual information could be used intelligently to determine which of several entries for one word would be the most appropriate in the parse. The final goal would be to have one program that performed the "syntactic" and "semantic" parsing simultaneously, exchanging information between the two.

The gap between the data structure obtainable from the natural language input and the deep level database necessary for extracting equations will eventually be bridged by inferences invoked at this equation extracting stage. This will

inevitably slow down the Marples algorithm. This situation could be improved by employing heuristics designed to reduce the time spent forming irrelevant equations. One possible heuristic might be to divide problems into types corresponding to the chapter headings in applied mathematics textbooks. Associated with each problem type would be an ordered list of equation names. If the problem type could be identified by contextual cues the associated equations would be formed first.

Another line of improvement would be to search for optimality as well as relevancy and irredundancy in the equations extracted. For instance, we could prefer equations which introduced the smallest number of intermediate unknowns.

Finally, there are further questions concerning the ability of PROLOG in the MECHO Project that can only be answered in light of the further development of the storage and debugging features of the language (Warren, 1976).

REFERENCES

Bundy, A., 1975. "Analysing Mathematical Proofs (or reading between the lines)", Proceedings of IJCAI4, Georgia, Cambridge, MIT-AI press.

Bundy, A., Luger, G. and Stone, M., 1975. "A Program to Solve Mechanics Problems Stated in English", Department of Artificial Intelligence Working Paper 8, University of Edinburgh.

Bundy, A. & Stone, M., 1975. "A Note on McDermott's Symbol Mapping Problem", SIGART Newsletter 53, 9-10.

Charniak, E., 1969. "Computer Solution of Calculus Word Problems", pp 303-316, Proceedings of IJCAI1, eds. Walker, D.E. and Norton, L.M. Washington, D.C.

Marples, D.L., 1974. "Argument and Technique in the solution of Problems in Mechanics and Electricity", CUED/C-Educ/TRI, Dept. of Engineering Memo, University of Cambridge, England.

Roussel, P., 1975. "PROLOG: Manuel de Reference et d'Utilisation", Groupe d'IA, Marseille-Luminy, September, 1975.

Warren, D., 1975. "Epilog": Users' Guide to DEC 10 PROLOG (Internal Memo, Department of Artificial Intelligence, University of Edinburgh).

Warren, D., 1976. Proposal to Reduce PROLOG Storage Requirements (Internal Note, Department of Artificial Intelligence, University of Edinburgh).

ACKNOWLEDGEMENTS

We would like to express our gratitude to our colleagues in the

Department of Artificial Intelligence at Edinburgh for their help and encouragement, especially David Warren, for making PROLOG available and advising us on how to use it. This research was funded by SRC Grant B:RG:9449.3 and by an SRC research studentship awarded to Bob Welham.

M. Eisenstadt

PROCESSING NEWSPAPER STORIES: SOME THOUGHTS ON FIGHTING AND STYLISTICS

This paper explores two aspects of story understanding: (a) the representation of human motivations and intentions; (b) different stylistic mechanisms for inter-relating pairs of sentences. The former aspect is examined by attempting to represent some of the concepts necessary to understand a "fighting" episode in a short newspaper story about a civil war. The latter aspect is used to emphasize that some of the inferences necessary to understand a particular story may be guided by "structural" properties of the story which rule out certain expectations. Finally, a possible synthesis of these two aspects is suggested.

INTRODUCTION

Recent AI work on story comprehension has emphasized the importance of representations of human motivations and intentions (e.g. Abelson, 1975; Charniak, 1975; Rieger, 1975; Schank, 1975; Schmidt, 1975). There have also been attempts to take advantage of characteristic structural properties of stories (e.g. folk tales) to help guide comprehension (Rumelhart, 1975). Newspaper stories pose an interesting challenge, because they deal with complex political motivations and intentions, and they are structurally quite different from the fable-like stories and paragraphs which have been popular in AI research. This paper investigates a few of the concepts which may be necessary to comprehend stories involving episodes such as "fighting" and then argues that we need a deeper understanding not only of the behaviour of the participants in a story, but also of the behaviour of the writer of a story. Writer-behaviour is discussed in terms of devices for maintaining intersentential cohesion. The analyses presented below are "passive" or "post hoc", in that they do not prescribe precise story-processing mechanisms, but rather hint at the direction such mechanisms might take.

The Problem

Given a newspaper story, a comprehension system should at least be able to construct an internal representation which reflects the causal (or other associative) links between the events, particpants, and setting mentioned in the story.

Such links often require complicated inferences, and their formation may represent the very act of "comprehension" (cf. Rieger, 1975). To illustrate this, let's look at an actual newspaper story:*

> 250 ARE KILLED IN CIVIL WAR
> At least 250 people are reported killed in four days of fighting between rival gangs in Angola.
> And more than 700 have been injured in the capital Luanda.
> Fighting flared between three nationalist groups in the African country's transitional government.
> Angola is due to become independent of Portugal in November.

An internal representation which reflects comprehension of this story might contain, among other things, the following information:

(Inference A): The "kill" and "injure" episodes both occur within the scope of the same "fight" episode.

(Inference B): The three groups referred to in the story each desire a change in the control of the government from Portugal to themselves; these desires are mutually exclusive, and are the cause of the fighting.

The central question we're asking is: what are the relationships among all the events (and participants) mentioned in the story? A program which could infer (A) and (B) above would go a long way towards clarifying such relationships, and thus towards comprehending the story. What does a story-comprehender need to know in order to infer these relationships? Knowledge about "fighting", "nationalism", and "civil wars" would certainly be useful. In addition to this, knowledge about the structure of newspaper stories could perhaps help to piece together the relationships stated in Inferences A and B.

---

* This story appeared in The Sun, 3rd May, 1975. The "popular" or "tabloid" papers, while often guilty of atrocious writing, tend to presuppose less knowledge on the part of the reader than do the "serious" papers, and thus contain stories which are more easily handled by a computer model with a (necessarily) naive political background. Note that this story appeared before "Angola" became a household word. The type of processing which takes place in a politically naive model would be more like that of a person who had never heard of Angola than it would be like that of the readers of this

This latter type of knowledge ("discourse knowledge") might not be as important as the former type ("domain knowledge"), but the two together should place powerful constraints on the scope of inferences which must be attempted in order to establish associative links among elements of a story. Let's first take a look at how "domain knowledge" would help to establish such links, and then go on to show how "discourse knowledge" would be of further assistance.

DOMAIN KNOWLEDGE: "FIGHTING".

How can we draw a link between the "kill" and "injure" events in the first two sentences of the Angola story (i.e. make Inference A)? It certainly seems reasonable that knowledge of the concept of "fighting" should lead, directly or indirectly, to knowledge of the possibility of "killing" or "injuring" taking place. Such knowledge might take the form of a schema which contained a "stepwise indexed algorithm" (Rieger, 1975) indicating various alternative courses a particular instance of fighting might follow. We can think of such a schema as a set of rewrite rules, the skeleton of which would be something like this:

```
FIGHT→HIT  |  SHOOT  |  YELL
SHOOT→FIRE-AT  +  WOUND
WOUND→DIE  |  INJURE
(etc.)
```

As Rieger emphasizes, our knowledge of which particular course of events to expect is influenced by many factors which must be made explicit. For example, if the participants in the FIGHT are friends, we would expect YELL or maybe HIT; if SHOOT did occur, however, we would still recognize it as a "legitimate" action (and we would anticipate certain ramifications of a SHOOT episode involving friends). Such a representation of FIGHT would, of course, make it easy to recognize "injure" and "kill" as occurring within the scope of a fight episode. Unfortunately, this representation runs the risk of tripping up whenever it encounters a sequence of events which was not anticipated by the schema. It is not clear that a "grammar" of human behavioural sequences can be successfully mapped out, even though some highly standardized or ritualized

---

A.I.S.B. paper, who now know quite a bit about Angola. The development of a story from day to day is an important aspect of the comprehension of newspaper stories, but will not be dealt with here.

sequences do occur. Clearly we need a way to capture such regularities, while at the same time allowing unexpected variations to occur.

A useful way to do this is to represent the motivations and intentions which underlie various actions, so that expectations regarding particular sequences of actions can be derived from the underlying representation. This approach has been emphasized by Abelson (1973; 1975), and also dominated Rieger's earlier work (1974). Although it may lead to cumbersome "unguided" inferences, such an approach in principle can handle (i.e. "understand") a greater variety of action sequences, because it is less committed to a particular set of expected actions. Abelson's "conflict theme", for instance, is highly relevant to the kind of processing required to understand the fighting in our Angola story. His theme emphasizes the goal of each participant as that of (roughly) thwarting the other's goal, and shows that this is done by each participant removing some state which would have enabled the opponent to achieve his/her goal.

Figure 1 shows a representation of "fighting" which attempts to elaborate on the conflict theme described above, and to make explicit the possible causes of the fighting.

The style of representation is that used by Norman et al. (1975), i.e. "active semantic networks". In these networks, ovals represent predicates, and the arrows point to the arguments. The arrows are labelled both for user/reader convenience and for data-base manipulation purposes (relations may be traversed in either direction). These "predicates" are not only data structures, but may also contain executable code (typically, the definition of a predicate is given in terms of "decomposition rules" which either assert or verify the existence of more primitive conceptual structures in the data base). In the diagrams, the argument-label (i.e. arrow-label) "A" stands for "Actant", meant to encompass the notions of "agent" and "experiencer". The label "O" stands for "object" and the label "S" stands for "subject". The particular selection of primitives was motivated by the set used by Rumelhart (1975).

The dotted lines in Figure 1 divide it into quadrants for the purpose of simplifying the explanation. The entire figure depicts a "state of affairs", which we can think of as a snap-shot of a "typical instant" in the course of a fight. The lower left hand quadrant represents the fact that X is DOing something (we don't yet know what) which CAUSEs a CHANGE from some state (which is DESIREd by Y) to some unspecified state. The lower right quadrant depicts the fact that Y is DOing precisely the same thing to X. Thus, each is CAUSing the loss of a state DESIREd by the other. The particular states involved might be, in Abelsonian terms, states which enable

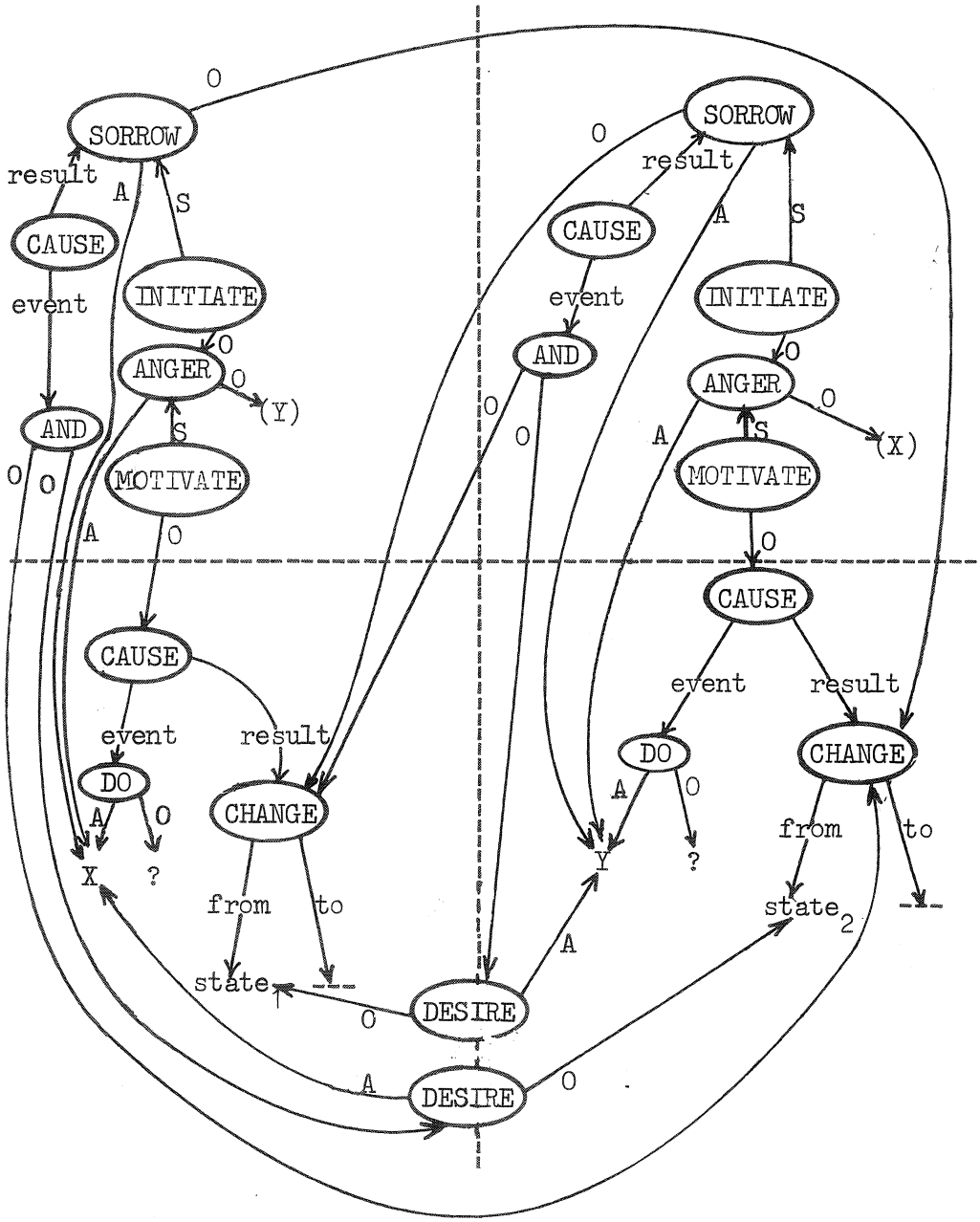


Figure 1: Representation of a "typical instant" in the course of "fighting". The dotted lines divide the figure into quadrants for the purposes of simplifying the explanation given in the text.

the performance of other actions. For such states, we can infer that they are DESIREd by the person for whom they are necessary.

The upper left quadrant reveals that X's SORROW about the CHANGE is CAUSEd by the conjunction ("AND") of two things: the CHANGE itself, and X's DESIRE for the initial (now lost) state. X's SORROW INITIATEs ANGER (at Y, shown in parentheses to avoid crossing lines in the diagram). This ANGER in turn MOTIVATEs X's entire behaviour in the lower left quadrant (i.e. CAUSing the loss of a state DESIREd by Y). The upper right quadrant is complementary to the upper left quadrant, with the roles of X and Y reversed. This fight, then, is simply a sequence of retalliation. Note that since figure 1 represents a snapshot of a typical instant during the course of a fight, the actual initiation of the fight, i.e. the "evil deed" which started it off, need not itself have been MOTIVATEd by SORROW and ANGER.

Now, what have we gained by all of this mental gymnastics? To begin with, the structure presented in Figure 1 provides a foundation for representing a wide range of possible variations, causes, and repercussions of "fighting" episodes. In a sense this is trivially true, since almost any human action could be thought of as DOing something which CAUSEs a CHANGE of some state! One would like to take advantage of this general type of representation, yet still maintain an expectation for specific outcomes (as in the "rewrite rules" shown earlier.) This could be done by specifying a set of possibilities wherever general actions or states are shown in the diagram. Thus, "steal" and "shoot" could be possibilities in place of the "?" associated with the dummy "DO" action in the diagram; possible states that might be affected would be "possession", and "health". The definitions of specific actions would ensure that they are linked with specific states (e.g. "steal" would be linked with "possession"). Any such action-possibility would have associated with it a retalliatory counteraction-possibility. In other words, possibilities for filling in details of the lower left quadrant of figure 1 would be linked to possibilities for filling in details of the lower right quadrant. Particular suggestions for retalliatory actions would be those which affected states comparable to those of the original action (e.g. health, possession), or those which thwarted the original action by removing one of its enabling conditions.

Since the states and actions are only suggested, a certain degree of flexibility is maintained, yet it is easy to recognize "kill" and "injure" as occurring within the scope of a "fight" episode (because "kill" and "injure" can be defined as involving CHANGEs in state of health, which we know to be a state which might be affected in the course of "fighting").

This gives us (the relatively simple) Inference A, which was one of our target inferences for demonstrating comprehension. The crucial thing now is to show that Inference B (about the cause of the fighting) can be derived from the same representation as depicted in figure 1, and thus further justify the representation as capturing an important part of what people know about "fighting". This requires that Y's behaviour in the lower right quadrant of figure 1 be hypothetical (i.e. in the future) from X's point of view. If X and Y share a common DESIRE to attain some state (such as control of the government), then Y's attainment of that state may very well preclude X's attainment of it. This would effectively CAUSE SORROW (upper left quadrant of figure 1), leading to ANGER which would MOTIVATE retalliation against Y, and then a full-blown fight would be underway. Since Y's behaviour is hypothetical, however, the anticipated SORROW and ANGER will MOTIVATE X to thwart Y's action, rather than retalliate against it. The representation for this is precisely the same as that shown for retalliation (i.e. the lower left quadrant of figure 1): X CAUSEs the loss of a state DESIREd by Y-- in this case the particular state is some state which would have enabled Y's hypothetical action to be performed. Given this analysis, a search for the causes of "fighting" could concentrate on DESIREd states, particularly those undergoing some CHANGE. Although a discussion of representations for "government", "control", "nationalism", and "civil war" is beyond the scope of the present paper, we can momentarily pretend that the appearance of "civil war", "nationalist", "transitional government", and "independent" in the Angola story has introduced or activated some notion of DESIREd states involving, say, "control" of "government". Since there are three groups involved, three sets of DESIREd states exist. The attainment of "control" by one group would preclude "control" by another group, thus CAUSing the loss of a DESIREd state, thus CAUSing SORROW, etc. In this manner, we can at least see a possible way to infer the cause of fighting (i.e. Inference B). This inference is our second criterion for demonstrating comprehension of the Angola story.

This representation, then, is a step in the direction of being able to handle a wide range of possible actions within the scope of some episode, while simultaneously being able to understand the events which caused a given episode to come about. In the case of the Angola story, we have "kill" and "injure" occurring within the scope of a "fight" episode (Inference A), while we also have a set of overlapping desires as a possible cause of the fighting (Inference B).

Note that the mechanisms discussed so far have relied entirely upon a representation of the way certain types of human interactions proceed. Even supposing that we thoroughly

understood human motivations and intentions, how important would this knowledge be in terms of setting up expectations to help guide the processing of a story? For some kinds of stories, such as fables, these expectations could be quite useful in helping to span "inferential gaps" in the story, since fables tend to proceed in the same sequence as the real-world events they depict. Other kinds of stories (e.g. newspaper stories) are structurally quite distinct from fables, since fidelity to real-world sequences need not be observed. It may be the case that expectations based on knowledge of the behaviour of participants in a story would have to be tempered by expectations based on knowledge of the behaviour of the writer of a story, as reflected in the structure of the text involved. Let's take a look at how the latter variety of expectations ("discourse knowledge") could be of assistance in drawing Inferences A and B in our Angola story.

DISCOURSE KNOWLEDGE: STYLISTICS

While it is tempting to try to characterize discourse knowledge by constructing a "text grammar" (cf. van Dijk, 1973; Petöfi, 1973), this ultimately has the same drawbacks as the "behaviour-grammar" mentioned earlier - it is too restrictive. Yet stories and texts do exhibit a definite "cohesion" which distinguishes them from arbitrary strings of sentences. The cohesive relation between pairs of sentences can be thought of as a "stylistic" device, for there are many different ways of maintaining cohesion, and each technique serves a different purpose. How can we characterize different cohesive devices, and how do such devices affect our expectations during text-processing?

Let's begin by regarding a story as a "movie" in which a variety of techniques are available for actually presenting the real-world sequence of events. Thus, a story could be a "straight run", in which case it would relate some observer's view of the events as they occurred. More likely, a story would be a "compressed-time composition" or "time-edit" in which case careful editing has resulted in sticking together sequences in the order in which they occur in real life, but leaving gaps for the reader (viewer) to fill in inferentially. This is the "normal" case, and is illustrated, for example, by the following pair of sentences:

Mary left New York on Friday, headed for St. Louis.
She arrived in St. Louis on Saturday night.

Other types of transition are possible. For instance, details may be highlighted by "zooming" in on them. This may be done either as an "action-zoom" or a "frozen-zoom", where the former

refers to a detail of a sub-action within the scope of some larger action (event), and the latter refers to a close-up of some item, participants, or location within the "frozen" image of a given scene. Thus, the following sentence-pair (from Rieger, 1975) involves an action-zoom as well as a time-edit:

Pete stole Jake's cattle.
Jake saddled his (Jake's) horse.

Table 1 shows a summary of this type of analysis applied to several different styles of cohesive transition. Note that these transitions can occur within a single sentence, and that several different types of transitions may be involved within or between sentences.

Table 1

Name	Meaning	Example
1. Straight run	real-time action	-
2. time-edit	compression of real sequence by editing out inferrable details.	Mary left New York on Friday, headed for St.Louis. She arrived in St. Louis on Saturday night.
3. flashback (reverse time-edit)	jump back to earlier event	Mary left New York on Friday, headed for St. Louis. She had only spent 3 hours in New York, after having come down by bicycle from Boston.
4. Zoom	specify details	
a. action-zoom	specify details of an action	Pete was saddling his horse. First he put a blanket on the horse's back, then he lifted the saddle off

Table 1 (continued)

Name	Meaning	Example
		the ground and placed it on top of the blanket.
b. component-zoom	specify details of some item, participant, or location.	Pete took out his wallet. It was bright yellow, with red polkadots.
5. Perspective	reveal background (opposite of zoom).	
a. action-perspective	reveal larger scope of action in which a given action is embedded.	Pete saddled his horse in an attempt to get to town and get the sherrif.
b. Component-perspective	reveal broader scope of item, location, or participant of an action.	Pete saddled his horse. Everyone else on 42nd street laughed at him.
6. Deep-focus	specifying ("clarifying") detail(s) which simultaneously put an event (or component) into perspective ("context"). This is a classic case of local-global interaction.	He did it, at midnight, in the kitchen, with a knife, to a loaf of bread.
7. Space-edit (cut)	"timeless" jump to another (possibly simultaneous) scene/event; special cases shown below.	(see below)

Table 1 (continued)

Name	Meaning	Example
a). pan ("paired-sub-element cut").	"smooth" transition to related scene/event, usually involves some related subaction or sub-components of larger scene/event.	Everyone at the party was having fun. Pete was loaded. Mary was doing the boogaloo.
b) paired-action cut	"clever" transition device, using related actions to link the two scene/events.	Nancy took out her wallet. Evelyn took out her baseball bat.
c). paired-component cut	like paired-action, but link involves components (participants, location) of an action.	Marcello took out his wallet. My own wallet had long since fallen apart.

Given the variety of transition devices which are available (and Table 1 is by no means exhaustive), it seems fair to say that the "trick" sentence-pairs studied by many AI researchers really constitute a special case (e.g. Rieger's cattle-theft example, given earlier). Expectations based purely on sequences of real-world events may be quite useful for tying together time-edits, and information about sub-actions within other actions may be handy for fitting together action-zooms (and particularly combined action-zoom/time edits), as in the cattle theft example. Nevertheless, it would be rash for a model to make many predictions if a particular style precluded the possibility of the actual fulfillment of those predictions! In newspaper texts, stories tend to begin with some item of interest, and then expand upon this item with elaborate interactions of action-perspective shifts, frozen-perspective shifts, action-zooms, frozen-zooms, pans, and flashbacks. Seldom is there a simple forward time-edit.

Knowledge of stylistic tendencies can help influence the set of expectations about what may appear next in a segment of text, and it is here that a text processor could capitalize on "discourse knowledge". To illustrate this point, let's look at a variation of the first two sentences from the Angola story:

> At least 250 people are reported X-ed in four days of Y-ing. And more than 700 have been Z-ed.

The point of using the dummy verbs "X" "Y" and "Z" is that a certain amount of cohesion is maintained, even with the dummy verbs. Moreover, a particular style of cohesion is still exhibited-- namely, a "pan" from X, which is one sub-action of Y, to Z, which is another sub-action of Y.

The way in which a particular cohesive style is recognized remains unclear at the moment. Nevertheless, if we have reason to suspect that event Z is within the scope of event Y, without even knowing anything about Y and Z, then we've simplified the task of arriving at Inference A (i.e. realizing that "injure" occurs within the scope of the main "fighting" episode). This is not to say that knowledge of cohesive styles is any more important than knowledge about human interactions. It merely emphasizes another source of knowledge which can serve to narrow the range of attempted inferences during story processing.

Inference B (about the cause of the fighting) could be assisted by the recognition of one of the following cohesive devices: flashback, perspective, or deep-focus. Each of these devices can potentially provide answers to the question: "Why did this particular event happen?" The third sentence of the Angola story ("Fighting flared between three nationalist groups in the African country's transitional government") fills in details about who, where, and why, and thus is a deep-focus transition. This analysis assumes that a "transitional government" could somehow lead to fighting, and thus relies more heavily on knowledge of "government" and "nationalism" than it does on knowledge of cohesive style. However, if the inference-making mechanisms ever need some assistance, there is a plausible argument on grounds of cohesion that "transitional government" may have something to do with the cause of the fighting, even if we do not know what "transitional government" means. Similarly, the fourth sentence ("Angola is due to become independent of Portugal in November") is linked cohesively to the rest of the story by means of a "perspective" shift (i.e. providing background context) and helps to answer both "Why is the government transitional?" and "Why is there fighting?"

Inference B, about the cause of the fighting, can thus be assisted by knowledge of cohesive devices, though to a lesser extent than Inference A. Formalization of knowledge about stylistic mechanisms should prove handy as an aid to establish-

ing the type of associative links (e.g. Inferences A and B) which are the criteria for successful comprehension of a story.

CONCLUSION

This paper has examined the roles of "domain knowledge" and "discourse knowledge" in comprehending a short newspaper story. The proposed representation for domain knowledge (i.e. knowledge of human motivations and intentions) has moderately firm footing in current AI methodology. The discussion of discourse knowledge (i.e. knowledge of cohesive devices) is of necessity mainly classificatory, but it does attempt to show how the different cohesive devices could actually help to draw crucial inferences during story processing.

Another way of approaching the problem of discourse knowledge would be to examine the speaker's (writer's) intentions during discourse. The writer's desire to convey information to the reader may motivate her to write things in a particular way. Simultaneously, her desire to sell newspapers (or to please her boss who obliges her to attempt to sell newspapers) may motivate her to write things in a slightly different way. If the reader knows that particular desires on the part of the writer manifest themselves in different stylistic ways, then she can take advantage of this knowledge by using it to help piece together sequences of text. The different types of intersentential relationships, as outlined above, would then contain an underlying basis, namely the writer's intentions. Knowledge of underlying motivations and intentions would provide a system with much greater flexibility for dealing with variations than would a simple list of the different kinds of intersentential relations. While the first part of this paper shows some progress in specifying aspects of motivation and intention, there is still an enormous gap between that and the principles of cohesion and stylistics discussed in the second part of the paper. This gap remains as a target for future work.

REFERENCES

Abelson,R.P. (1973) The structure of belief systems, in Computer Models of Thought and Language (eds R.C.Schank & K.M.Colby), 287-339. San Francisco: W.H.Freeman.

Abelson,R.P. (1975) Concepts for representing mundane reality in plans, in Representation and Understanding: Studies in Cognitive Science (eds D.G.Bobrow & A.M.Collins), 273-309. New York: Academic Press.

Charniak,E. (1975) Organization and inference in a frame-like system of common sense knowledge, in Proceedings of the Conference on Theoretical Issues in Natural Language Processing (eds R.C.Schank & B.L.Nash-Webber), 42-51. Arlington, VA.: Association for Computational Linguistics.

van Dijk,T.A. (1973) Models for text grammars. Linguistics, 105, 35-68.
Norman,D.A., D.E.Rumelhart & the LNR Research Group (1975) Explorations in Cognition. San Francisco: W.H.Freeman.
Petöfi,J.S. (1973) Towards an empirically motivated grammatical theory of verbal texts, in Studies in Text Grammar (eds J.S.Petöfi & H.Rieser), 205-275. Dordrecht-Holland: D.Reidel.
Rieger,C. (1974) Conceptual memory: a theory and computer program for processing the meaning content of natural language utterances. AI Memo 233, Department of Computer Science, Stanford University.
Rieger,C. (1975) Conceptual overlays: a mechanism for the interpretation of sentence meaning in context, in Proceedings of the Fourth International Joint Conference on Artificial Intelligence. Tbilisi, USSR.
Rumelhart,D.E. (1975) Notes on a schema for stories, in Representation and Understanding: Studies in Cognitive Science (eds D.GBobrow & A.M.Collins), 211-236. New York: Academic Press.
Schank,R.C. (1975) Using knowledge to understand, in Proceedings of the Conference on Theoretical Issues in Natural Language Processing (eds R.C.Schank & B.L.Nash-Webber), 117-121. Arlington, VA.: Association for Computational Linguistics.
Schmidt,C.F. (1975) Understanding human action, in Proceedings of the Conference on Theoretical Issues in Natural Language Processing (eds R.C.Schank & B.L.Nash-Webber), 196-200. Arlington, VA.: Association for Computational Linguistics.

Ricky Emanuel
Sylvia Weir

CATALYSING COMMUNICATION IN AN AUTISTIC CHILD IN A LOGO-LIKE LEARNING ENVIRONMENT

Abstract. The fascination for machines shown by autistic children can be exploited in a LOGO-based learning environment. We report an experience with a seven-year-old autistic child whose active and enjoyable explorations in controlling the LOGO turtle formed the basis for the development of *language for communication, both verbal and nonverbal.* The strikingly rapid effect produced involved two features seldom shown by the child, VIZ. the onset of *spontaneous* language based on descriptions of the turtle's behaviour, and the *active seeking out of social interaction.* We argue that this follows from the self-validating effect of understanding and being understood, i.e. sharing a sense of relevance, and that this in turn follows from the highly structured but creatively open-ended nature of the LOGO environment, in which the crucial step of "seeing what is relevant" is made transparently easy.

Keywords: Autism, LOGO, perception, learning environment.

AUTISM

In 1943 Kanner described eleven children whose outstanding fundamental disorder was "the children's inability to relate themselves in the ordinary way to people and situations" (Kanner 1943). Since then there have been many case reports of the syndrome of *early childhood autism,* characterised by a background of serious retardation, in which islets of normal or exceptional intellectual function or skill may appear. The chief features are "autism" i.e. a disturbance of interpersonal relationships as shown by aloofness, an apparent lack of interest in people, an avoidance of eye-to-eye gaze; profound abnormalities in language development; and a variety of ritualistic and compulsive phenomena e.g. an intense preoccupation with particular objects, stereotyped mannerisms and a sustained resistance to change in the environment. (Creak 1961; Rutter 1968).

Not all autistic children show all these behavioural manifestations and indeed other children can show some of these characteristics at some stage in their development. It is the particular combination of these features and the degree to which they are exhibited that distinguishes an autistic child from other children.

WHY WE CHOSE TO STUDY AUTISM

To come into contact with an autistic child is a riveting experience. The combination of alertness and multiple cognitive defects, especially the gross impairment of language, compels attention in a way that the subnormal child does not. Further, to ponder why autistic children come to shun humanity as they do, is to raise the question of the source of the impulse towards communication in normal children. Just as the study of perceptual illusions is interesting largely because it sheds light on the processes of perception, so the exaggerated behaviour patterns of autistics may be able to illuminate aspects of interpersonal behaviour and of cognitive activities such as language and learning.

We favour Trevarthen's view (1974) that the spontaneous *social* gesture is normally a feature of very early infant development and that this is closely associated with the development of SELF AS AGENT. A similar association between a child's *cognitive* development and an environment where he can be the AGENT OF HIS OWN LEARNING characterises the work with computers in education at M.I.T. (Papert 1973) and in our own laboratory (Howe and O'Shea 1976). Our approach to the autistic child seeks to bring together these two ideas. In this report, we give the theoretical basis for our methods and describe a rewarding experience with David, a seven-year-old autistic child, over a period of six weeks. Both the remedial therapeutic potential and the possibility of illuminating aspects of cognitive development seem to us to be of interest.

BACKGROUND TO THE PROJECT

The basic assumption underlying our approach is:-

ALL BEHAVIOUR IS MEANINGFUL, RELEVANT AND VALID FOR THE PERSON CONCERNED.

To expose our view of the possible sources of such meaning, we need to consider one of the Artificial Intelligence (A.I.) accounts of perception (Clowes 1973; Winston 1975) as a constructive hypothesis-testing system involving evocation of organised memory. The elements of such an organisation are internal models or schemata, and a computational account of perception provides the means whereby stimulus fragments can address a set of possible models and a model can address the stimulus pattern. That is to say, it is in terms of internal models that stimulus fragments achieve relevance.

Our perception is thus always mediated by our model of the world i.e. we can only interpret reality, not perceive it directly. Two people can construe the same situation differently since they have different world models. As Clowes puts it:

"We can not SEE. We can only SEE-AS."

He goes on to say:

> "The view of behaviour analogous to this is that it is expressive of purposes supported by meaning ... Even the most bizarre behaviour patterns are meaningful because they are based upon an interpretation of the situational stimulus." (Clowes 1973)

It is by observing an individual's behaviour, both verbal and nonverbal, that we can begin to speculate about his or her internal models, both shared and idiosyncratic. The A.I. view of perception grew out of work on "object" or "thing" perception; when it comes to "interpersonal" perception, the relevant internal models will be those which represent our relationships to our "significant others" i.e. our interpersonal schemata. Weir (1975) explores some of the consequences for person perception of the view that there always is a *seeing-as* process going on, during which we, necessarily, add information to what is given to the senses; we make attributions; we *project*. Clearly, when we move from object perception to person perception, the amount of projection we do is greatly increased, since the idiosyncratic experience of individuals plays a larger part in making sense of highly ambiguous cues. However, there remains a large area of what might be called "shared projections", corresponding to shared world views, on the basis of which we communicate with others. Our hypothesis is that autistic children do not communicate just because they lack such shared projections in their perception both of people and of things.

CAUSES vs MEANING

The connection between the internal models and the surface behaviour of an individual was a central concern of Freud, and is revealed in his view of the symptom as a disguised but meaningful communication. (For a discussion of this notion see Rycroft 1968.) Looking for the *meaning of a symptom rather than its cause*, i.e. the *semantic* problem of giving meaning to behaviour, is rather like the problem of understanding a foreign language. We do not look for the "cause" of a person's particular linguistic behaviour if he speaks a language other than our own. It makes little sense to search for the aetiology of speaking French.

As Szasz (1972) says, "We can only ask with "mental illness", as with language, how it was learned and what it means".

LEARNING

The learning paradigm to which we subscribe sees the *learner as a model builder.* Papert's view of learning as implemented in his LOGO system (Papert 1973) asserts that a child learns by actively exploring his environment, a Piagetian notion. This exploration involves building models or theories of the world, on the basis of which he generates predictions, and which he continues to refine in the light of further experience. Just as in perception to which it is intimately connected, the learning process is concerned with knowing what is relevant in a situation i.e. knowing which aspects to incorporate into the schemata being constructed or extended.

WHAT DOES ALL THIS MEAN FOR OUR WORK ?

What it means is that one cannot dismiss as invalid, the sometimes bizarre, seemingly meaningless behaviour of autistics, or for that matter, any person's behaviour. Behaviour is a meaningful expression of the person's idiosyncratic experience of the world. The "symptoms" of autistics are to be seen as a series of communications. We must ask: Why does a child behave this way instead of some other way ? Only by decoding the communication will we be able to appreciate the particular perceptual vantage point from which the autistic child views the world. And in turn, if we are to help him to be receptive to our communications, we need to speak less ambiguously - to aid in his perception of what is relevant i.e. to aid his understanding. If the perception of relevance is shared, this can form a basis for communication.

And what of his desire for communication with us ? "This is what I have found. Let me tell you about it." We have already stressed our belief that it is only in circumstances where the child is the agent of his own learning that spontaneous communicative behaviour, so conspicuously absent from the repertoire of the behaviour of an autistic child, will begin to manifest itself. Our task is to provide an environment which can generate the fulfilling effect of understanding and being understood.

Having stated our biases, we can proceed with a description of our project.

OUR METHODS

A central component of our method is the child's interaction with a computer. Papert (1973) has devised a LOGO learning environment to enable a child to communicate with a turtle i.e. a computer-controlled device moved about the floor according to a set of instructions (a program written in LOGO) entered into the computer from a typewriter terminal. It has a pen on its underside which can be lowered or raised; with

the pen down, the turtle leaves a trace of its path as it moves along. A LOGO environment has been set up in our laboratory by Dr Howe for use in an ongoing project based on the ideas of learning outlined above, and our present study would not have been possible without access to such a system.

We have adopted a version of LOGO which, instead of using a teletypewriter as an input device, employs a BUTTON BOX consisting of 16 buttons. The button box and the turtle are connected to a small computer (Honeywell 316). Each button represents a turtle COMMAND e.g. FORWARD, BACKWARD, LEFT, RIGHT, PENUP, PENDOWN and HOOT. Pushing the button causes it to light up, as well as command the turtle to execute the required action. The button light is extinguished when the command has been executed. There are also "number" buttons which the child can use in conjunction with the command buttons to cause the turtle to execute the command the requisite number of times. This ICONIC version of the principal instructions in the LOGO programming language has been constructed for use as a preliminary introduction to LOGO, as well as in our study. Details of the language, together with a description of the ICONS we use to represent the turtle commands, can be found in Du Boulay and Emanuel (1975).

REASONS FOR OPTING TO WORK WITH A COMPUTER, and in particular, with the LOGO system.

Some of the advantages of using a computer with autistic children are stressed by Colby when he describes the system he used with non-speaking autistics for over four years. Colby's aim was to use the computer as a catalyst to stimulate language development, and to this end he used games which involve combining computer-controlled displays of pictures and letters with their sounds. The child is the ACTIVE agent in controlling the machine, which in turn is "untiring, predictable, always saying the same things the same way, never angry and never bored" (Colby 1973). Furthermore, a small amount of effort produces a large effect; there is no risk of failure and no notion of a right or wrong answer (Colby and Smith 1971).

All but four of the seventeen children he and his colleagues worked with were judged to have improved in language development after between 50-100 half-hour sessions. All these children confirmed the clinical observation that autistic children often show a fascination for objects and machines which they prefer to people. We too wished to capitalise on this feature, and, similarly, we wanted the child to be more active in the environment so that whatever happens, he can say: "I did it and my doing it make a difference" (Bettleheim 1967). Indeed we wanted to go further than Colby by increasing the range of possibilities for action available to the child, over and above the choice which game to play, i.e. offering the

child scope for creative agency. In addition, and crucially, *we wanted to actively facilitate the perception of a shared relevance as a basis for human interaction.*

We said earlier that part of learning involves knowing what to record in the situation. In a complex setting, it is difficult to see what is relevant and what is not. Autistic children have a notoriously different sense of relevance to us. This is reflected in the highly idiosyncratic nature of their language and Kanner (1946) stresses the importance of trying to discover the context in which any particular apparently "meaningless" phrase was acquired. One of the crucial features of the LOGO environment is that it is relatively easy to isolate the salient features because these correspond to the only elements in the turtle's state which the child can change VIZ the *position* of the turtle, its *heading* and the state of the pen. Thus the child knows "where to look". Knowing this enables him to build up a description or a model which he can then use as a basis for action.

The range of possibilities for action i.e. the way that more complex sequences of actions and their descriptions can be constructed from simple but powerful primitive commands, is creatively open-ended. It is also *structured* with the kind of regularity which characterises the thinking of autistics.* Furthermore the execution of the task is transparent because it has been exteriorised. If the sequence of commands given by the child fails to elicit the response *he* was after, it is easier for him to see why, than would be the case if he was performing an internal task.

Furthermore, we hope to get a synergistic effect between the availability of such descriptions and another feature of the situation VIZ the fact that controlling the turtle involves the idea that in order to use the turtle to achieve some overall goal, the child must give it a directed sequence of individual commands. This could act as a model for the child in terms of which to grasp a notion of communication. The point is that just as learning is not something someone does to you but something you get for yourself, so communication is something you are actively involved in, not something thrust upon you.

---

*Examples of this kind of thinking which involves an overcommittal to regularity, are given by Park (1968) in her revealing and informative book about her autistic daughter, Elly. E.g. using a very good rule, Elly generates ZEROTY and ONETY when told about TWENTY and THIRTY.

OUR EXPERIENCE WITH DAVID

David (born 1.8.67) attends a special unit for autistic children in Edinburgh. He presents a classical picture of autism with no evidence of brain damage.

Before beginning our work with David we observed him at the autistic unit in Edinburgh. He had learned to read and write through behaviour modification techniques. His reading was surprisingly good but very unidimensional and mechanical in texture as was his counting and recital of simple poems. It was difficult to avoid the impression that much of this involved a kind of rote learning and that there was a real lack of understanding. We found his social behaviour in the unit inhibited and withdrawn. He avoided eye contact and responded to questions with a stilted high pitched "unnatural voice". He also showed a reluctance to commit himself to anything. A striking feature is the occurrence repeatedly throughout his extensive case notes of statements of the form

'has never made a spontaneous statement to us, except under stress'
'speech has to be prompted every time'
'no spontaneity - has to be asked again and again'

We saw David 7 times over a period of 6 weeks; each visit lasted about an hour and was videotaped. We also kept a record of the sequence of buttons he pressed. Due to limitations in space, we cannot go into detail about actual sessions with David. For a fuller description of these see Weir and Emanuel (1976).

There are several themes described below, which arose in various guises during the individual sessions.

Themes:

The movements of the turtle correspond to the child's own body movements. Papert's view of learning, as implemented in the LOGO environment, emphasises the fact that understanding new ideas depends on the ability to relate them to already existing schemata, and, in particular, to such schemata as have been developed during the child's active exploration of his environment, i.e. the child's representations of his own movements in space. The process of acquiring understanding is facilitated by the deliberate structuring of the learning environment, and is transparently displayed in David's case by the way he always *acted out* the behaviour of the turtle by reference to his own existing body schemata.

Some examples of this phenomenon were his accurate imitation from the very beginning of the hoot; opening and closing his eyes in synchrony with the button light going on and off each time he pushed it whilst vocalizing "open your eyes, close your eyes". We find sequences of alternate button pushing and

corresponding turtle action preceded or followed by David himself moving in space and vocalising the labels of the buttons. For example, he would shuffle his chair backwards and forwards and say "back ..." "forward..." after making the turtle execute these actions, or raise and lower himself in his chair saying "penup" and "pendown". He frequently poked his belly button before executing the action with his body, thus identifying his belly button with the buttons on the box as the instrument of the action.

Thus the new schemata he constructed while "playing" with the turtle encapsulated the connection he made between his own actions and those of the turtle. During this construction process we observed three striking phenomena, i.e.

(a) He began to predict events on the basis of a growing understanding of the relationship between his button pushing and the turtle's behaviour. For example, he pushed his belly button, said "hoot" and then pushed the hoot button.

(b) There was an element of communication in the way he created explicit acting-out sequences as if to "demonstrate" his understanding to us. While acting out some aspect of the turtle's behaviour, David would actually seek eye contact and vocalize the actions he was performing, as if he were making an "action speech".

(c) He began to vocalize his thoughts *spontaneously* during his turtle play and these spontaneous utterances made immediate sense to us. Furthermore, these vocalizations involved descriptions at an increasingly higher level of complexity, reflecting the progressive differentiation and refinement of his schemata. For example, he began by uttering single words like "turtle", "turn". "left", "drawing", etc. Later he constructed and verbalized the entities "turtle goes backwards and forwards ... backwards and forwards" (while pushing corresponding buttons), and the entity "left and right". Later he linked together all his previous descriptions of the turtle functions and said "Emanuel, make the turtle goes forwards and backwards and left and right, up and down ... hoot". After systematically stepping through the number buttons in conjunction with the hoot button, he volunteered the metacomment "See how it works". Crucially, these utterances were accompanied by a series of SOCIAL GESTURES, e.g. actually seeking eye contact, turning towards the person being addressed.

These remarks contrast sharply with the essentially private nature of much of his usual monologue, such as it is. For periods becoming longer and more frequent, we shared a sense of relevance with him, centred on the turtle "play". During these communicative moments he showed a free flow of affect and looked as though he was enjoying himself. He spoke in a

warm, vibrant low-pitched voice; he would chuckle and hum to himself; actively seek eye contact and even become rumbustiously excited. In short, he was emotionally committed. We learned to recognise a particular set of circumstances which interrupted this communicative interaction. When this happened he switched from the self-initiating, spontaneous boy to the alienated, inhibited, stilted child we had known previously. There was a return of eye contact avoidance; he adopted the previously described high-pitched mechanical voice, saying "yes please" in response to all our questions and he ceased all spontaneous activity. We call this his "passive pupil" role. It seemed as if David had developed at least two MODELS OF HIMSELF in relation to others, i.e. he usually adopted one of two roles, i.e. he was either the "passive pupil" or was "emotionally committed".

The switch occurred in a number of situations in which he was prevented from being the free agent of his own activity. For example, when we introduced the number buttons to him and asked him to recognise the numbers written on the buttons, he switched into this "passive pupil" role. We were intrusively requesting him to perform in a situation which shared features with his behaviour modification experiences. He was being "taught" and did not quite understand what was being expected of him. Another situation which evoked this "passive-pupil" role was when we asked him to do something he was not quite sure he could manage, and didn't want to expose himself to failure. For example, in an early session we asked him to use the turtle to knock down a skittle. He completely ignored the request. However, in the next session, when we were out of the room, he skillfully manoeuvred the turtle to knock down a skittle. However, his reluctance to commit himself in our presence diminished as our communication grew.

When we as observers saw David adopt his "emotionally-committed" model it evoked our model of authenticity. We felt this was the true self of David. Notice how evocation of models was going on at several different levels, VIZ. David's perception of the situation he was in, including the objects and our behaviour towards him, our view of the situation and of David's behaviour in it and our view of our perceptions and his perceptions - an interpersonal situation of truly Laingian proportions (Laing 1971).

DISCUSSION

Much of what we have learned from our six weeks experience with David emerged after the actual event when we pored over the videotape observing David, our own reactions and the interrelationship. The videotape had the beneficial effect of exteriorising our own perceptions.

Underpinning the method we have used is the notion that somewhere along the line in the autist's development, there was a lack of a sense of shared relevance in his relationships with the people around him. His reaction to a world filled with a lack of understanding and not being understood, was to develop a range of sophisticated withdrawing techniques and negativistic behaviour. We have nothing to say about how such a lack of shared relevance could have arisen. We think we have learned a lot about how to reduce this and about the beneficial consequences which flow from such a reduction. One little experience of mutual understanding can produce a very big effect.

The striking feature about our structured and open-ended environment, is the speed with which its effects were observed, and the apparent total personal involvement which David displayed when he interacted with it. He rapidly built up action sequences and descriptions through reference to his own body schemata, which we assume have developed very early in life. Acting out behaviour has been reported by therapists such as Bettleheim (1967), who concurs with Piaget in his view that any learning in early developmental stages has to come through bodily experience. David's acting out behaviour is reminiscent of Piaget's sixth stage of sensori-motor thinking (Flavell 1963). David exploits the subtle relationship between himself as agent in control of the turtle as instrument, and the turtle in turn as indirect agent, when for example he manoeuvres it to knock down an object in its path; and he elaborates to make his hand an agent controlling himself. The relevant features of the situation are isolated, i.e. the relation between the turtle movements and the buttons is transparently obvious, and thus readily accessible to him. Since we share this with him, the possibility of communication is heightened.

As soon as this shared sense of relevance was lost we observed David switch into his "passive-pupil" role. The fact that he could change roles like this showed his ability to perceive and react to subtle interpersonal situational stimuli in a way which the "perceptual deficit" theory of autism (Wing 1966) does not seem to recognise (perceive). An example of this role switch (evocation of different model of self) was when David perceived and reacted to the similarity between our environment and the school situation, referred to above.

As far as David's social communication is concerned, we share with Bettleheim the view that

> "Autistic children come to life only when we are able to create conditions or otherwise be the catalysts that induce them to action on their own behalf ... Speech in the sense of communication simply cannot be forced

out of children. It can only be acquired as the outcome of personal relations" (1967)

In order to build up a strong personal relationship of this kind, which in Rycroft's (1968) view is the essence of the therapeutic situation, we have to convey to the child that we respect his autonomy and understand the meaning of his behaviour.

What have we gained by treating David's behaviour as a series of meaningful communications ? In the process of acting-out, David seems to be both telling himself and telling us what he has understood - his MONOLOGUE trails into DIALOGUE spontaneously. Overt non-verbal and verbal social gestures and an increasing willingness to commit himself followed from the reality of his being the free agent of his own actions and learning, and of the self validating effect of understanding and being understood. The LOGO environment served as a catalyst in developing our relationship with David precisely because he was able to actively control and understand an object of common interest, the turtle. This enabled the development of a shared mini-world view which became the basis for communication.

ACKNOWLEDGEMENTS

We are grateful to the Social Science Research Council (S.W.) and the Commonwealth Scholarship Committee (R.E.) for financial support during the period this work was carried out. We are indebted to Dr J. Howe for the existence of the LOGO laboratory, without which this work would not have been possible; and most particularly, to David's parents and to the staff of Milton House Autistic Unit for their cooperation during the study.

REFERENCES

Bettleheim,B. (1967) *The Empty Fortress -- Infantile Autism and the Birth of the Self.* New York: Free Press.

Clowes,M. (1973) Lecture Notes of A.I.S.B. Oxford Summer School.

Colby,K. & Smith (1971) Computers in the treatment of non-speaking Autistic Children. *Current Psychiatric Therapies*, Vol. *11*.

Colby,K. (1973) Rationale for Computer Based Treatment of Language Difficulties in Non-speaking Autistic Children. *Journal of Autism and Childhood Schizophrenia*, Vol. *3*, No. *3*.

Creak,M. (1961) Schizophrenic Syndrome in Childhood. *The Lancet*, October 7, 1961.

du Boulay,B. & R.Emanuel (1975) LOGO without Tears. *Working Paper No. 11*, Department of Artificial Intelligence, University of Edinburgh.

Flavell,J.H. (1963) *The Developmental Psychology of Jean Piaget* New York: Van Norstrand.

Howe,J.A.M. & T.O'Shea (1976) Computational Metaphors for Children. *D.A.I. Research Report* (in preparation), Department of Artificial Intelligence, University of Edinburgh.

Kanner,L. (1943) Autistic Disturbances of Affective Contact. In *Child Psychosis: Initial Studies and New Insights.* John Wiley & Sons 1973.

Kanner,L. (1946) Irrelevant and Metaphorical Language in Early Infantile Autism. In *Child Psychosis: Initial Studies and New Insights.* John Wiley & Sons 1973.

Laing,R.D. (1971) *Self and Others.* London: Penguin Books Ltd.

Papert,S. (1973) The Use of Technology to Enhance Education. *LOGO Memo No. 8,* U.S.A.: M.I.T. The A.I. Laboratory.

Park,C. (1968) *The Siege.* Gerrard's Cross: Colin Smithe Ltd.

Rutter,M. (1968) Concepts of Autism: A Review of Research. *Journal Child Psychology and Psychiatry,* Vol. *9*, 1-25.

Rycroft,C. (1968) *Psychoanalysis Observed.* London: Penguin Books Ltd.

Szasz,T. (1972) *The Myth of Mental Illness.* St Albans: Paladin Books.

Trevarthen,C. (1974) Conversations with a 2-month old. *New Scientist,* 2nd May 1974, 230.

Weir,S. (1975) The Perception of Motion: Actions, Motives and Feelings. *D.A.I. Research Report No. 13,* Department of Artificial Intelligence, University of Edinburgh.

Weir,S. & R.Emanuel (1976) Using LOGO to catalyse communication in an autistic child. *D.A.I. Research Report No. 15,* Department of Artificial Intelligence, University of Edinburgh.

Wing,J.K. (1966) *Early Childhood Autism: Clinical, Educational and Social Aspects.* London: Pergamon Press.

Winston,P. (1975) *The Psychology of Computer Vision.* McGraw Hill

W.T.Harwood
F.K.Hanna

A DISTRIBUTED ACTIVITY PROCESSING SYSTEM FOR A.I.

This paper is concerned with the development of a, primarily hardware, knowledge system with common sense like capabilities which eventually may prove more useful than 'logical deduction' systems. To this end the following section makes some observations about human intelligent behaviour and reformulates these in 'machine terms'. After this, two themes are developed (i) a simple set of control primitives for a distributed system, and (ii) the development of "priority" and "relevance" schemes to the above stated goal. What has been omitted from this paper is a detailed discussion of processor-processor deadly embraces and general network message saturation (although these are touched upon). This is a deliberate omission on the grounds that space precludes detailed discussion of this. The authors will be commenting else where on these aspects.

MOTIVATION

Primarily, our notion of intelligence derives from human behaviour, hence it is worthwhile to consider some informal statements about human problem solving and see what requirements this imposes on an artificial system.

The following statements are offered not because they are indisputable (if they were research would be very boring) but because they sum up problems that we are interested in.

(i) A person selects a few solution techniques that are believed to be useful for any given problem.

(ii) People reformulate and consolidate their knowledge to produce new assertions[1] about the world even when not solving specific problems.

(iii) A person is 'aware of' many aspects of the environment simultaneously and can be aware of many aspects of a problem simultaneously. In this sense a person is 'multi-theme'.

(iv) A person can recognise 'old' problems and use this to arrive at rapid solutions to them.

[1] Note that by assertion we mean any assertion or theorem in the PLANNER sense (Hewitt 1969) - since a theorem is only an assertion with a variable truth value.

(v) A person is rarely at a 'total loss' for an approach to a problem.
(vi) A person can argue by analogy.
(vii) Knowledge is stored implicitly and explicitly i.e. as motor information and as sensory information.

From the above we may form restrictions on an artificial system that is to perform well in similar areas as a person. Splitting the statements above into more useful groupings:-

(I) from i & ii we have: knowledge is in some sense a dynamic entity which is reviewed continuously (see Hanna (1975)) and is represented by both procedural and textual (i.e. descriptive or sensory) information.
(II) from iii: the system must have many parallel processes active simultaneously that are dynamically coupled.
(III) from iv, v & vi: knowledge can be recovered from memory by total, partial and semantic matching of information with memory. This in general will be called retrieval of knowledge.
(IV) from i: a problem solver must conduct a rapidly self-pruning search of possible solution start points and should follow the one that it believes most promising, i.e. the one that has the highest belief value.

In order to fulfill the above loosely stated specification the "activities system" was formulated.

ACTIVITIES SYSTEM

An "activities system" is a distributed knowledge system composed of a regular network of processing elements, called activities.

An activity is a processor with memory and software that supports a local (to the activity) data base of assertions about the "world". Activities may pass messages to their nearest neighbour in order to request or provide information. (The neighbour will pass on information to more distant activities if requested). When activities are not talking to their neighbours they consolidate their own data bases by (i) deriving new assertions from their current assertions and (ii) "noticing" general messages that carry information about network activity and updating their data bases accordingly.

The above mentioned data base is a simple entity which when given a symbol string returns a list of values for that string, these values are arbitrary symbol strings which may be constants, variables, lists, procedures, etc. The values are obtained by a fetch instruction and successive elements are obtained by next (note that the current element may be accessed by same). The exact mechanics of the operation of these primitives is irrelevant in that they could be implemented by say CONNIVER (Sussman & McDermott 1972) possibilities lists

and generators or PLASMA (Hewitt 1975) message sending techniques. The following defines the required data base operators

(a) Add to(x,y): add structure(x,y) to data base with name x and value y.
(b) Delete(x): remove structure with name x.
(c) Fetch(x): fetch values of x.
(d) Next (x): get next value on list x.
(e) Same (x): take current value from list x.

Activity Communication

As previously stated activities talk to one another; this is achieved by the passing of *programs* between activities which are evaluated by the receiver. To enable useful reply a context is given to the message which the receiver can use in making a reply. Every process of an activity is given a context name; if the process if suspended the context can be used to return it to activity. A set of control primitives for message sending and control is given below. All parameters are passed by value and the primitives QQ and UQ are used to alter this.

(f) QQ(x) & UQ(x): QQ behaves like a LISP QUOTE except that if when within a single level of QQ's a UQ is met then the argument of UQ is evaluated and inserted, viz, QQ(Add to UQ(W) S) where W=* would be assessed as QQ(Add to * S).
(g) Suspend: Suspend the current process and store in data base under the context name.
(h) Start(x):Start (or re-start) the context x.
(i) Kill(x): Throw away the process x.
(j) Send(x,y):Send message y to destination x attaching source name and context to the message (note a message may be sent to all activities by setting x=GM(general message)).
(k) Context: The current context name and a pseudo primitive.
(l) Reply(y):Get source and original context of process and reply with message y to it.

These primitives are used to supplement any suitable programing language. The system outlined so far has the basic requirements for specifications I & II and part of III; we now proceed to develop this to fulfill III and IV in the next section.

DEVELOPMENTS

Firstly we introduce a new structure to the data base of each activity, this is the activities interest list. This defines the type and degree of interest of each activity in terms of other activities (c.f. semantic nets). This is stored as a list of pairs <interest, extent> where extent takes the range $[0,1]$. This list defines the 'fuzzy set of interests' of each activity and we will introduce a new primitive matchk(p),where p is a list of interests, that test the intersection of p and

the activities list of interests. If the intersection is above an activity defined threshold the primitive has no effect, if it is below it however, the primitive kills the current process. This allows control of processes in activities by relevance. A more general version of match(p) is also needed(i.e. it does not kill).

Another pseudo primitive is introduced which incorporates a send and matchk, this is the event primitive. Its action is to carry a 'fuzzy description' of an event, (i.e. a set of interests) and to inform the event sender of those activities which match the interest. Put another way, it tells the activities which match the description that an event of interest has happened so that they can request further information. Such event messages are sent when either some activity notices a change in the 'external world' or when it derives a highly believed assertion.

Hence a mechanism exists by which activities can gain information about non-local assertions of the network and so dynamically assess the relevance of its local assertions to the network requirements.

SELF PRUNING AND SATURATION

Since many messages will be passing around the network some means of message priority is required in which low priority messages are suppressed or queued if when passing through an activity they have a priority lower than the current threshold at the activity. This threshold is simply a function of the amount of message passing the activity is doing at that moment.

Self-pruning is implemented by using an approach similar to that used in FUZZY (Le Faivre 1974) and FUZZY PLANNER (Klung 1974) viz, by incorporating a-priori "belief values" and "resultant (calculated) belief values" in assertions and by ceaseing a line of 'proof' when the "resultant belief value" falls below a given level.

Hence activities may be allowed to invoke many activities in parallel and of these only those that are "highly believed" will continue and propagate to more activities. It is therefore obvious that some activities will receive messages simultaneously and hence a priority scheme is necessary. The activity queues await messages and select the highest priority messages for attention first. A good choice of priority is the product of the "belief value" of the message (as supplied by the last activity) and the relevance of the message defined by the value of the fuzzy intersection computed by matchk for event messages and unity for direct messages. It is now possible to define the message priority mentioned above as a function of the sending activities priority or "belief in its current process" hence:-

Priority = relevance x belief
Message Priority = f(Priority source)

The belief, or priority of source, is defined as the "resultant belief value" which starts of at a value given at process initialisation and is recalculated as new assertions are introduced from the data base.

To allow selection of useful and relevant assertions from the data base of an activity during problem solving we use 4 values attached to a record[1]:-

(a) A-priori belief - a dynamically assessed relevance value.
(b) Resultant belief - the belief value used in calculating the effect of incorporating the assertion in the process : this may be a constant or a procedure.
(c) Generality value - how many things can the assertion be expected to apply to - it is a function of the constraints on variables in assertions.
(d) Work value - the amount of effort involved in using an assertion.

(N.B. all values in range $[0,1]$).

During consolidation activities select highly "believed" low "generality" and low "work" assertions and apply them to derive new assertions about the world. In many ways this is analogous to the activities selecting those assertions that they "wish" to believe like PLANNER "thante" while regarding the rest as "thconse".

The simplest method of choice is to take the product of "belief", (1-"generality") and (1-"work") and apply an assertion if this value is above same threshold set for the activity. If we make message sending take a large amount of "work" then this will effectively limit the number of activities that will request external data from other activities. (Here a choice of a threshold level dependant on regional message passing is sensible).

If, however, there are activities in the system such that most of their assertions are message sending, then some of these will become active. When such an activity has a high initial "priority" then messages from it can spread over the system constituting a "theme" of computation in a manner independant of the external environment (in the sense that it is not necessary for the environment to trigger it).

Hence, such an activity could be concerned with some aspect of, say, danger to the system and it would try to seek out from its global knowledge possible changes and act before they became actualities. In any system a large number of "themes" could co-exist, some inate, and some stimulated into existance by the environment.

[1] This is in some ways similar to the work of Becher (1973).

Suppose now that there is an input to the system and the input processing activities send out event messages. These messages, if sufficiently important to travel across the system, are fuzzily matched with other activities. The matched activities reply in order to get the detailed 'program' for action. Many different activities may get the same or similar programs (problems) and produce different types of answers from them. Hence different activities can be regarded as considering different aspects of the same problem.

Although in this example the cause was taken as external it is equally possible for it to be internal, that is, due to an 'intrinsically active' activity that contains a highly (a-priori) believed assertion of the form 'for any X if X is "highly believed" send an event message'.

DEADLOCKS

Deadlocks are of 2 types:-

(A) message protocol level deadlocks which is the 'normal' set of deadlock problems encountered in 'packet switched networks' and therefore not dealt with here except to say that under the restrictions of a regular network these problems are easily surmountable since (a) if activities can communicate between processors in a 'cell' of a regular network preventing deadlocks then (b) activities can communicate between cells preventing deadlock and hence communicate to any point in the network.

(B) recursive-activity processes. In which an activity sends a message directly or indirectly to itself for assessment. This is resolved by the fact that each 'assertion' application to a process reduces the 'belief in' the process and so a recursive process either terminates naturally or the activity handling it gives up and throws away the process due to lack of belief! This also takes place for co-recursive activities.

CONCLUSION

It is believed that the system outlined here is capable of "intelligent behaviour" as defined in this paper, and at present an attempt is being made to incorporate part of it into an actual system.

The activities scheme shows a basically different approach to the problem of knowledge representation in-so-much-as it deals with a different dichotomy of representation. Many distinctions have been made (e.g. intrinsic/extrinsic, procedural/predicate calculus): this paper presents another, which is perhaps more fundamental, the "active"/"passive" distinction. It is argued that the activities system is a natural and necessary extension of procedural knowledge systems (Winograd

1972) and of Hewitt's Actor systems (Hewitt 1973). Some may argue that the distinction between "active" and "passive" systems is slight and that anything that can be done "actively" can also be done "passively". This can only be answered in the manner which Hewitt meets his critics - such equivalence is trivial - it is a matter of the ease of implementation of an idea, some fit well within a "passive" frame, others not.

REFERENCES

Becker,J.D. (1973) A model for the encoding of experimental information, in Computer Models of Thought and Language, Shank & Colby (ed.).

Hanna,F.K. (1975) An active distributed system for representing and manipulating real world knowledge. (Available from Cybernetics Research Group, Electronics Laboratory, Univ. of Kent).

Hewitt,C. (1969) PLANNER: A language for proving theorems in Robots, in Proc. IJCAI 1, Mitre Corp.

Hewitt,C. (1973) A universal modular actor formalism for artificial intelligence. Proc. IJCAI 3, Stanford, U.S.A.

Hewitt,C. (1975) A PLASMA primer.

Klung,R. (1974) FUZZY PLANNER: Reasoning with inexact concepts in a procedural problem-solving language, in J.Cybernetics 4, 2, pp.105-122.

LeFaivre,R.A. (1974) The representation of Fuzzy knowledge. (Ibid) pp.57-66.

Negoita,C.V. (1973) On the notion of relevance in information retrieval, in Kybernetes, Vol.2, pp.161-165.

Winograd,T. (1972) Understanding natural language. Academic Press.

P. J. Hayes
M. A. Rosner

ULLY: A PROGRAM FOR HANDLING CONVERSATIONS

What is it that happens when we get into conversation with someone? And how did we get there in the first place? And how do we get out of it when we've decided we've had enough? And how do we change the subject? And why do we sometimes feel that people are breaking the rules and making us annoyed?

The intrinsic interest of these and many other related questions about conversation, plus the fact that they all seem to be concerned with questions of the form 'How do people achieve such and such a goal?' or 'Why do people feel X in situation Y?' suggested that they could form the basis of a good AI project whose central concern was with the pragmatics of conversation qua conversation (and not as a vehicle though which some other abilities of a program should be displayed, as in Winograd or Colby). This position paper is a short introduction to our ideas about how such a program might be written.

Others have considered similar problems (Powers 1974, Bruce 1975), and we have learned a lot from their work.

WHAT WE EXPECT THE PROGRAM TO DO

The general idea is to limit ourselves to 2 party conversations between the program and the user displayed over terminals (see below). The conversational setting will be a fictitious cocktail party at which both the user and the program are 'guests'. Initially the program will have built into it a certain amount of knowledge about the other guests present, such as their names, what they do, how they are related to each other (e.g. X detests Y). Its knowledge of the user will be (to start with) an empty skeleton upon which more facts about him can gradually be hung as Ully* elicits more information. At the end of a session, therefore, Ully will have a more refined idea of what the user is like. It would clearly be desirable to arrange for the storage of such personal information gained to be handled in a sensible way, e.g. in the form of a 'previous occasions' data base. This, like the

*The program will be called Ully, for reasons that are too complicated to explain here.

initial model of other guests, could form part of the background to further conversations. The building up of conversationally relevant descriptions of people will therefore be an important facet of this project, although to date most of our efforts have been concerned with the way the handling of conversation itself is likely to proceed.

We want Ully to be a friendly and reasonably co-operative conversational partner, who is politely nosey. We can sum up his desirable properties as follows.

Ully should:

(1) Try to find out as much as possible about the other and the cocktail party 'world' i.e. the other people in it. This seems to entail that he has an idea of what sorts of thing it is appropriate to ask about.
(2) Be polite (unless provoked).
(3) Try to increase the level of intimacy (but consistently with (2)).
(4) Be prepared to respond appropriately to the initiative of the other e.g. changes of subject, possible 'closings' (Sacks and Schegloff (1973)) etc., and to take the initiative if the occasion demands it.
(5) Try to keep the conversation at a level which it understands.
(6) Be always capable of producing some fairly appropriate response, even if it is unable to understand something said to it. It should, for example, be prepared to 'bluff its way through' if this shows a reasonable chance of paying off (it won't always, of course).
(7) Treat assertions by the other with 'care'. For example, where an assertion by the other has been rejected, further assertions by him may simply not be believed, (more about this mechanism below).

Of course, merely stating these maxims gives no clue as to how they might be achieved. We hope that the discussion in the remainder of this paper will help to define a framework in which their sense could be expressed in a reasonably precise way.

A possible conversation with Ully might begin like this:

P1: Hello how are you?
U1: I'm fine, but er, who are you?
P2: You mean you don't remember me?
U2: REMEMBER YOU? I can't see you.
P3: Of course, sorry. It's Pat.
U3: Pat! Hey how's Jackie?

Figure 1

This conversation is an artifact, though we would naturally prefer to use genuine transcriptions. To this end we are implementing a pilot system for obtaining real data from terminals. Given that figure 1 does display some conversational structure, however, our analysis would proceed in the following manner:

P1: Hello, how are you?

'Hello' greets: Roughly 'I am here. I want to talk to you'.

'How are you?': Taken literally, an adequate answer to this question could be extended. However, we feel that a more likely interpretation is that this is a peice of institutionalised politeness, the full version of which is - How are you? - Fine. How are you? - Fine. Rituals such as these seem to have an important social function (Goffman), and their significance will be discussed in greater detail below.

A plausible constraint on the use of this ritual is that the participants already know each other, or at least have met before (not too recently, however). This is a piece of evidence which entitles Ully, hearing 'how are you', to infer that the other was talking as though he already knew him. But clearly to discover whether this claim to familiarity is warranted, Ully needs to know who the other is.

U1: I'm fine, but er, who are you?

Politeness calls for the completion of the first half of the ritual (I'm fine), just as it structures the remainder of the utterance: Ully has a problem - he doesn't know who P is, yet by asking his name and exposing his ignorance he runs the risk of insult, because generally it is a blunder to forget someone's name. 'But er', then, signals hesitation in the face of a potentially embarrassing question (an example of a conversational 'tactic').

At this point P has been asked a direct question which at face value requires a factual answer containing the vital item of personal information. However, P knows that he has met U before, and he therefore also knows that he has a right to expect U to remember his name. P's last utterance conflicts with this expectation (despite the attempt at politeness), and

P1: You mean you don't remember? expresses a slight indignation by 'blaming' Ully for demanding information he ought to know i.e. he's not saying 'you don't remember me', but (1) you ask who I am, (2) I know we've met before - I'm surprised at your action which can only be explained if I assume you've forgotten me, failing on your part. This interpretation is in part aided by

preposing 'you mean', and the 'rising intonation' conveyed by the '?' (another 'tactic', see below).

Ully's next utterance

U2: REMEMBER YOU? I can't see you

has to be perceived as an adequate answer to P's 'objection', i.e. blame for not remembering P's name. Yet as far as Ully is concerned, its reason for asking U1 still holds, so that P's objection must seem unjustified. Some defence is therefore called for. The first move is to question the 'topical centre' of the previous utterance

\- REMEMBER YOU?

Standing alone, this would normally be construed by the hearer as a request for further justification (see under tactics below), but when immediately followed with another utterance by the same speaker, this second utterance tends to be perceived as a reason for the first. And it does not seem too implausible to suggest that the program's understanding of the relationships between the concepts of seeing, remembering (in the above sense), and knowing the name of should be sufficient for it to understand

not seeing P as a reason for not knowing P's name

and not knowing P's name as a reason for not 'remembering' P.

Hence, immediately following: I can't see you.

Now P, realising that his 'objection' is unwarranted, chooses to

(1) Accept Ully's last utterance as a <u>reason</u> for the inappropriateness of his last utterance

\- of course

(2) Apologise for the embarrassment caused

\- sorry

(3) Convey the vital item of information

\- it's Pat

Ully now knows who he is talking to, and is in a position to look P up (on the assumption that knowledge gained about particular people is kept). Finding an entry

- Pat!

greets by using an address term.

A plausible item of information kept about Pat could be that his wife's name is Jackie (gleaned from some previous conversation), and it is a fairly well established convention to ask after close relatives (especially if, for example, they haven't been well), so

- Hey, how's Jackie?

The hey is prefixed to signal a change of topic (though this is only one of its uses - it can also precede an unexpected (by the hearer) interruption for example).

We are aware of the shortcomings of this analysis: it assumes that many problems have been solved, such as how to identify what the current topic is, why P is prepared to 'apologise for the embarrassment caused', and how Ully comes to see his own embarrassment as a cause of P's apology. Nevertheless, we don't feel these problems are insoluble, and only by performing analyses of this sort will we come to realise the kind of reasoning ability a program like Ully will need to have.

THE MEANING OF UTTERANCES IN CONTEXT

There doesn't seem to be any direct connection between the 'syntactic form' of an utterance (i.e. declarative interrogative imperative) and its conversational function. It's no use, that is, treating a question, say, simply as a request for information. This is just too weak, because it fails to distinguish between the different purposes to which the interrogative form can be used. Questions can convey all kinds of messages: they can invite (e.g. why don't you come over tonight?), they can confer politeness on a command (e.g. could you close the door?), they can (of course) request further information about some established topic as well (e.g. what kind of a guy is he?), and change the dominance pattern after a previous question (e.g. you mean you don't remember me?).

A better way to analyse 'the question' is to get a little closer to the purposes and goals which lurk beneath - then we might be able to see more clearly the relationship between particular syntactic forms and the effective expression of a speaker's intentions. By 'effective' is intended not only that the utterance should convey the essential information, but also that it should be 'comfortable' by paying respect to those interests of the listener to which respect is due. For example, 'can you close the door?' is more polite than 'close the door' because it allows the addressee (notionally at least)

the opportunity of refusing. Like getting up when a lady comes into the room which already has some free seats, the politeness lies in the possible world where there are no seats left in the one case, or where the person has some reason for not shutting the door in the other.

There seems to be some evidence for the consideration of the other's 'rights' as an important determining factor in politeness. We stress the analogy with the distinction between public and private property in the physical sense, and the rituals which surround our social interactions with these, built up to avoid treading on each other's 'territory'. This analogy also clarifies the way in which politeness seems so intimately bound up with 'intimacy': the closer we are socially the more liberties I can take with your property without causing offence. And this notion of closeness may itself be complex, involving, for instance, how long we have known each other, our relative age, sex, and social status, the amount of personal information we have revealed to each other, and so on.

This has a direct bearing on why Ully is embarrassed in figure 1 about forgetting P's name: people who know each other's names are in some sense closer than those who don't. They are in possession of a 'gift' which each has bestowed upon the other, and forgetting someone's name is like leaving behind the hat which they've just given you for a present.

THE NOTION OF CONVERSATIONAL STATE

Being committed to the idea that understanding utterances requires an active search for explanations in terms of the belief states of the participants, we would suggest that the flow of a conversation can be fruitfully described as a sequence of 'states'. A state of the conversation is a mutually accepted set of relationships existing between the participants at a given moment, for example, whose turn it is to speak, who is dominant (e.g. if A has asked a question which B is answering, then A is dominant although B is speaking. The concept of dominance is similar in some ways to 'tempo' in chess), the level of politeness currently appropriate ('level' is, we realise, rather naive: it suggests a numerical scale; whereas politeness clearly occupies a more structural role), the degree of intimacy between the participants, the shared assumptions the participants have, and the current topic of conversation. Ully will, at any moment, have beliefs about the conversational state, expressed as assertions about these relationships.

The idea of state also suggests a starting point for a taxonomy of conversation: a 'move' (Goffman 1971) is the smallest utterance or

part of an utterance which results in a change of state.

There are various suggestions in the literature for a taxonomy of moves: some obvious categories include assertions, requests for information, agreements and disagreements, commands, invitations, thankings, acknowledgements, offers, complaints; to name but a few. (See Searle (1969) for a fuller discussion.)

A: Why don't you come over tonight?
B: Thanks, I'd like to very much.
A: O.K. then, see you about 7.30

Figure 2

Moves build into larger units (variously called 'interchange' (Goffman 1971), 'cycle (Bellack et al 1966), 'pair' (Sacks and Schegloff 1973), 'exchange' (Coulthard and Ashby 1973) in structured ways, i.e. not all sequencings are equally plausible. For example, an interchange with the structure

(invitation/acceptance/acknowledgement)

is consistent and completes one small transaction (c.f. figure 2) whereas the structure

(invitation/acknowledgement/assertion)

is less plausible, so much so, that if we transpose the third line into the 'acceptance' position:

A: Why don't you come over tonight?
B: O.K. then, see you about 7.30.

Figure 3

its perceived role changes into an 'ungracious' acceptance. The ungraciousness which comes from the lack of formal thanks, and the fact that B has taken an initiative (made a suggestion) which imposes a constraint on an invitation which 'belongs' to A, suggests either that B is rude or that the level of intimacy between the participants is sufficiently high that such lack of formality is acceptable.

CONVERSATIONAL TACTICS

Another central idea is that of a conversational 'tactic'. Tactics are recipes for achieving well defined immediate objectives in the conversation. Such objectives include, for example, alterations to the state (claiming a turn to speak, or changing the dominance pattern by asking a direct question); gaining information from the other; conveying an attitude (e.g. showing friendliness, showing

offence by becoming icily polite); avoiding answering questions etc. etc.

We have deliberately postponed the question of how tactics are to be represented inside the program. But our choice will be influenced by a desire to use the same representation both when a tactic governs the program's behaviour and when it is recognised as a pattern in the other's.

EXAMPLES OF TACTICS

(1) To relieve the effect of an embarrassing utterance

prepose (hesitation)

(2) To draw attention to a conversational initiative

prefix move with 'Look, ...'

or 'say, ...' or 'you know .....'

(3) To indicate that something said was news, and/or to request more information, say 'really?'

(4) To convey '(Do X)!', preserving politeness

where (x) is not onerous,

use (interrogative form of (Do X))

(this might be instantiated as say 'Can you (do X)?')

We would like to extend the hierarchical classification of conversational fragments in an upward direction, though it is not clear to what extent this can be done. A step in this direction concerns our conceptualisation of rituals, which could be regarded as patterns of interacting tactics which have become 'ossified' through social overuse. The rules for responding to 'How are you?' are clear: either one says 'Fine, how are you?', and the ritual continues, else one goes into some explanation of one's health, or whatnot, and the ritual is aborted.

Whether rituals as described will be of any use to a program like Ully remains at this stage an open question. The idea seems attractive (the way in which a series of moves could be specified as a sequence of functionally specified 'slots' suggests a frame-like representation). But it could turn out that trying to treat dynamic pieces of conversation as 'frozen' items will ultimately prove too inflexible.

HOW A PROGRAM MIGHT WORK

We'll assume the program has a set of (possibly changing) beliefs about the 'world', the state of conversation, the other's beliefs etc., expressed in assertions (which have the advantage of being passive to their use). Beliefs are structured by being tagged with a reason, which may be another belief or set of beliefs, an inference rule, an exterior source (e.g. the other), or simply 'this is known'. Reasons are used to push contradictions back to their sources, a process central to the program's reasoning about what is being said.

All hypotheses, and things inferred from them, are revocable in the face of contradiction. The basic perceptual paradigm we propose is that lexical 'cues' suggest hypotheses (from the lexicon), which are checked out by inferencing and seeking contradictions. The program's basic task is to find the 'least contradictory' collection of beliefs (c.f. preference semantics - Wilks (1973)), and must be organised around the idea of actively seeking weak points in the set of hypotheses with as little irrelevant inferencing as possible. Right now much of our activity is directed towards identifying ways of rapidly detecting contradiction. An important idea here is the 'attention directing' assertion, for which the failure to find a certain piece of knowledge constitutes inconsistency. Case structures serve just this role (Survey in Bruce (1975)) by identifying required pieces of knowledge.

An important aspect of our conceptual framework is that hypotheses and contradictions may involve lexical, syntactic, 'semantic', and pragmatic concepts, (again, an assertional representation will facilitate this mixing of different sorts of concept in inferencing). We expect, for example, that a considerable amount of 'grammatical' information will have to be used by Ully in understanding. But we do not believe that this information must be deployed at any specific time in the hypothesising and hypothesis-checking process. The activity of parsing should be integrated into the general hypothesis-making and consistency-checking activity of the system.

The lexicon will be one of the basic sources of hypotheses, and we follow Becker (1975) and Bolinger (1965) in believing that a large part of any reasonable lexicon will consist of phrases as well as words. Ours will also contain certain punctuation symbols, and phrases containing these, indexed hierarchically so that an entry points to all 'superphrases' containing it as an initial part (to facilitate the analysis of long phrases). We have been surprised to find that most of the tactics we have identified have direct lexical

features which can be taken as cues. We expected to have to do much more 'parsing' before being able to hypothesise tactical uses.

The program has to be able both to use conversational tactics, and perceive their use by the other. A large number of tactics will be explicitly described in the data base in terms of their purposes, 'conversational presuppositions', and realisation rules (which may be more or less specific). For example, why is it that the appropriate answer to 'Can you tell me the time?' isn't yes or no, but either the time or an apology, and why does this differ from 'Can you run a mile in four minutes?' Our explanation is that there are two hypotheses suggested by the form 'Can you (X)?'. H1, suggested by '?', is (interrogative entity requiring an answer). H2, suggested by 'Can you' is (use of tactic (4)) (see above). Each requires a different sort of explanation: H1 - why does he want to know the answer, and H2 - why does he want me to do (X). In the 'time' case, no explanation for H1 would be forthcoming, so H2 is preferred. In the other, H2 yields a contradiction (the task being onerous), so H1 is preferred. Searle (1975) puts forward a rather different analysis of this sort of example, but does not spell out a mechanism for deciding between rival hypotheses.

MEANING AND INFERENCE

It is customary to draw a sharp distinction between facts which are directly available to a parser (such as that actions have animate agents) and facts which are available only through inference (sometimes phrased as the distinction between a fact which is part of the meaning of a word and one which is a mere fact about the concept the word denotes).

We reject this distinction. It seems to us that the difference is to be analysed in terms of the greater importance, in a pragmatic sense of some facts over others in making inferences.

Even the most immediate 'case like' inference (matching an agent to an action, for example) is just that - an inference. The semantic inconsistency which leads a parser to reject the reading of 'John gave Mary the ball' in which Mary was given to the ball, (the ball is inanimate and therefore cannot be an agent) is exactly analagous to the 'pragmatic' inconsistency which causes the rejection of 'H1' in the telling the time example above.

We would maintain that this false distinction has been partly responsible for a great deal of confused discussion of the significance of 'primitives'. The idea that the meaning of a word is some conceptual structure built from some 'primitive' elements is just no use in the conversational domain. What conceptual structure

could possibly be the meaning of 'Oh ....', for example?

Generality comes, we believe, not from the identification of a few 'universal' meaning primitives, but rather from having available a lot of knowledge about a lot of subjects (concepts) and being able to use that knowledge to check consistency (as well as make inferences). And to understand 'Oh ....', some of that knowledge has to be about conversational structure and why people say the things they do.

ACKNOWLEDGEMENTS

The ideas in this paper came from many sources. We would like to thank all the members of our corridor, especially Richard Bornat, Jim Doran and Bruce Anderson for many illuminating discussions and helpful comments. The referees made helpful comments on an earlier draft.

REFERENCES

Becker, J.D. (1975) The phrasal lexicon. Proc. TINLAP Conference. Cambridge, Massachusets.

Bellack et al (1966) The language of the classroom. New York: Teachers College Press.

Bruce, B. (1975) Case systems for natural language. AI Journal, 6, 4.

Bruce, B. (1975) Generation as a social action. Proc. TINLAP Conference. Cambridge, Massachusets.

Bolinger, D (1965) The atomisation of meaning. Language, 41, 555-573.

Coulthard & Ashby (1973) Doctor-patient interviews. Working Papers in Discourse Analysis. Birmingham University.

Goffman, E (1971) Relations in public. Penguin.

Sacks, H. & Schegloff (1973) Opening up closings. Semiotica, 8.

Searle, J.R (1975) Indirect speech acts. Syntax & Semantics, Vol. 3: Academic Press.

Searle, J.R. (1969) Speech Acts. Cambridge University Press.

Wilks, Y. (1973) Reference semantics. Memo AIM-206, AI Project, Stanford University.

G.Hinton
Cognitive Studies Program
University of Sussex, Brighton

USING RELAXATION TO FIND A PUPPET

ABSTRACT

The problem of finding a puppet in a configuration of overlapping, transparent rectangles is used to show how a relaxation algorithm can extract the globally best figure from a network of conflicting local interpretations.

INTRODUCTION

The program takes as input the co-ordinates of the corners of some overlapping, transparent rectangles (See figure 1). The problem is to find the best possible instantiation of a model of a puppet. The difficulty is that if we only consider a rectangle and its overlapping neighbours, then each rectangle could be several different puppet parts or none at all, so local ambiguities have to be resolved by finding the best global interpretation. The aim of this paper is to show how a relaxation method can be used instead of the obvious search through the space of all combinations of locally possible interpretations. The relaxation method has several advantages:

1. Using parallel computation the best global interpretation can be found quickly. The time taken is not exponential in the number of local possibilities because combinations are not dealt with explicitly.

2. The computing space required increases only linearly with the number of possibilities, which makes this method better than an exhaustive, breadth-first parallel search, for which there is a combinatorial explosion in space.

3. It produces the best global interpretation, not just a good one as in heuristic search.

All these reasons make relaxation look good as a model of how the brain resolves conflicting low-level visual hypotheses. A conventional, serial A.I. search would be very slow, given the brain's sluggish hardware (Sutherland 1974).

THE PUPPET MODEL

The puppet, which is always depicted in side view, consists of fifteen rectangular parts having the following properties and

relationships:

1. Each part has a proximal end and a distal end. The proximal end is the one anatomically nearest to the top of the head. The length of a part, measured along the proximal-distal axis must be greater than its width.

2. The trunk must be wider than any of the upper limb-parts, and each of these, in turn, must be wider than its connected lower limb-part. Also, the head and trunk must be wider than the neck.

3. The head must be greater in area than the neck and the lower limb-parts must be larger than their associated hands or feet.

4. Anatomically connected parts must overlap in the right way. This is defined by specifying zones in each part and then specifying pairs of zones, one in each part, which must or must not overlap. The definition of a satisfactory joint between calf and thigh is shown in figure 2, together with some examples and near misses.

This puppet model is fairly arbitrary, but something more than simple connectivity must be used to exclude cases like figure 3. One way in which people are more flexible (as perceivers!) is that they will allow some relations or proportions to be stretched provided the rest are reasonable. The implications of this will be discussed later.

INCOMPLETE PUPPETS

The data-structure which represents the interpretation of a rectangle as a puppet part is called a percept and has slots which are filled by relations to other percepts. The relations are also represented explicitly by data-structures, which have two slots, one for each of the related percepts. When there is nothing better in the picture, people happily find incomplete puppets, i.e. ones in which some percepts have vacant slots. The program can do the same if it is given some way of evaluating incomplete puppets so that it can avoid poor global interpretations when there are better alternatives. Currently, the best puppet is defined as the one containing the most anatomical relations whilst satisfying the following constraints:

1. No rectangle can be seen as more than one part.

2. No slot can be filled by more than one relation, except for the thigh and upper-arm slots in the trunk, which can have two.

3. No type of part can be instantiated more times than it occurs in the model: e.g. there must not be more than two

thighs.

4. A relation cannot exist unless its percepts do.

This definition has its problems (see discussion section), but for many pictures it is adequate.

For pictures containing perfect puppets plus some optional extra rectangles, the task can be done fairly simply using a technique like the Waltz filter (Waltz 1972). Each rectangle is given a list of locally possible percepts and then, for example, all those potential necks which are not connected to any potential head are deleted. Iterating this process often produces a unique global interpretation and always greatly reduces the search space. Unfortunately, if a part of the puppet is missing, then the correct interpretations of the adjacent parts get deleted until eventually all percepts are wiped out. If the filtering process is weakened sufficiently to prevent incorrect deletion, it loses most of its power, and without an effective filter, the search becomes large, since the possibility that the puppet is imperfect prevents early backtracking.

GROWING A NETWORK OF LOCAL INTERPRETATIONS

Since the potential incompleteness of the puppet makes it hard to rule out any percepts on local grounds, the program uses the alternative approach of ruling them in, starting from local configurations which strongly suggest particular percepts. From these nuclei, it grows a network by attempting to fill the vacant slots with relations to already existing percepts. If this fails, and there are suitable overlapping rectangles, relations to freshly created percepts are used, and the other slots of these new percepts then act as further growing points.

Provided the best instantiation of the model contains at least one nucleus, the resulting network will contain all the required percepts. It will also contain many others and some slots will be filled by several competing relations (See figure 1b). Generally, however, a network grown in this way will be considerably smaller than one consisting of all the local possibilities.

INTERACTIONS BETWEEN PERCEPTS

Parallel processing must create a surfeit of local possibilities to be sure of generating the correct ones, so the time advantages of parallelity are lost unless there is a fast way of eliminating the rest. Simple local competition will not work because a correct percept sometimes has a locally better alternative (See figure 4), but if percepts are also allowed to help one another via their relations, then support may propagate through the network to aid a globally consistent but

locally inferior percept (See figure 4 again). A system of this type, in which global patterns emerge from local interactions, is attractive as a basis for Gestalt phenomena, but only if the system quickly reaches a stable state and there is some guarantee that the best pattern emerges.

A certain amount of mathematics is required to show how the interactions can be organised so as to satisfy these conditions. In this brief presentation, I have decided to omit proofs and the precise formalism they require, and concentrate instead on making the flavour of the ideas more easily available to a general audience. A more formal treatment of a similar, independently developed system can be found in Rosenfeld et al. (1975), though their failure to distinguish adequately between preferences and constraints (see below) makes them abandon a linear model prematurely.

PREFERENCE-CONSTRAINT NETWORKS

Finding the best puppet is equivalent to extracting from a network whose nodes are the percepts and relations the best subnet satisfying certain constraints. If the value of a subnet can be expressed as the sum of the preferences for its individual nodes, and if the constraints are equivalent to hyperplanes in the space of possible states (see below), then a relaxation method can be applied. Each node is given an associated real number between 1 and 0 called its credibility. This quantity, which should not be confused with the preference, can be interpreted as the current probability that the node is correct, i.e. part of the best consistent subnet. The constraints are expressed as linear equalities and inequalities between credibilities. For example, n or m is expressed as:

$$c(n) + c(m) \geqslant 1 \quad \text{where } c(n) \text{ is the credibility of node n.}$$

The credibilities of the nodes can be represented as the axes of a multi-dimensional space. A credibility distribution is then a point in the space, and a constraint corresponds to a hyper-plane. To satisfy an equality or inequality constraint, a point must lie on the relevant hyper-plane or on the appropriate side of it. The states which satisfy all the constraints are called legal states and the region of the space corresponding to them is a convex polyhedron, because it is the intersection of some hyper-planes (equality constraints) and some half-spaces (inequality constraints).

The value of a credibility distribution is the scalar product of the preference vector with the credibility vector. In spatial terms this means that the preferences for the individual nodes define a direction in the space of credibility distributions, and the best legal state is the one furthest in this direction. In general, this will be a vertex of the

legal region and can be found using the Simplex algorithm, a standard technique in linear programming (Pierre 1969), or by relaxation which may be better given parallel hardware.

Starting with any assignment of credibilities, each node in turn has its credibility altered so as to minimise the total amount by which the constraints involving the node are violated. It may help to think of the constraints as exerting forces proportional to the size of the violation. These forces cause the credibility to move to an equilibrium position in which they are balanced. Iterating the process for all the nodes, continually reduces the sum of all the violations until there are none left.

This process moves the credibility distribution into the legal region. In order to find the best point in this region, much weaker forces, proportional to the preferences are applied during relaxation. This allows the preferences to push the credibility distribution in the best direction, but prevents them from causing significant violations.

We have seen how the best legal state can be found. If it only contains credibilities of one or zero, then the nodes with a credibility of one definitely correspond to the best consistent subnet. If, however, intermediate credibilities occur in the best state, as they do in some as yet ill-defined circumstances, a search must be performed by fixing some nodes at one or zero, and using relaxation to find the values of the others.

APPLYING RELAXATION TO PUPPET PICTURES

A definition of the best instantiation of the puppet was given in the section on incomplete puppets. The constraints listed there can be expressed in terms of credibilities as follows:

1. For percepts corresponding to one rectangle,
$$\sum c(p) \leqslant 1$$
2. For relations competing for a slot in a percept:
$$\sum c(r) \leqslant 1 \quad \text{or} \quad \sum c(r) \leqslant 2$$
3. For percepts of a type of part that occurs n times in the model:
$$\sum c(p) \leqslant n$$
4. For a relation, r, between two percepts, p, q:
$$c(r) \leqslant c(p) \quad \text{and} \quad c(r) \leqslant c(q)$$

The preferences are zero for all percepts and positive and equal for all relations. Figures 1 and 4 show two of the pictures on which the program has been tried. So far, it has always found the best puppet, but further analysis and testing

are required. Notice how sensibly the relaxation method resolves the local ambiguities concerning the trunk in figure 1.

DISCUSSION

The task which has been used to reveal the principles of a relaxation approach, has been simplified in many ways. One easily modified feature is the lack of attention to the angles of the knee and elbow joints. A better puppet model would require that the elbows bent one way and the knees the other, and it should be possible to mobilise such knowledge to make the best puppet emerge. The theoretical interest of this type of constraint is that it is non-local, like number agreement which is problematic for context free grammars (Lyons 1968). The practical solution is to introduce global nodes representing the side of the puppet in view. These side nodes are related to each other by an exclusive-or constraint and each relevant relation is related to the compatible global side node by a material implication constraint. The best instantiation will now have compatible knees and elbows. In some cases this is too severe a restriction, since a broken elbow is better than none. So alternative weaker relations without the extra constraints are introduced. These conflict with the stronger relations, so if they have lower preferences, good elbows will beat poor ones, but poor ones will beat none.

A more serious simplification is that relations and proportions are either definitely satisfactory or definitely not. Intermediate cases could be included in the network, but there are so many of them in a complex scene that the network would become cumbersome, and a lot of computing would have to be done for very little return. The alternative is to start by creating only the good percepts and relations. As the relaxation process is applied, some percepts will emerge with high credibilities and their vacant slots can then be developed. Integrating the growing and the running of the network in this way, effectively prevents a local cue which conflicts with globally better alternatives, from initiating a search for supporting evidence. Its disadvantage is that there is no longer any guarantee that the final global interpretation is the best one, because some of the nodes of the best possible interpretation may never be added to the network.

Finally, in the simple task described, there are only two levels of structure, the puppet and the rectangles. These do not give rise to situations in which higher level knowledge is required to form the right lower level structures. Given an imperfect line drawing with occlusion, however, puppet knowledge may be required to decide which line segments form a rectangle. The obvious way of doing this is to find the best puppet composed of easily found rectangles, and then to search

for poorly depicted parts to complete it. A different and novel method is to create many potential rectangles and to set up constraints between them. Rectangles which conflict over the interpretation of a line segment, for example, cannot both depict parts of the best puppet. Instead of using relaxation immediately to find the best consistent set of rectangles, all the possibilities can be used to find potential percepts and relations. The percepts can then be linked by material implication constraints to their corresponding rectangles, thus creating a larger network containing conflicting structures at several levels. By running the whole lot at once, the best puppet can be found, and higher level knowledge can be made to influence the choice of rectangles.

REFERENCES

Lyons,J. (1968) Introduction to Theoretical Linguistics. Cambridge University Press.

Pierre,D.A. (1969) Optimisation Theory with Applications. John Wiley and Sons.

Rosenfeld,A., R.A. Hummel & S.W.Zucker (1975) Scene Labelling by Relaxation Operations. Technical Report TR-379, Computer Science Centre, University of Maryland, College Park.

Sutherland,N.S. (1974) Intelligent Picture Processing Mamuscript to be published in: Sutherland,N.S. (ed.) Tutorial Essays in Psychology. Larry Erlbaum Associates. Washington.

Waltz,D. (1975) Understanding Line Drawings of Scenes with Shadows, in The Psychology of Computer Vision, Winston,P.H. (ed.) McGraw Hill: New York.

ACKNOWLEDGEMENTS

I would like to thank all the members of the A.I. community at Sussex, particularly Christopher Longuet-Higgins and Aaron Sloman, for their helpful criticism and useful suggestions.

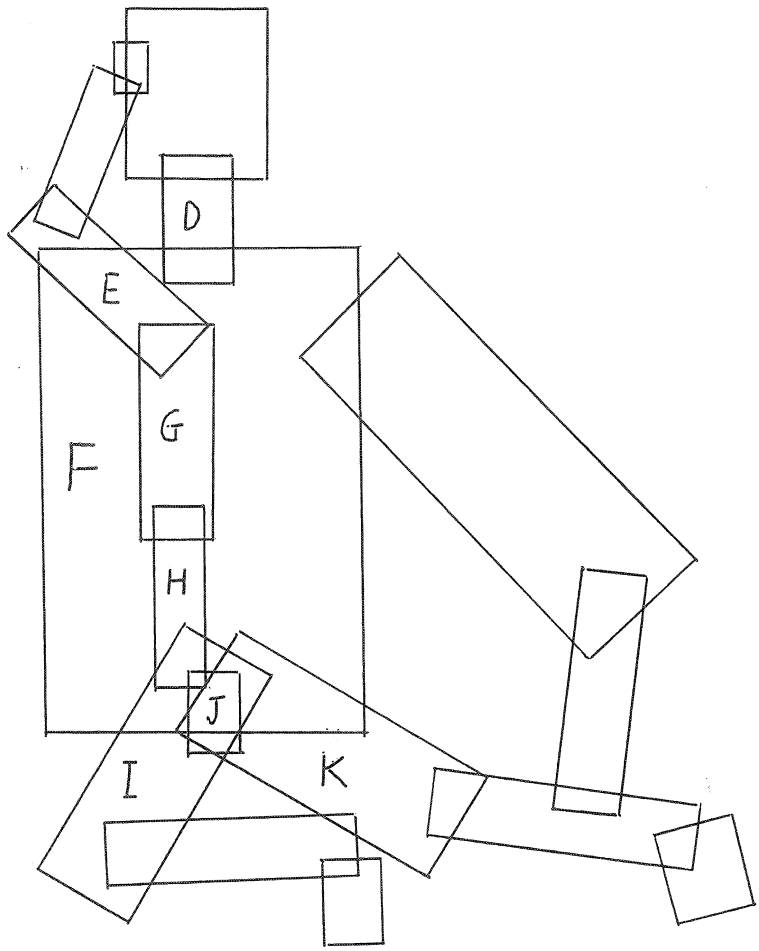


Figure 1a. A puppet with some extra rectangles.

LOCAL HYPOTHESES FOR F BEFORE RELAXATION:

F DOWN TRUNK: NECK I UPPERARM G H I J K THIGH G D E
F UP TRUNK: NECK D UPPERARM G D E THIGH G H I J K
F DOWN HEAD: NECK D

AFTER RELAXATION:

F UP TRUNK: NECK D UPPERARM G E THIGH K I

Figure 1b. The local hypotheses for a rectangle before and after relaxation. The second column indicates the direction in the picture, in which the proximal end of the percept is facing. Directly related percepts are shown after the colon.

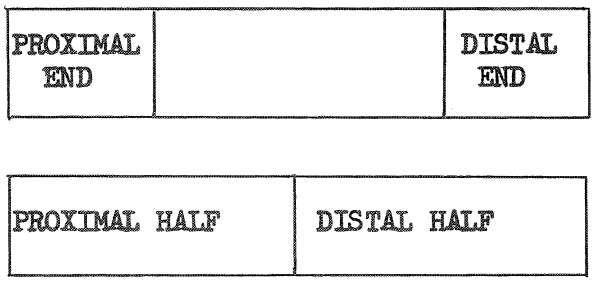


Figure 2a. Four zones of a percept.

CALF	THIGH	OVERLAP?
PROX. END	DIST. END	MUST OVERLAP
PROX. END	PROX. HALF	MUST NOT OVERLAP
DIST. HALF	DIST. END	MUST NOT OVERLAP

Figure 2b. A table showing the definition of the knee relationship in terms of zone relationships.

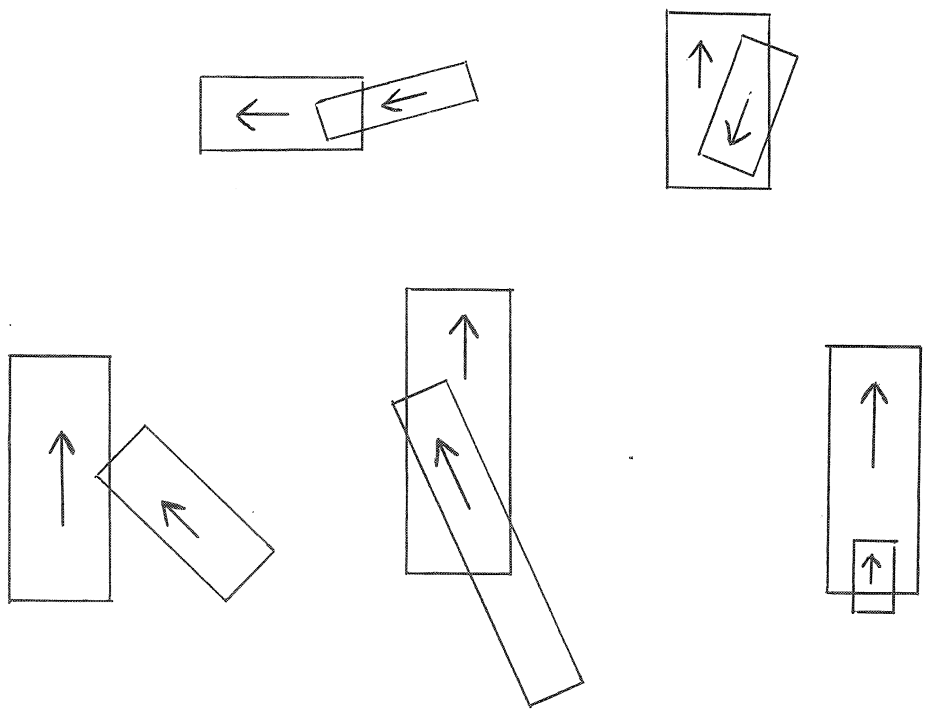


Figure 2c. Two examples of a satisfactory knee joint (top) and three near misses. An arrow indicates the distal → proximal direction. The thigh is always the wider of the two.

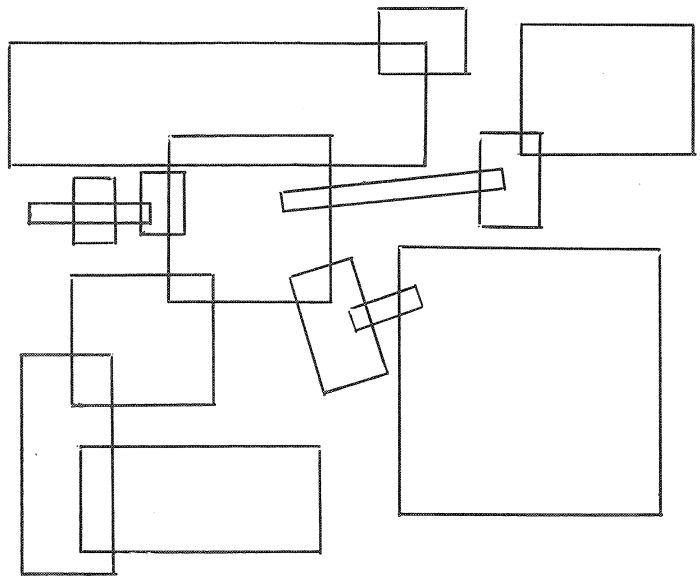


Figure 3. A configuration of rectangles with the same connectivity graph as a puppet, but with different relations and proportions.

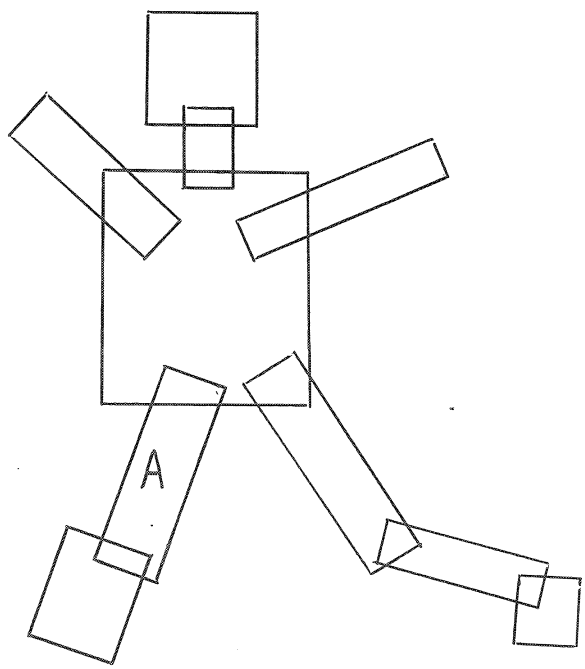


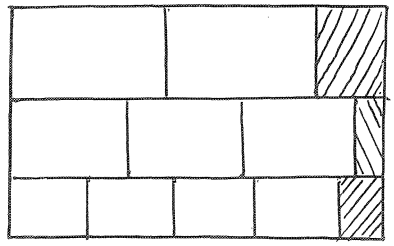
Figure 4. Relaxation picks out the interpretation of A as a thigh even though a calf is a locally better alternative.

A.I.Hinxman

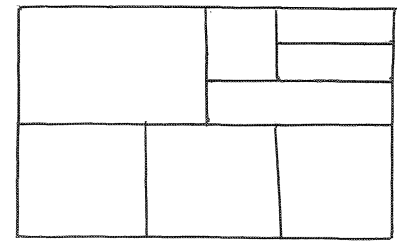
PROBLEM REDUCTION AND THE TWO-DIMENSIONAL TRIM-LOSS PROBLEM

The trim-loss, or cutting stock, problem arises wherever material manufactured continuously or in large pieces has to be cut into pieces of sizes ordered by customers. The problem is to so organize the cutting as to minimize the amount of waste (trim-loss) resulting from the cutting.

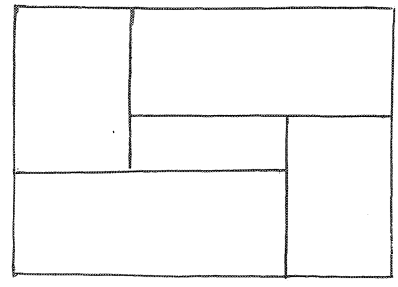
In the two-dimensional case the stock is held in large rectangular sheets (stock sheets) from which smaller order rectangles must be cut. With many materials, e.g. steel, glass, there is a restriction on the problem that all cuts shall be guillotine cuts. In other words, a cut must be a straight line from one side of the sheet to the opposite side, or a similar cut in a sub-sheet resulting from previous such cuts (see figure 1). Work has been done on this problem by Gilmore and Gomory (1965), Hahn (1969), Escudero and Garbayo (1973), and Dyson and Gregory (1974). In this work the cutting is restricted to two stages, where the sheet is cut into strips and the strips into order rectangles, or three stages, where the strips are cut into sub-strips which are then cut into order rectangles.



two-stage guillotine cutting



multi-stage guillotine cutting



non-guillotine cutting

Figure 1

All the work mentioned uses a linear programming formulation of the problem and develops a solution using dynamic programming or branch and bound methods. In the case where non-guillotine cutting is allowed, or even in the case where the number of stages of guillotine cutting is not restricted, there is not a tractable mathematical formulation.

In this paper we will examine the way in which artificial intelligence techniques can be used to produce solutions to the problem in the absence of such a formulation. The program that has been developed is essentially unrestricted in its definition of instructions as to how sheets are to be cut (cutting patterns). It therefore will solve a class of problems for which a computer solution method has not previously been available.

PROBLEM REDUCTION

Problem reduction techniques (Nilsson 1971) break a complex problem down into a number of simpler ones such that when the simpler problems have been solved a solution to the complex one will be available. The goal is the solution of the complex problem. In order that this goal may be achieved a number of subgoals of the solution of simpler problems may be set up. In more complicated cases a goal may be achieved by the solution of one of a number of combinations of subgoals.

Formulation of the problem

It is assumed that all stock sheets have the same dimensions. The order list specifies for each shape of order rectangle (piece) the number to be cut. A sub-tessellation is the juxtaposition of a number of similarly oriented identical pieces to form a rectangular area that can be included in a sheet or fragment. The set of pieces labelled T* in Figure 3(i) form a sub-tessellation.

Investigation of possible methods of solution to the problem reveals that a number of features are likely to recur in the design of cutting patterns.

(i) The different shapes of piece can be ranked as to the difficulty of designing cutting patterns that include pieces of that shape. In general, pieces that have neither side length dividing a side length of the stock sheet (non-tessellating pieces) present more difficulty than those which have one side length dividing a side length of the stock sheet (semi-tessellating pieces), which in turn present more difficulty than those that can be oriented so that both side lengths divide the corresponding stock sheet side lengths (tessellating pieces). We say that pieces with higher ranking in the order list have greater urgency.

(ii) When a sub-tessellation is cut from a stock sheet it will either consume the entire sheet, or leave an L-shaped or rectangular fragment uncut.

(iii) An L-shaped fragment can be divided into two rectangular fragments.

(iv) When a sub-tessellation is cut from a rectangular fragment, it will either consume the fragment entirely, or leave an L-shaped or rectangular fragment uncut.

Identification of these features leads to the formulation of three types of subgoal.

(i) Specify the sub-tessellation composed of pieces of the greatest urgency on the order list that is to be cut from the next stock sheet.

(ii) Specify the way in which an L-shaped fragment is to be divided into rectangular fragments.

(iii) Specify the way in which a rectangular fragment is to be divided into order pieces.

We define a partial cutting pattern to be an incomplete set of instructions as to how a sheet is to be cut, and a global step to be the design of a cutting pattern.

Shown in figure 2 is a subgoal tree for this formulation, developed to the point at which a cutting pattern as been determined.

METHOD OF SOLUTION

Subgoal Formulation

In principle, it would be possible, given an incomplete set of cutting patterns and a list of remaining order rectangles, to make the decision that some of the cutting patterns should not be used. The pieces involved in these cutting patterns would be returned to the order list and an attempt made to find a set of cutting patterns that would fulfil the resultant order list. In practice, the computational cost of this approach would be unacceptably high. In the program that has been developed, each global step leads to a simple extension of the incomplete set of cutting patterns. Once a usable cutting pattern has been designed, the maximum number of times it can be used is determined. The total numbers of pieces resulting from such cuttings are then subtracted from the order list and the resultant order list used in the next global step.

In a typical global step, we try to find a cutting pattern which includes pieces of the type with highest urgency (T*) and involves no trimloss (scrap). The generation of such a cutting pattern by problem reduction is controlled by a routine called SEARCH.

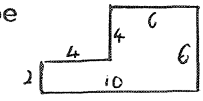
Initial order list
10 of 4 × 4
25 of 10 × 1
25 of 8 × 1
15 of 6 × 1

Sheet size 10 × 6

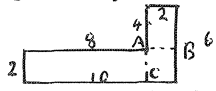
Notation: Q-definition of set of subgoals, solution of one or more of which is necessary for solution of current goal
A- alternative selected from Q of parent
S-explicit solution of subgoal

Q: Which sub-tessellation with 4 × 4 pieces should be used?

A: One 4 × 4 piece, leaving L-shape



A: Two 4 × 4 pieces, leaving L-shape



Q: Should L-shape be divided along AB or AC

A: Along AB

A: Along AC

Q: How may 10 × 2 fragment be divided and how may 4 × 2 fragment be divided?

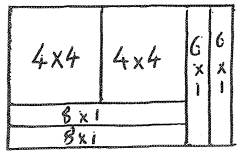
A: 10×2 fragment
S: Two 10×1 pieces

A: 4×2 fragment
S: NO SOLUTION

Q: How may 8 × 2 fragment be divided and how may 6 × 2 fragment be divided?

A: 8×2 fragment
S: Two 8×1 pieces

A: 6×2 fragment
S: Two 6×1 pieces



Resultant cutting pattern, used 5 times
leaving order list
0 of 4 × 4
25 of 10 × 1
15 of 8 × 1
5 of 6 × 1

Figure 2

SEARCH maintains a tree of subgoals. The label at each node includes:

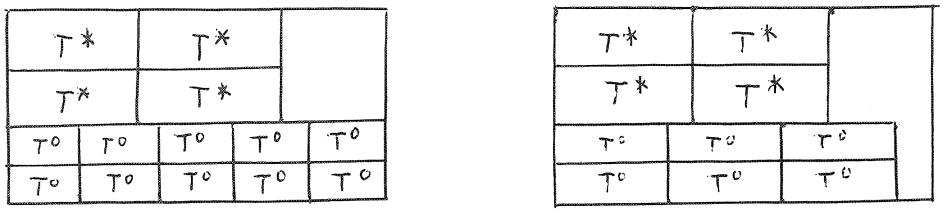
(i) the name of a function that will either solve the subgoal or set up additional subgoals,

(ii) a marker that indicates the status of the node as "active", "suspended" or "inactive",

(iii) a cost used to determine which subgoal should be considered next.

The functions whose names occur in node labels can be divided into those that specify sub-tessellations with pieces of type T* to be cut from the sheet, and those concerned with how rectangular and L-shaped fragments of a sheet can be divided into order rectangles.

If T* is non-tessellating, the first sub-tessellation looked for is one such that there is a type of piece T^o of which a number can be cut to reduce the L-shaped fragment resulting from the cutting of the T* pieces to a rectangle or a smaller L-shape (see figure 3). The reduction to a rectangle is preferred. The subsequent sequence of sub-tessellations is ordered on the number of pieces in each direction in the sub-tessellation.



(i) reduction to rectangular fragment

(ii) reduction to L-shaped fragment

Figure 3

If T* is semi-tessellating, sub-tessellations leaving a rectangular fragment uncut are preferred. If any such sub-tessellations exist in which there is a type of T^o of which a number can be cut to reduce the rectangular fragment to a smaller rectangular fragment, that in which T^o has greatest urgency is preferred. The cutting pattern when T* is tessellating is obvious, provided that the number of these pieces in the order list is not less than the number that can be cut from one sheet. Otherwise T* is treated as being semi-tessellating.

There are two ways in which an L-shaped fragment can be divided into two rectangular fragments. In each case the division of the second rectangle into order pieces is not considered until a division of the first has been found.

The division of rectangular fragments into order pieces is handled by the function RECTFILL. Each entry to RECTFILL causes the specification of a sub-tessellation to be cut from the rectangular fragment. If the dimensions of the sub-tessellation are equal to those of the fragment then a solution to the subgoal has been found. Otherwise a subgoal is set up for the division of the remaining fragment, which may be either rectangle or L-shaped, into order pieces.

Development of subgoal tree

Let us now examine the control structure used to decide which

subgoal is to be considered next. A step in the search may result in the solution of a subgoal. Suppose that the previous step did not result in such a solution. Then a search is made for the node which is active and has the lowest cost in its label. The function in this label is executed with the values of its local variables set from information in the label until one of three things happens:

(i) it finds a solution of the subgoal,

(ii) it creates a new subgoal,

(iii) it terminates having failed to do either of these things.

If it creates a new subgoal it may add to the costs of the current node and certain ancestor nodes. The cost function is used simply to guide the search into the "most promising" areas of the search tree, and is in no sense an estimate of the cost of the completed cutting pattern. The increments to node cost are closely connected with the implementation of the search. In the program that was written some of the conceptual subgoals described here were in fact implemented using several levels of subgoaling. Usually the node associated with a new subgoal will have initial cost zero and will be marked active. If case (iii) occurred, the current node is marked inactive.

If, on the other hand, the previous step did result in the solution of a subgoal, the parent node of that subgoal is considered. The function in its label is executed with its local variables set from information in the subgoal solution. The function will either report the solution of this subgoal, in which case the present node is marked inactive, or create a new subgoal.

Learning Lists

Subgoals of dividing an L-shaped or rectangular fragment of particular dimensions into order pieces may occur in several different places in the search tree. We call such a set of subgoals a shape class. Learning lists based on the shape classes are used to reduce duplication of effort in the search. Their maintenance is based on three assumptions.

(i) If several subgoals in a shape class have been set up, but none has ever been solved, then the greater the number of times they have been set up, the less the likelihood that any solution will ever be found. Therefore if a new subgoal in this shape class is set up it should be given a high initial cost.

(ii) If several methods exist for setting up further subgoals from subgoals in a shape class, then different methods should initially be used at different subgoals in this shape class, as this will avoid the parallel pursuit of solutions to a possibly insoluble problem. After all methods have been

tried on some node in the shape class, only one node in the shape class should be marked "active". The others are marked "suspended" until the first becomes inactive, and then another is marked active.

(iii) If a solution has been found to one subgoal in a shape class, all subgoals in the class should be marked "active" and the solution made available to any further subgoals in the shape class that may be set up.

If, after a certain amount of computing resources have been used, no cutting pattern has been found, the search is re-started with an allowance for scrap in the order list. If again no cutting pattern is determined, a separate routine, IRREG, based on the methods described in (Hinxman 1974) is invoked to search for patterns not restricted to guillotine cuts. If this also fails, the amount of scrap is doubled and SEARCH entered again. If it fails again, IRREG is entered and the cycle repeats.

RESULTS AND CONCLUSIONS

A program using these techniques has been written in POP-2 (Burstall, Collins and Popplestone 1971) and tested on an ICL 4130 against 16 sets of data. A typical data set is shown in table 1. The results obtained from these test runs are shown in table 2. Because the test data was generated in a random way the total area of each set of orders is not an integer multiple of the area of a sheet. Thus the number of area units of scrap, recorded in column 4 of table 2, contains a component that is inevitable. Where the amount of scrap exceeds the inevitable quantity, the excess may be recorded in the form of the number of full sheets it amounts to. This is shown in column 5.

As the type of problem we are concerned with has not previously been solved on a computer, it is only possible to comment on the execution times relative to each other. In all cases except 5, 7 and C the execution time is less than 15 minutes. The three exceptional cases indicate that a considerable saving could result from the development of heuristics to avoid abortive searches. Further effort in this direction should, however, be in terms of some practical application rather than the abstract problem.

Table 1. Typical data set (9)

597 of 8 × 8	452 of 8 × 7	136 of 9 × 3	978 of 3 × 3
265 of 10 × 6	990 of 9 × 5	941 of 7 × 4	499 of 5 × 3
395 of 4 × 3	233 of 6 × 2	873 of 9 × 1	604 of 3 × 2
375 of 3 × 1	987 of 10 × 5	42 of 5 × 4	1080 of 4 × 1
265 of 2 × 1		sheet size 20 × 20	

Table 2. Results of program test runs

Data Set	CPU time in secs	Number of sheets cut	Units of scrap	Number of full sheets waste
1	224	476	107	0
2	253	836	320	0
3	355	802	370	0
4	204	802	298	0
5	2059	1077	688	1 (= .09%)
6	286	846	37	0
7	4567	845	2369	5 (= .59%)
8	387	566	83	0
9	477	614	141	0
A	276	460	197	0
B	710	660	17	0
C	1709	737	98	0
D	533	923	120	0
E	502	472	389	0
F	785	585	96	0
G	183	608	80	0

REFERENCES

Burstall,R.M., J.S.Collins & R.J.Popplestone (1971) Programming in POP-2. Edinburgh: Edinburgh University Press.

Dyson,R.G. & A.S.Gregory (1974) The cutting stock problem in the flat glass industry. Opl.Res.Q.25, 41-53.

Escudero,L.P. & E.Garbayo(1973) The cutting stock problem: application of combinatorial techniques and mixed integer programming. VIII International Symposium on Mathematical Programming (unpublished).

Gilmore,P.C. & R.E.Gomory (1965) Multistage cutting stock problems of two and more dimensions. Opns.Res.13, 94-120.

Hahn,S.G. (1968) On the optimal cutting of defective sheets. Opns.Res.16, 1100-1113.

Hinxman,A.I. (1974) An heuristic approach to the $1\frac{1}{2}$-dimensional trim-loss problem. AISB 1974 Summer Conference (unpublished).

Nilsson,N.J. (1971) Problem Solving Methods in Artificial Intelligence, 80-155. New York: McGraw-Hill.

C.A. Kulikowski
S. Weiss, M. Trigoboff, A. Safir

CLINICAL CONSULTATION AND THE REPRESENTATION OF DISEASE PROCESSES: SOME A.I. APPROACHES

ABSTRACT

Computer consultation systems have been developed based on semantic models of the mechanisms and clinical course of treated and untreated glaucoma. A novel characteristic of these systems is that they can, for a particular case, present alternative opinions and reasoning derived from different consultants. To provide the system with a variety of opinions we have established a computer-based network of collaborating consultants who share in the development and testing of the programs. By representing in the computer detailed patterns of disease evolving with the passage of time, we are able to deal with multiple followup visits. Sequences of suggested therapies for the various types and stages of progressions of glaucoma have also been incorporated in the model.

1. INTRODUCTION

When faced with a complex case a primary physician will often refer a patient to a specialist for an alternative opinion. The specialist provides the referring practitioner with an interpretation of the case and with suggestions for treatment that draw upon up-to-date information from research and clinical practice in his specialty and related fields. With an ever-increasing rate of growth in medical knowledge, the need for expert consultant services grows apace.

Building a flexible and sophisticated computer-based expert consultation system is a formidable task because of the complexity and heterogeneity of medical knowledge and our very limited understanding of clinical reasoning processes. Historically, the first phase of development of computer consultation systems was characterized by a concentration on diagnostic reasoning alone, using homogeneous information processing structures that applied either deterministic or statistical decision rules to a single static model of the patient's condition. A comprehensive review can be found in (Patrick et al. 1974). Looking back over the past five years we can detect the evolution of a new phase, marked by the building of models of patients and diseases that combine knowledge from a variety

of sources with a diversity of structural representations, and the experimentation with an equally varied array of inferential problem solving strategies (Kulikowski and Weiss, 1971; Amarel and Kulikowski, 1972; Shortliffe et al., 1973; Pople et al., 1975; Rubin 1975; Silverman 1974; Shortliffe 1974; Miller 1975). These systems all use artificial intelligence methods in attempting to simulate the activities of an expert consultant, although they differ substantially in scope and choice of tasks and methodological approaches. The CASNET program (Kulikowski and Weiss, 1971; Weiss 1974) has been developed by us to incorporate the knowledge of a network of clinical researchers in glaucoma, a serious eye disease that can cause blindness. We have developed a causal-associational representation for evolving disease processes. It can be utilized by a variety of reasoning strategies (Kulikowski and Weiss, 1972) to provide diagnostic, prognostic, and therapeutic recommendations, together with explanations and references to diverse, alternative expert opinions. In this paper we summarize the main features of our existing CASNET system, and describe some of the methodological and practical issues that arise in developing an expert consultation system for a restricted medical specialty. We also outline a representational scheme, currently being implemented, that generalizes the semantic description of disease processes and extends the scope of the control strategies.

2. KNOWLEDGE ACQUISITION AND REPRESENTATION

It is not difficult for the designer of a computer consultation system to acquire information from medical consultants, texts and research papers. The difficulties lie in sifting through a collection of descriptive and normative statements, and in organizing and selecting those items that are necessary for problem solving. Different or additional kinds of information will often be needed to produce an explanation. In describing a given problem area, an expert will interweave general statements about the mechanisms, course, and intensity of a disease, with examples from typical individual cases. He will add statements about risk factors and population statistics when available, sometimes providing subjective estimates in their absence. Judgmental knowledge, often highly context-dependent and idiosyncratic, will be mixed with causal and empirical explanations. Having a consultant review his reasoning about a case usually provides us with only fragmentary traces of plausible, consistent strategy and logic. Except in rather clear-cut cases, intuitive leaps in clinical reasoning are only too frequent. Their ex post facto rationalization constitutes an interesting explanation, but sheds little light on how the physician

actually reasoned. In contrast, causal descriptions of disease emerge clearly and consistently from transcripts of discussions with clinicians. We concluded that it would be advantageous to develop a qualitative representation of diseases that would be natural and acceptable to physicians, together with accurate and efficient reasoning strategies, whether or not these actually simulated the physicians own.

The first problem that arose in developing a representation for diseases was how to account for the great variability in individual cases while extracting a general description of the mechanisms of dysfunction. Our initial work was deliberately restricted to a well-defined, circumscribed medical problem, that was important enough to elicit interest from clinical researchers, and would serve as a demonstration prototype for an in-depth consultant in a single specialty. In this application, cases can be seen from a single unifying context; they are submitted to the system under suspicion of having the disease in question. The purpose of the consultation is to not only ascertain whether the suspicion is well-founded but to provide a detailed interpretation of the case. Under such circumstances, multiple contexts are not essential, and we developed a uniform representation for describing the possible causes and evolutionary pathways of a disease: a causal network model. The variability of individual observations was accounted for by postulating a separate *associational model of observations*. The causal network then stands as its underlying conceptual model. The separation between the *observational* and *conceptual models* is epistemologically important, and distinguishes our approach from a previous use of causality in clinical decision-making (Bonner et al., 1964).

3. CAUSAL ASSOCIATIONAL NETWORK MODELS

The relationships between the elements of a causal associational network (CASNET) model are: *Observation-to-state mappings* - Large numbers of interdependent observations or tests of the patient contribute to the detailed description of the disease. Many of the redundancies are reduced by mapping into the causal states in the form $B(T) \xrightarrow{Q} S$, where B stands for a boolean combination of tests, S the state into which it is mapped, and Q is a confidence weight. All observations are quantized and binarized so that tests may be only true, false or undertermined. When the result of test T is obtained, the weight Q determines the confidence with which we assert the presence ($Q>0$) or absence ($Q<0$) of state S, independently of other factors. Any boolean function of tests is permissible, enabling the summarization of complex patterns of results. Logical and semantic constraints among the observations them-

selves are encoded in the structure of questions, which permits us to specify mutual exclusivity of responses, implicational and precedence relations, etc. *State network rules* - the states are linked to each other by either causal or substate rules of the form: $S1 \xrightarrow{W} S2$. The weight W is the confidence with which we expect effect S2 to follow from the occurrence of S1. Its range is from 0 to 1, but clinically we have found it sufficient to use but a few quantization levels, corresponding to the interpretations of: never, sometimes, often, usually, almost always, and always. Mutually exclusive and exhaustive causes and effects need not be specified, and transition weights are assigned independently of one another, resulting in a first order dependence network. Such a structure is attractive for several reasons: physicians naturally describe their reasoning in causal terms, the major processes of dysfunction are usually decomposable into subprocesses, and by defining sufficiently complex combination states, we can always capture higher order dependencies in the mechanisms of disease. Inverse causal weights, however, which depend strongly on prior context, cannot be assigned statically at the time the model is designed, unless we wish to investigate all likely dependencies from antecedent pathways in the network. Thus, the inverse weights are calculated by an algorithm that takes into account the configuration of confirmed, denied and undetermined nodes for a patient every time a new observation is received. Some states represent basic causes of disease, for which no antecedent causes are defined. Others, representing the furthest stages of progressions of the disease, are the terminal states for which no effects are present. It is these states that define the prior context and scope of the model, respectively. *State-classification mappings* - configurations of simple states from the causal network are mapped into complex states representing disease hypotheses by means of classification tables. These are logically ordered rules for combining patterns of confirmed and denied states that select pathways through the causal net for every primary cause that is applicable to the patient. The construction of a complete hypothesis for the patient is carried out dynamically by various strategies (Kulikowski and Weiss, 1972), and the classification tables serve only to elicit the appropriate constitutent parts, some of which may disappear in the final conclusion, having been subsumed or excluded by others.

Classes of treatments, which are represented as distinct node types, are also related to the network configuration by classification tables. Within each class there is an additional ordering by degree of intensity of the treatment. The individual characteristics of a patient are examined for

possible contraindications, and a strategy selects the minimal intensity therapy from among those belonging to the class of most effective therapies. For patients already under treatment, we must first evaluate its effectiveness. Criteria of effectiveness are expressed as patterns of desired effects, given the prior condition of the patient. For example, while we may expect to reduce intraocular pressure, we can at best hope to arrest further progression of neural ischemia and visual field loss because they are irreversible processes. If the current treatment is found to be insufficiently effective, a more intense therapy will be selected from the sequence that corresponds to that patient's condition. If, however, new information is received on side-effects of the medication; or if a revaluation of the findings results in a new diagnostic status, a different class of therapies will then be selected. Each time a follow-up visit is entered into the system, the new findings are added to a master file that summarizes all the information to date on the patient. States that describe changes in a condition are automatically deduced from observations made at different times. This permits us to characterize in detail the course of the disease, and recommend appropriately subtle variations in treatment as advised by our medical consultants.

4. SEMANTIC NETWORK MODEL

The causal-associational model that serves as the basis for our glaucoma consultation program is a special case of a semantic network. It is one in which we have emphasized the causal relationships between states to obtain a first order description of a disease. In seeking to characterize disease processes more fully we have developed an alternative, general semantic network model, which allows us to use a greater variety of relationships, such as preconditions, precursors, causes, effects, complications, successors, etc. Conditional transition rules can be explicitly stated to take into account multiple preconditions for the occurrence of a state and the dependence of the causal link on these preconditions. First order effects can be distinguished from unexpected effects (complications) and the hierarchical ordering of complex disease states is represented by a generalized set of rules (Trigoboff, 1976). Flexibility is gained by enabling the further specification of a state (by location, time characteristics, magnitude modifiers) to describe individual patient variations within the scope of a given state. Observations on a patient serve as evidence that is propagated across the various links in the semantic net. Propagation across a link is controlled by a set of rules, which specify both the preconditions and the results of pro-

pagation as detailed specifications of the appropriate nodes. Different classes of rules correspond to different relation types in the semantic net and special rules can be created for controlling an atypical propagation situation. In this manner, a variety of reasoning strategies can be pursued, ranging from highly structured and goal-oriented ones, to an almost completely uncontrolled bottom-up spread of the evidence through the semantic net. What is particularly attractive is the possibility of implementing alternative strategies for the same patient, corresponding to different ways in which experts might reason about the case.

5. GLAUCOMA CONSULTATION PROGRAMS.

Consultation programs have several different purposes when they are addressed to distinct classes of users. As a research tool for the specialist, the consultation program serves as an interpretative adjunct to the data base on specialty cases. It permits the expert to continually match his latest information from research studies against an existing model. The computer scientist with whom he works will be interested in studying the matching process so as to learn how to improve and update the model, hopefully gaining new insights into problems of representation and reasoning. As a teaching device for the resident-in-training or the physician in continuing education a consultation program can be particularly useful if it is able to provide alternative explanations from different experts about difficult clinical cases. In our programs, we provide such alternative opinions, supported by appropriate references and quotations, and by facilities to explore the logic of the program that led to the different conclusions. In an alternative mode, and dispensing with such details, the program serves as a controlling system to guide the paramedic in acquiring and storing patient data, and as a summarizing system for producing patient records.

Each of the above applications requires a different kind of input/output format, controlling strategy and vocabulary. We have developed several alternative versions of, and options within, the glaucoma consultation program to make it useful to a variety of users. For speed and efficiency our CASNET program is written in interactive FORTRAN IV, running in 35k of memory, on a PDP-10 computer under either the TOPS-10 or TENEX operating systems. The semantic net program is written in INTERLISP. The compiled code occupies about 60 pages of file space on a TENEX system. A restricted natural language interpreter is currently being developed for the semantic net program, based on an augmented transition network model(Woods, 1970). The ability to enter unstructured sets of findings, though with a few linguistic constraints, is particularly important

to the clinician.

To illustrate some capabilities of the glaucoma consultation system and its underlying CASNET model we have extracted a representative but simplified subgraph. This is shown in Figure 1, where a configuration of observations, states and status conditions are displayed.

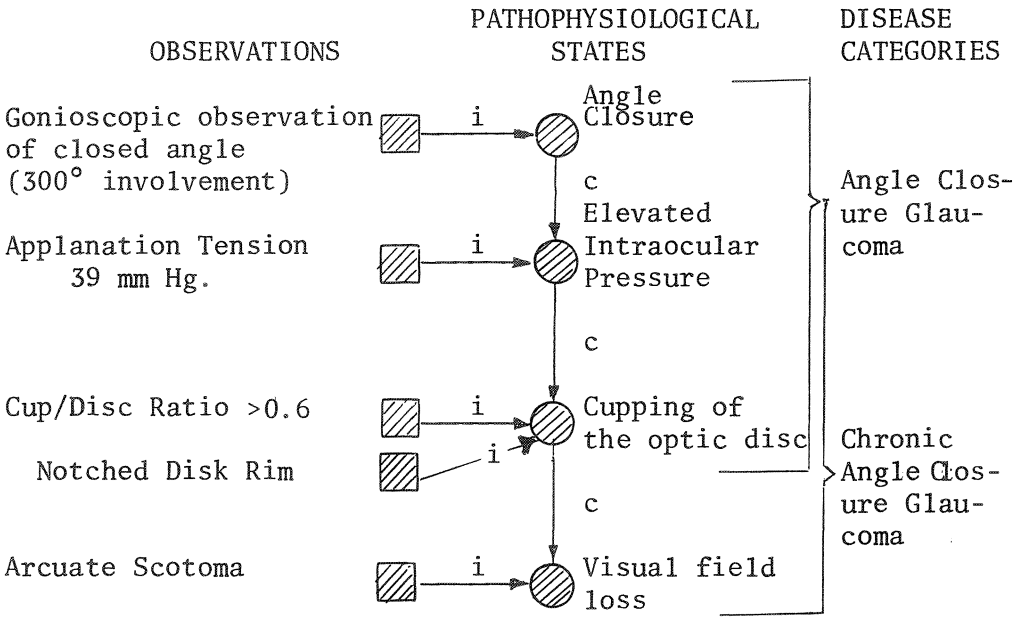


Figure 1. A Simplified Example of the CASNET Glaucoma Model Relationships: i= implies; c= causes.

Because of the continual growth and complexity of the CASNET glaucoma model, which now contains over 300 observations, about 100 states and 200 status statements (partial interpretations), validation is not a simple procedure. The program has been undergoing clinical trials since 1973. At this point the first prototype model had been developed, and was tested on about 80 cases from the records of the Mt.Sinai School of Medicine. This showed that the program was able to diagnose most primary and many secondary cases of glaucoma (Weiss 1974). It was then decided to enlist the cooperation of other clinical researchers in glaucoma to test the model and suggest improvements. The program has subsequently been tested at the five clinical centers that were referred to earlier. If the recommendations of the program differ from those of the consultant that enters a case, the relevant components of the model are evaluated for possible faults in

the logical or semantic structure. Such cases are reviewed by the other consultants, and changes in the model are agreed upon. Few important disagreements have arisen on points of diagnostic interpretation, but differences of opinion on treatment have occurred. This has prompted us to allow for alternative recommendations within the model.

The validation of the model requires information from two sources: underlying physiological knowledge and clinical trials. The results of the clinical trials are now being recorded in a data base which can be updated as the model is improved, while keeping a trace of the performance of the older versions of the program. The physiological knowledge incorporated in the causal network has been reviewed by the glaucoma consultants and agreed upon as a reasonable representation of the process of dysfunction for this disease.

In summary, our programs provide consultation for difficult clinical cases, with complex histories, over multiple follow-up visits of a patient. An important feature is their ability to present alternative opinions derived from different consultants for a given case. To obtain a variety of opinions, we have established a network of collaborating investigator-consultants who share in the development and testing of the program. The data base of glaucoma cases entered into the system provides the opportunity to study and analyze patterns of disease, which in turn leads to improvements in existing models or changes in their representation.

REFERENCES

Amarel, S. and Kulikowski, C. (1972): Medical Decision-Making and Computer Modeling, in Proc. Subconf. Comp. in Biomed., Fifth Hawaii International Conference on Systems Sciences: 173.

Bonner, R.E. Evangelisti, C.J.; Steinbeck, H.D. and Cohen, L. (1964): A Diagnostic Assistance Program, Proc. Sixth IBM Medical Symposium.

Kulikowski, C. A. and Weiss, S. (1971): Computer-Based Models of Glaucoma, Computers in Biomedicine Technical Report No. 3, Department of Computer Science, Rutgers Univ.

Kulikowski, C.A. and Weiss, S. (1972): Strategies of Data Base Utilization in Sequential Pattern Recognition, Proc. IEEE Decision and Control Conference, 11th Symposium on Adaptive Processes.

Miller, P. (1975): Strategy Selection in Medical Diagnosis, Project MAC-TR-153, M.I.T. Cambridge.

Patrick, E.A., Stelmack, F.P. Shen, L.Y.L.(1974): Review of Pattern Recognition in Medical Diagnosis and Consulting

Relative to a New System Model, IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-4, No. 1, 1-16.

Pople, H.E., Jr., Myers, J.D. and Miller, R.A. (1975): DIALOG: A Model of Diagnostic Logic for Internal Medicine. Proc. 4th IJCAI; 2: 849-855.

Rubin, A. (1975): The Role of Hypotheses in Medical Diagnosis, Proc. 4th IJCAI, 2: 856-862.

Shortliffe, E. H.; Axline, S.G., Buchanan, B.G., et. al.(1973): An Artificial Intelligence Program to Advise Physicians Regarding Antimicrobial Therapy. *Computers in Biomedical Research*, 6: 544:60.

Shortliffe, E.H. (1974): MYCIN- A Rule-based Computer Program for Advising Physicians Antimicrobial Therapy Selection, A.I. Memo 251, Computer Science Dept. Stanford Univ.

Silverman, H. (1974): A Digitalis Therapy Advisor, S.M. Thesis, Dept. of Electrical Engineering, M.I.T., Cambridge.

Trigoboff, M. (1976): Propagation of Information through a Semantic Net; Computers in Biomedicine Technical Report No.57, Department of Computer Science, Rutgers University.

Weiss, S.M. (1974): A System for Model-based Computer Aided Diagnosis and Therapy, Computers in Biomedicine Tech. Rept. No.27, Dept of Computer Science, Rutgers University.

Woods, W.A. (1970): Transition Network Grammars for Natural Language Analysis, CACM, 13: 591-606.

C.Lamontagne
Psychology Department, University of Quebec in Montreal,
Montreal, Canada

VISUAL MOTION DETECTION: A COMPUTATIONAL THEORY AND SOME OF THE PSYCHOLOGICAL DATA WHICH IT INTEGRATES

In the context of a *computational* theory of visual motion detection some apparently conflictual psychological data about *human* visual motion perception are shown to be perfectly compatible. The hypothesis is put forward that the human visual motion detection system is limited to considering only *one* visual object at a time, this visual object being however allowed to embrace any part of the observed scene, whether the chosen part consists of many physical objects, or only one, or only a small part of one.

0. INTRODUCTION

This paper is but a glimpse at a part of a doctoral dissertation (Lamontagne 1975) describing an attempt to create a general computational theory of visual motion detection. This attempt was described in the dissertation as bearing on three main issues:

1. creating computational concepts which cover the widest possible range of particular visual motion detection systems;
2. using these concepts to articulate a complete particular working system; and
3. looking for abilities of this particular system to serve as computational model of visual motion detection in particular biological systems.

As can be found in the dissertation these three issues were tackled at each of two rather different levels: firstly in the more global context of questions such as when, where, and how motion detection abilities should be used in a complete visual system, and secondly in the more local context of questions such as what the motion detection abilities themselves should consist of and how they could actually be set to work. A 'bottom up' approach was favoured to start with , contrasting with most other current efforts at modelling motion perception in computational terms (see for instance Weir (1974) and Badler (1975)). Short accounts of what was achieved in our enterprise have, until now, been exclusively devoted to results obtained at the more *local* level of analysis (Lamontagne 1973 ,1974); the present paper reports a result obtained at the

more *global* level of analysis, in the context of our search for abilities of our motion detection system to serve as computational model of *human* visual motion detection. In a first section the relevant aspects of the theory will be presented, and in a second one the psychological data accounted for will be discussed.

1. THEORETICAL CONSIDERATIONS

Our first theoretical efforts were made in the restricted context of a *monocular* visual system reacting in an 'all or none' manner to light intensities projected on to its retina from some stimulus structure consisting of bright line-drawings on dark background. Furthermore, the system's retina was assumed to consist of tightly packed receptors breaking the continuous flow of light reaching the eye into discrete units in time as well as in space. These discrete units, representing 'dots of light' in particular positions at particular moments, were acknowledged as most primitive *visual entities* in the whole visual system ('visual entity' referring to the conceptual substratum which is implied by the computing of any set of features) and were consequently labelled 'atomic visual entities' or a.v.e.'s, the implication being that *all* visual entities derived by the visual system can be traced back to some set of a.v.e.'s. In this context the task of any visual system was stated as being one of:

1. *grouping* visual entities (starting with a.v.e.'s) under well defined criteria into higher level visual entities;
2. *characterizing* the visual entities so obtained with well defined features bearing on each one of these entities *as a whole*;
3. repeating this process until visual entities are obtained which represent and qualify adequately the physical environment.

In the more particular case of visual *motion* detection, special kind of grouping involving non-contemporary visual entities, the most acute problems were found to lie in the IDENTIFICATION of corresponding visual entities as they appear through successive moments, and in the SPECIFICATION OF THE TRANSFORMATIONS which they undergo from moment to moment. Now since both processes have to be based on an analysis of the current states *of the characterizing features attributed to the visual entities with which they are concerned* (Identification looks specifically at features which are 'transformation-free', e.g. shape attributes like 'squareness' or 'concavity', while Transformation Specification looks at features which can take at every moment one of a number of possible

values, e.g. position, orientation, speed, acceleration...), and since the range of characterizing features which can be computed for any given visual entity depends essentially on *the level* of grouping at which the visual entity is created, it is crucial to realise that the computational strategies required to achieve Identification and those required to achieve Transformation Specification *will vary with the level of grouping at which one decides to apply them.* It is indeed the case for instance that at the level of a.v.e.'s, since the only possible (non-trivial) characterizing features are 'position on the retina' and 'moment in the processing', the Identification of corresponding a.v.e.'s through time (in cases where many a.v.e.'s are involved at each moment) is often a much more hazardous enterprise (bound to fail in many cases in fact) than the Identification of corresponding visual entities at a level where a.v.e.'s have been grouped into wholes which correspond to physical entities in the observed scene and have been characterized with such features as 'table-like', 'triangular', 'tree-like', etc..; on the other hand, the grouping of a.v.e.'s into wholes corresponding to physical entities calls generally for highly complex and time consuming processes, as can be gathered from the efforts made in this direction by research workers in the field of Artificial Intelligence (see for instance Winston, 1975). The problem of choosing a good level of grouping at which to compute motion turned out to be a most fundamental issue. A detailed analysis of the problem revealed many important and unforeseen computational constraints to be coped with by anyone wanting to attack the motion detection problem with any hope of success. These constraints provided a rather elaborate theoretical context for evaluating past attempts at solving the motion detection problem as well as for choosing an optimal path through the problem.

The path which we were almost forced to follow led us to a particular motion detection system whose fundamentals are as follows:

1. the system will group a.v.e.'s right up to the level of a *single* global visual entity, or *visual object*, before motion is computed, this visual object embracing either many physical objects in the observed scene, or only one, or only a part of one (i.e. no correspondence is *required* between the visual object and any single physical object in the observed scene);
2. the grouping of a.v.e.'s leading to the visual object will be carried out on the basis of low level information such as *local changes* undergone by the a.v.e.'s themselves or by other low level visual entities;

3. once the single visual object is obtained, as many (global) characterizing features will be attributed to it as required to specify its possible transformations, or motions, Identification being trivially achieved on the basis of the fact that only *one* visual object exists at a time!

Having adopted these basic computational principles we were left with the rather demanding task of providing them with precise embodiments expressed in terms of effective decision procedures capable of actually computing visual motion in the expected way. However, this level of preoccupation lies much beyond the scope of this paper and since the general computational principles listed above are all that is needed to discuss the psychological phenomena which are the main subject of this paper let us close this section and refer the unsated reader to the dissertation's detailed account of the completete work.

2. PSYCHOLOGICAL APPLICATIONS

In the context of looking for abilities of our motion detection system to serve as model of particular biological motion detection systems we hypothesized that the general computational principles listed at the end of Section 1 could account for any evidence regarding the functioning of the *human* visual motion detection system. At first, the drastic implication that no human observer should be able to see at any one moment more than *one* visual object in motion seemed much too restrictive to support the apparent multiplicity and diversity of the visual impressions constantly assailing one's consciousness. Who would indeed deny being able to see simultaneously *two* men shaking hands, or *half a dozen* cars moving at different speeds on a motorway, or *three* football players rushing towards the ball, or *hundreds* of tree leaves waving in the wind? However, a second look at the evidence sowed the seeds of the conviction that none of the events considered were totally incompatible with the 'single visual object' hypothesis, the critical point being that the single visual object can embrace *any number* of physical object or *any part* of any of them, a considerable diversity of description being available from computing motion on the basis of global features attributed to the visual object as a whole. If the second look sowed the seeds of our conviction that the hypothesis was good, the third look grew them high; new evidence had been made available, and this time it was found to be more than 'not totally incompatible' with the hypothesis: it was found to confirm it quite strongly. The new evidence came from observations made by P.Kolers (1972), and we will start by presenting these observations in their original context before interpreting them

in our own.

One of the most important questions which Kolers tries to answer in his book is whether or not the *shapes* of observed objects are taken into consideration by the human visual system in the *Identification* process on which motion detection rests. Seeking an answer on exclusively empirical grounds Kolers is finally brought to the conclusion that although shape is most of the time totally ignored it is sometimes taken into consideration; however, few clues are provided regarding possible criteria determining *when* shape is or is not taken into consideration. The conflicting observations which brought Kolers to this conclusion were made in the context of experiments which are highly similar to the following ones, which we carried out ourselves, confirming Kolers' results.

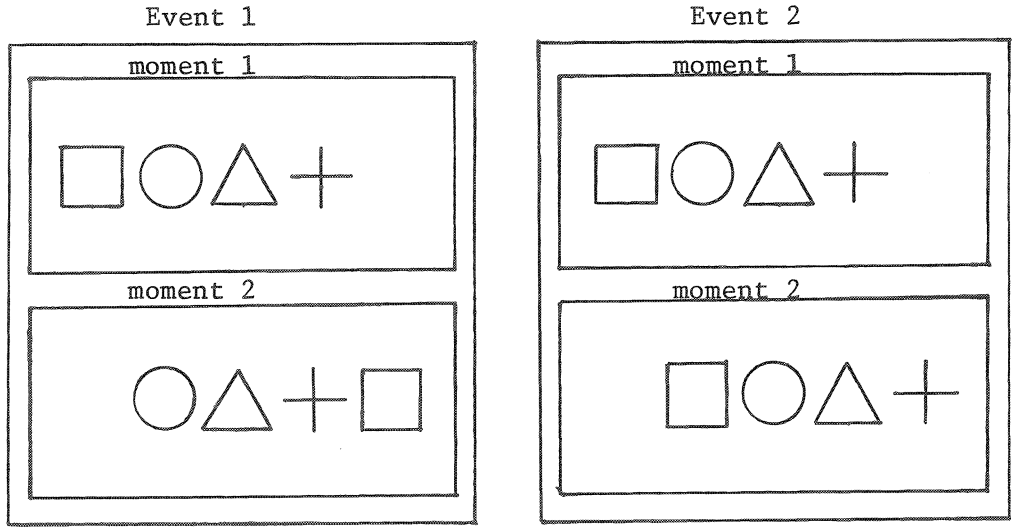


figure 1

In one experiment the two events portrayed in figure 1 were presented to the subjects. Both events imply the presentation of only two successive stimulus frames (moment 1 and moment 2), the content of the first moment's frame being the same in both cases. The critical difference between the two events is that in one case (event 1) the simplest complete account of the event is that the 'square' has moved from the left to the rignt hand side of the stimulus structure, while in the other case (event 2) the simplest complete account of the event is that all four shapes have moved slightly to the right. This is of course what the subjects reported having seen when presented with the events. Such facts were interpreted by Kolers as meaning that *shapes* do play a certain role in the Identification

process on which motion detection rests. Indeed, is it not the case that in order to overcome the 'global' similarity of the two events (i.e. a 'bunch of objects' on the left at moment 1, and a 'bunch of objects' pushed to the right at moment 2) the human visual system ought to group a.v.e.'s up to the level of the four disconnected objects, characterizing at least some of them with their shape (viz. a square, a circle, a triangle, or a cross sign), thereby providing a basis on which to establish the difference between the case where only the 'square' changes location and the case where the 'square', the 'circle', the 'triangle', and the 'cross sign' all change location?

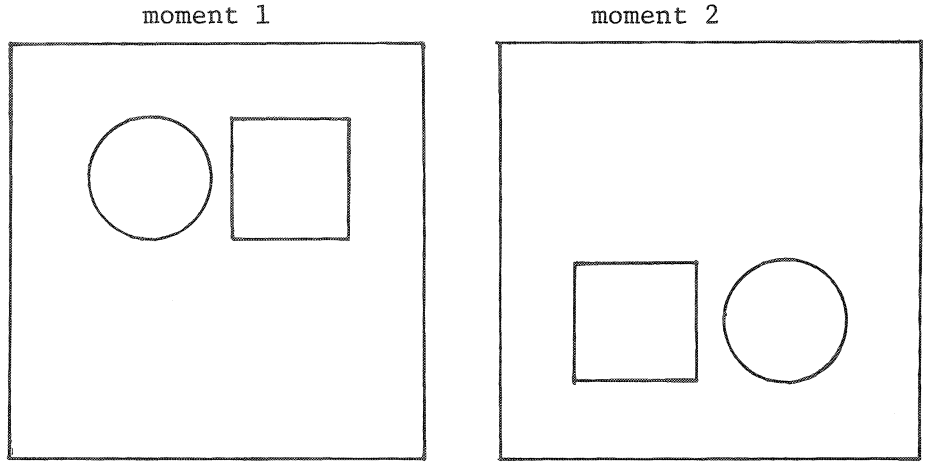


figure 2

Now in another experiment the event portrayed in figure 2 was presented to the subjects. On the basis of the result obtained in the experiment described above we could be led to expect the subjects to see the two shapes move along diagonal paths crossing each other as shown in figure 3a, both the circle and the square being successfully identified at every moment on the basis of their respective shapes. This result was never obtained; instead, the motion was always reported to be vertically downwards for the two objects in the scene (see figure 3b), however hard the subjects tried to obtain the diagonal crossing motion. This time there was no sign of Identification on the basis of shape.

If in Kolers' context such facts as those made available through the above described experiments are to some extent conflictual it is not at all the case in our context. Before discussing the facts themselves let us stress some crucial characteristics of our motion detection model:

1. only *one visual object* can be taken into account at any moment, this object embracing any part of the visual scene;
2. shapes are totally superfluous in the context of the system's Identification strategy, this one being trivially based on the sole existence of the single visual object computed at any moment.

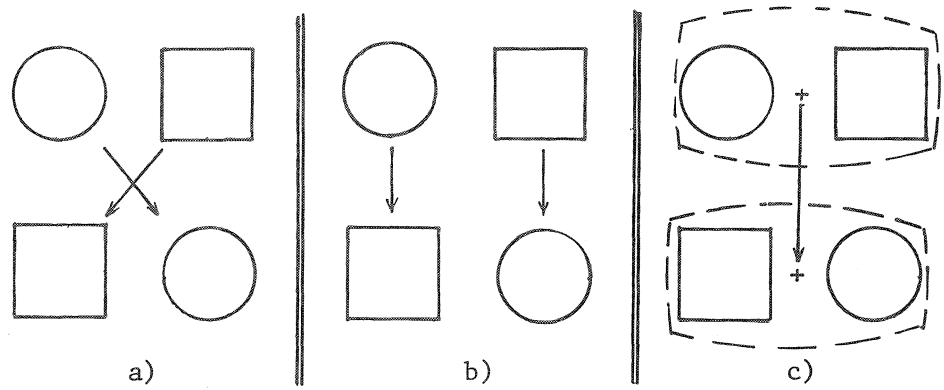


figure 3

Now concerning the facts themselves let us start by discussing the results obtained in the second experiment. Kolers reports that the subjects could never see the two objects crossing each other: it is also the case for our system for the very simple reason that it cannot see *two* visual objects moving at a time; for our system there is always only *one* visual object in the scene, and this object, in the particular situation being discussed, embraces both the square and the circle. The way in which our system deals with the scene is that since *all* a.v.e.'s undergo a 'change' from moment 1 to moment 2 all of them are thrown in the 'visual object box' where they are characterized as a whole and consequently seen as moving down as a whole. Kolers' 'two objects' can never be seen to cross each other because they are welded into a *single visual object* by our system. On this basis we maintain that what is seen by the subjects can be described more adequately as in figure 3c than as in figure 3b.

The results obtained in the other experiment, the first one presented, might seem a little more troublesome since in our context shapes are totally ignored in the Identification process. But the fact is that our system can easily do without the specification of the individual shapes used in the experiment (namely a square, a circle, a triangle, and a cross sign) and still come up with the desired interpretation of the sti-

mulus events (with in fact probally much less computation than is required by the solution involving shape specification). In the case of the first event the desired interpretation (i.e. the square alone moves from the left to the right hand side of the remaining shapes) is achieved on the basis of the fact that from the first to the second moment *only those a.v.e.'s making up the square undergo 'some change'*; this is sufficient to put them all together in the 'visual object box', thereby allowing the square to be characterized as a whole and 'seen' moving from one side of the scene to the other. Similarly, in the case of the second event the desired interpretation (i.e. the whole set of shapes is shifted slightly to the right) is achieved on the basis of the fact that from the first to the second moment *all a.v.e.'s in the scene undergo some change*; this is sufficient to put them all together in the 'visual object box' thereby allowing all four objects to be characterized *as a single whole* which, further analysed motionwise, is 'seen' as being shifted to the right. Figure 4 shows how the 'single global visual object' scheme is a winner again in the context of this experiment.

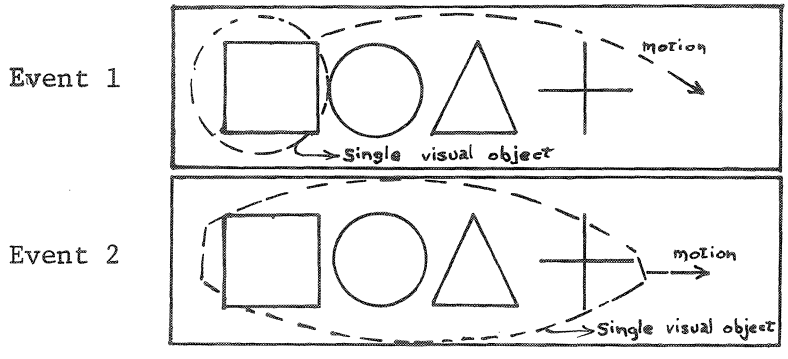


figure 4

As a conclusion, if we can say that our theoretical context provides a unifying framework for experimental results such as those discussed above we can hardly claim that the challenge of the 'single visual object' hypothesis has been totally met yet, many problems remaining to be solved at both the theoretical and the empirical levels. However, we feel that the hypothesis is now sufficiently strong to be compared advantageously to any other existing hypothesis claiming power over the motion detection issue.

REFERENCES

Badler,N.I. (1975) *Temporal Scene Analysis: Conceptual Descriptions of Object Movements*, Technical Report No. 80, Dept.

of Computer Science, University of Toronto.
Kolers,P.A. (1972) *Aspects of Motion Perception*. Pergamon Press, Oxford.
Lamontagne,C. (1973) A new experimental paradigm for the investigation of the secondary system of human visual motion perception. *Perception, 2,* 167-180.
Lamontagne,C. (1974) Defining some primitives for a computational theory of visual motion perception. *A.I.S.B. Summer Conference*. Brighton University, Brighton, U.K..
Lamontagne,C. (1975) *Steps Towards a Computational Theory of Visual Motion Detection: Designing a Working System.* Doctoral dissertation, Edinburgh University, Edinburgh, U.K..
Weir,S. (1974) Action Perception. *A.I.S.B. Summer Conference,* Brighton University, Brighton, U.K..
Winston,P., ed (1975) *The Psychology of Computer Vision.* McGraw-Hill.

ACKNOWLEDGEMENTS

I wish to express my gratitude to the many members of the Department of Artificial Intelligence of the University of Edinburgh who helped me in one way or another to complete my Ph.D. project, and especially Dr. J.A.M. Howe who supervised the enterprise.

J.A. Levin
J.A. Moore

DIALOGUE-GAMES:
A PROCESS MODEL OF NATURAL LANGUAGE INTERACTION

People engage in dialogue to achieve certain goals. The nature of this partnership in communication is such that it is effective only for certain classes of goal-pairs. For these classes, however, there have evolved a collection of conventions, which we have called Dialogue-games, appropriate to the joint pursuit of a particular pair of goals. These conventions represent shared knowledge by the participants regarding how each is to communicate to the other his/her own goals and knowledge state. A large amount of this information is not communicated in any explicit form, but is understood by virtue of the participants' recognition of the current Dialogue-game. In this paper, we define what we mean by a Dialogue-game, exhibit one in some detail, and describe processes using these Dialogue-games that model aspects of the assimilation of natural dialogue.

People use language to effect communication, which in turn serves to advance certain of their goals. In fact, the selection of the form and substance of their communicative acts is dictated by the goals in whose service the acts are to be.

If two individuals are engaged in dialogue, the goals they hope to pursue thereby must stand in certain relationships to each other. The culture has discovered that certain pairs of goals lend themselves well to joint pursuit by dialogue (e.g.: negotiating a sale & negotiating a purchase; acquiring information & disseminating information; cause subordinate to perform an action & perform whatever action is desired by superior).

The research reported here is supported by the Human Resources Research Office of the Advance Research Projects Agency, Order 2930, Contract N00014-75-C-0710. We want to thank William C Mann and James H Carlisle, the other members of our Dialogue Modeling Project at USC/Information Sciences Institute, Marina del Rey, CA, USA, for their contributions to these ideas.

Certain of these synergistic sets of goals occur with sufficient frequency that the culture has evolved shorthand linguistic forms both for signaling that the speaker wants to engage in a dialogue in service of one of these goal-pairs, and also for conducting such a dialogue, once initiated. As an example of the compression achieved by this mechanism, consider the common request:

Do you have a match?

This is customarily taken to mean something like the following:

I have a task I'd like to perform (usually lighting a cigarette) which necessarily involves my having possession of a match.
I do not have a match.
I want to have a match.
I suspect that perhaps you have a match.
I hope that you have a match.
I anticipate that you are aware that your helping me achieve a goal will make me happy and pleased with you.
I also hope that you will regard it as being in service of your own (assumed) goals of enriching your self-image as well as enhancing my image of you, to help me achieve my goals by performing, for me, an action of small cost to you.
I am asking (explicitly) if you possess a match.
I want you to give me a match.
I am expressing this desire to you.
I expect that you have the same understandings of the conventions for polite requests as do I.
Therefore I anticipate you will understand my needs and the implied request, described above.
Finally, I fully expect that, If you have a match, and if its loss is not a significant cost to you, and nothing is preventing you from doing so, you will indeed give me the match.
Alternatively, if you do not have a match, but instead have a cigarette lighter, I expect you will perceive that it will satisfy my goals and will loan it to me for a short time.

Consider another example: Were we to perceive only half of a dialogue (e.g. listening to someone conduct a telephone call) we would nonetheless be able to attribute to his/her partner, a considerable body of knowledge. This, despite the fact that we have no direct evidence of any of it.

How can we account for the surprising reliability, quality and diversity of this indirect communication? What would it take to represent a satisfactory (or useful) explanation of these phenomena?

If we were able to recognize these shorthand forms, and associate with them the rich aggregate of information they seem to entail for human speakers, we would be able to deduce a considerable body of unexpressed information about the knowledge and goals of the dialogue participants as well as the set of expectations they hold for each other. The collection of these forms and the description of the conventionally-agreed upon rules of conduct, for a particular such goal-pair is what we have called a Dialogue-game.

The problems we are addressing, then, are: 1) to discover a collection of these synergistic goal-pairs, in service of which, the language has evolved specific Dialogue-games; 2) to characterize the communicative forms used by the participants to initiate and conduct these games; 3) to determine what knowledge is shared by these participants; 4) to create a formalism in which this information can be represented; and 5) to design a collection of processes which integrate this structure with the full task of comprehending the dialogue. This paper describes the beginnings of such an effort.

The Dialogue-game construct grew out of a consideration of Wittgenstein's Language Game notion (Wittgenstein, 1958) but diverges considerably both in level of abstraction and in the inclusion of a second, goal-pursuing language user. Although the Dialogue-game construct draws on the notion of ordinary games as a metaphor, a Dialogue-game is not necessarily competitive, non-serious, conscious, or zero-sum. It was developed to handle some of the process control issues observed in naturally occurring human dialogues by the Dialogue Modeling Project at USC/Information Sciences Institute (Mann, Moore, Levin and Carlisle, 1975; Mann, 1975). We also expect Dialogue-games to provide us with a means of detecting and appropriately interpreting what Searle (1975) calls 'indirect speech acts'.

Many of the other current theoretical constructs for modeling knowledge structures also share this function of representing knowledge that is commonly shared in our society ('scripts' (Schank and Abelson, 1975); 'frames' (Charniak, 1975; Minsky, 1974); commonsense algorithms' (Rieger, 1975)). Our approach differs from these others in at least two ways: Dialogue-games seek to represent the mechanics of the dialogue

process where the emphasis of these other approaches is on capturing the structure of the subject matter being conveyed in the language. Also, other systems have not dealt with the interrelated nature of the goals jointly attempted by independent problem-solvers.

In the balance of this paper, we will first describe an informal study of a series of dialogues which led to the subsequent proposed structure of the Helping Dialogue-game. Next we outline a collection of processes which we are developing as a model of how Dialogue-games can be used in a model of dialogue comprehension. Finally we briefly sketch the possible interaction between Dialogue-games.

ANALYSIS OF HELP-SEEKING DIALOGUES

We have analyzed fourteen helping dialogues, which were naturally occurring remote interactions between TENEX computer operators and computer users, each typing on their terminals. Thus, these transcripts are automatically transcribed accurately.

The help-seeking interaction is usually initiated by some communication action associated with filling in the two participant Roles of Helpee and Helper and the Task for which help is requested. The initiator of the interaction (the computer user) either (a) inquired about the desired action or object (5 cases) (*'How do I get RUNOFF to work?'*), (b) described being in an undesirable state (4 cases) (*' Hi ... I cant get SNDMSG to work ...'*), (c) described his/her own needs (3 cases)(*'Hi - I am at [computer site name] - need help re FTP'*), or (d) inquired about the other participant's knowledge (2 cases)(*'Know anything about TELNET?'*).

The computer operator responded to this proposed initiation in three general ways: (a) explicitly refusing help, because of inability or lack of knowledge (5 times), (b) explicitly agreeing to help (sometimes with conditions) (4 times), or (c) implicitly agreeing to help by responding in a way consistent with the Helper's Role (5 times).

We can conclude from this analysis that there is no single access path used to initiate the Helping Dialogue-game. In only four of the fourteen cases did the Helpee even mention the word 'help'. Instead, the Helpee attempted to initiate the Helping Dialogue-game by communication actions that serve to establish the parameters of the Dialogue-game.

If those parameters cannot be filled (for example, when the

other person doesn't know about the Task), the proposed Helping Dialogue-game is rejected, and the Helpee has to find some other way to achieve his/her goal.

In the fourteen operator-linker helping dialogues, there seem to be two somewhat distinct stages. The first part is the diagnosis of exactly what help is needed, and the second is the treatment of the diagnosed problem.

The Helpee initially assumes an active Role, laying out for the Helper a problem description, in the following format: First, s/he describes a series of events which occurred which were normal and expected (context statements). Then, s/he describes the occurrence of some event or action which was NOT expected. This simple CONTEXT->VIOLATION sequence occurred in 5 cases, the CONTEXT->VIOLATION->MORE CONTEXT sequence occurred in 3 more cases, and compound CONTEXT->VIOLATION->MORE CONTEXT->ANOTHER VIOLATION occurred in 2 more cases. (The remaining cases were: a description of the desired end state; context description followed by description of desired end state; a context description only (after which the operator asked what the problem was)).

In this stage, the Helper (always the computer operator in these dialogues) is relatively passive, responding to the Helpee's Problem Description in one of many ways: (a) requesting more information (3 cases), (b) failing to see a problem (3 cases), (c) correcting an erroneous statement (2 cases), (d) asking about the Helpee's abilities or knowledge (2 cases), asking about the Helpee's current state or goal (2 cases).

Once the problem has been described by the Helpee, the Helper responded in two ways: communicating a series of directives for action to solve the problem (7 cases), or an apology for not being able to help, often combined with a directive to seek help from some other person (5 cases). The Helpee passively responds to the Helper in this stage, occasionally raising questions about the directives (2 cases).

Finally, the Helpee acknowledged the directives, generally by saying 'OK' (6 cases), promising that the directives would be followed (5 cases), and saying thanks (11 cases). The Helper often told the Helpee that s/he was welcome (9 cases).

FORMAL STRUCTURE OF THE HELPING DIALOGUE-GAME

Here is a formal specification of the Helping Dialogue-game, as derived from our analysis of naturally occurring helping dialogues.

In this specification, the assertion that person P has sub-goal G is represented by the structure P *wants* G.

Helping is Dialogue-game where:

Parameters are:
 Participants: Helpee and Helper
 Topic: Task
 Parameter specifications are:
  Helpee is a person;
  Helpee wants (Helpee perform Task);
  Helpee unable (Helpee perform Task);
  Helpee permitted (Helpee perform Task);
  Helpee wants (Helpee able (Helpee perform Task));
  Helpee wants (Helper enable (Helpee perform Task));
  Helper is a person;
  Helper is willing (Helper enable (Helpee perform Task));
  Helper is able (Helper enable (Helpee perform Task)).

Components are:
 Helpee wants (Helper know (Helpee perceived Action/expected-1));
  Helpee know Scene/task;
  Action/expected-1 component-of Scene/task;
 Helpee wants (Helper know (Helpee not perceived Action/expected-2));
  Action/expected-2 component-of Scene/task;

then

 Helper wants (Helpee know Action/new);
  Action/new component-of Scene/task;
  (Helpee perform Action/new) causes Action/expected-2.

DIALOGUE-GAMES PROCESSOR

This section outlines the processes necessary to use our concept of Dialogue-games in the task of assimilating the content of a dialogue. We will examine these processes in three parts: the recognition that a speaker is proposing a certain Dialogue-game, the conduct of a Dialogue-game that has been initiated, and the termination of the Dialogue-game.

Since our overall research effort has concentrated on receptive acts of language usage, the processes described here will deal with the contributions made by Dialogue-games in the comprehension of dialogue, rather than in its generation.

Recognition and Initiation

As we have seen from the analysis of helping dialogues, there is no simple 'invocation by name' of Dialogue-games. A speaker wishing to initiate a particular Dialogue-game either

asserts those aspects of his/her knowledge and goal states which qualify him/her for one of the Roles, or queries the corresponding states of his/her partner to determine the partner's suitability to fill the opposite Role. Thus, the recipient of a bid to open a Dialogue-game hears one or more utterances of the forms: 'I know ...', 'I want ...', 'Do you know ...?', 'Do you want ...?'. That is to say, s/he attempts to interpret what s/he hears as meaning either: 'I fit [Role 1].' or 'Do you fit [Role 2]?'. If the recipient succeeds at this, then by identifying either proposed Role, he will know what Dialogue-game is being initiated, since each Role is uniquely associated with a particular Dialogue-game. Thus, s/he will also know what is expected from him/her and what s/he can expect from his/her partner.

What sort of a mechanism might succeed at the recognition task just outlined? Our model, as derived from the Proteus system (Levin, 1976), represents each participant's knowledge and goal states in a semantic network Long Term Memory (LTM), where each concept is linked to all 'related' ones. Whenever one of these nodes is 'active', its 'influence' propogates through the network, along the node-links, with a decay factor related to the distance from the initial source. When a node receives influence from more than one of its neighbors, its new rate of activation is the sum of the incoming influences plus any activation it may already have.

Whenever the model receives new dialogue input, the parsing mechanism creates new structures, corresponding to the input, and bestows on these new structures a high level of activation. An attempt is made to match any new structure to nodes in LTM. For any of these matches that succeed (in the sense that '(John give Mary a book)' matches '([person1] give [person2] [object])'), a link is created from the new to the old. Thus, any input causes a wave of activation to spread through LTM. In general, each new input will initiate activation from several nodes.

These new increments of activation, combined with the current levels, will lead to certain nodes accumulating a concentration of activation. This is interpreted as an indication that these particular nodes are especially likely to be relevant to the most recent utterance.

Dialogue-games are also in LTM, having links to themselves from each of their component expressions. Certain of these expressions will be matched by structures created by the input, leading to the activity of these expressions and thus to the activity of the Dialogue-game, itself. In order to decide

what Dialogue-game(s) the other person is attempting to initiate, one (or more) of the most highly active Dialogue-games in LTM is selected as an hypothesis and an attempt is made to use it to assimilate the subsequent dialogue.

For example, The input: 'How do I get RUNOFF to work?' would lead to the activation of (among others) the following expressions:

(task is (speaker get (RUNOFF to work)))
(speaker unable (speaker perform task))
(speaker want (speaker perform task))
(speaker want (hearer enable (speaker perform task)))
RUNOFF
speaker

These last two activate the following, already present (but inactive) in LTM:

(hearer knows RUNOFF)
(hearer able(hearer enable (speaker get (RUNOFF to work))))
(hearer willing (hearer enable (speaker perform task)))
(speaker is person)

Since a number of these are specifications on the Roles within the Helping Dialogue-game, there is a concentration of activity at the node for this Dialogue-game. Clearly, other Dialogue-games will also be influenced, but the 'weight of evidence' will favor the Helping Dialogue-game.

We are not committed to making the one and only right choice at the beginning. The strategy is to consider a few of the most active Dialogue-games, attempt to fulfil all of them, and expect most to fail due to some subsequent mismatch.

Conduct and Assimilation

Once a Dialogue-game has been selected, the hearer has available a set of specific expectations for his/her partner's current cognitive state and subsequent utterances. For any Dialogue-game, the speaker may provide further expressions which augment or confirm the Role specifications. For example, the speaker may say: 'How do I delete a file in directory <XXX>?' from which the hearer might assume the speaker had the necessary authority to do this. However, it is perfectly reasonable for the speaker to continue: 'I am supposed to have delete-access to those files and ... ', confirming the expectation: '(Helpee permitted (Helpee perform Task))'.

Dialogue-games also have a set of Component subgoals, which describe the customary course of events for that particular Dialogue-game to achieve the overall goals of the participants.

In the case of the Helping Dialogue-game, the hearer (now the Helper) can expect the speaker (Helpee) to describe a sequence of perceptions and actions, which the Helpee regards as a reasonable preamble to the accomplishment of his/her desired task. This will be followed by a description of an unexpected perception, a contradiction to the Helpee's expectations. Once having reached this stage, the Helpee will expect the Helper to explain why the apparent contradiction exists. Hopefully, this explanation will leave the Helpee with a sufficient understanding to accomplish the task.

At each stage ofthe dialogue, there will be a set of expectations, some optional, some mandatory. For each of these that is satisfied, the model will expand its representation of the knowledge/goal states of the two participants, thus assimilating the dialogue.

Termination

A Dialogue-game can terminate in basically two ways: expected or unexpected. A stage in the Dialogue-game may be reached where both participants expect Termination to be the next appropriate step. This type of Termination usually corresponds to 'successful' use of the Dialogue-game, i.e. satisfaction of the subgoal which motivated the initiator to propose the Dialogue-game in the first place.

On the other hand, a Dialogue-game may terminate when of one of the required Roles becomes (or is discovered to be) falsified. For example, if the Helpee should decide that s/he is no longer interested in performing the task, s/he will not permit a Dialogue-game to continue if it is only intended to help him/her do that. In most cases, this form of Termination would represent an 'unsuccessful' attempt to use the Dialogue-game.

This use of expectations to guide the assimilation takes as its central question: Can I view the incoming utterance as a further specification of one of my expectations, and if so, how? In this light, our effort has been influenced by the work on MERLIN (Moore and Newell, 1974).

MULTIPLE DIALOGUE-GAMES

There are a number of other Dialogue-games which we have studied, though not in as much detail as the Helping Dialogue-game. There are two kinds of question Dialogue-games, Information-seeking and Information-probing, that are initiated in highly similar ways (the initiator asks a question), but for which the Role specification and Components differ (in one case, the

initiator knows the answer; in the other, s/he doesn't). Two Dialogue-games that occur regularly in our computer operator-user transcripts are the Gripe Dialogue-game and the Smalltalk Dialogue-game. Also, we have investigated a very general Dialogue-game, Polite-conversation, which contains very general rules holding over most dialogues (these rules are very similar to the 'conversational postulates' proposed by philosophers of language (Grice, 1975)).

The control structure for our Dialogue-games Processor has to be more flexible than the conventional serial hierarchical control because Dialogue-games are not strictly hierarchical in their behavior. We find Dialogue-games that are initiated, but never terminated - instead some new one is initiated and the old one just fades away. Often several Dialogue-games are simultaneously active, and the Component subgoals of all the Dialogue-games are relevant to the dialogue.

SUMMARY

Dialogue-games have been presented as conceptual structures, used to organize multiple turns of a dialogue. Each Dialogue-game has a set of partially specified Parameters, including Roles, to be filled by people with certain goals, knowledge and abilities (or lack thereof), and a Topic. Dialogue-games have Component subgoals, which are multiply active, sometimes temporally ordered production-like entities. We have discussed processes for initiating, conducting, and terminating Dialogue-games in natural language interaction.

REFERENCES

Charniak, E. (1975) Organization and inference in a frame-like system of common sense knowledge. In R. Schank & B. L. Nash-Webber (Eds.), *Theoretical issues in natural language processing*, Cambridge, MA: Bolt Beranek and Newman.

Grice, H.P. (1975) Logic and conversation. In P. Cole & J.L. Morgan (Eds.), *Syntax and Semantics*, New York: Academic Press.

Levin, J.A. (1976) Proteus: A framework for cognitive process models. Unpublished thesis, La Jolla, CA, University of California, San Diego.

Mann, W.C. (1975) Dialogue-based research in man-machine communication. (Tech. Rep. ISI/RR-75-41). Marina del Rey, CA: Information Sciences Institute.

Mann, W.C., J. A. Moore, J.A. Levin & J.H. Carlisle. (1975) Observation methods for human dialogue. (Tech. Rep. ISI/RR-75-33). Marina del Rey, CA: Information Sciences Institute. (Available only from NTIS, as AD-A013 242.)

Minsky, M. (1975) A framework for representing knowledge. In P. Winston (Ed.), *Visual information processing.* Cambridge, MA: MIT Press.

Moore, J., & A. Newell (1974) How can MERLIN understand? In L.W. Gregg (Ed.), *Knowledge and cognition.* Potomac, MD: Erlbaum Associates.

Rieger, C. (1975) The commonsense algorithm as a basis for computer models of human memory, inference, belief and contextual language comprehension. In R. Schank & B.L. Nash-Webber (Eds.), *Theoretical issues in natural language processing,* Cambridge, MA: Bolt Beranek and Newman.

Schank, R.C., & R. P. Abelson (1975) Scripts, plans, and knowledge. In *Advanced Papers of the Fourth International Joint Conference on Artificial Intelligence.* Cambridge, MA: MIT AI Laboratory.

Searle, J.R. (1975) Indirect speech acts. In P. Cole & J. L. Morgan (Eds.), *Syntax and Semantics,* New York: Academic Press.

Wittgenstein, L. (1958) Philosophical investigations. (3rd ed.), New York: Macmillan.

Frank O'Gorman
Cognitive Studies Programme,
School of Social Sciences
University of Sussex.

EDGE DETECTION USING WALSH FUNCTIONS

Many of the digitised pictures used in scene analysis are composed, to a first approximation, of regions of uniform intensity, and an attempt is often made to find the edges separating these regions. To my knowledge, all techniques for finding edges use some form of operator which, when applied to a small region ("window") of the picture, will give an indication of whether or not an edge passes through the window. The exact form of the operator, however, depends on the technique being used. For example, if edges are found by tracking, then the operator is used to test whether an edge which has already been partially found can be extended, and the fact that some characteristics of the edge, particularly its orientation, are thus known can be exploited in the design of the operator (e.g. Shirai 1973). In another class of techniques the operator is applied to the whole picture in a raster scan, and the "feature points" found in this scan are subsequently grouped into edges. In this case no prior information about the edges is available, so the operator must be capable of detecting edges of any orientation. In this paper I describe how efficient operators for this latter class of techniques can be designed.

Let us look first at some further requirements that such an operator must meet. Since the operator is applied to the whole picture (typically 50,000 intensity samples) it must be very fast. Other requirements arise from the criteria used to group feature points. The earliest edge finders (e.g. Roberts 1965) used only proximity and (for straight edges) collinearity. Since these are predicates on the positions of feature points they pose no problems for the operator. More recent edge finders, however, have exploited additional constraints such as consistency of the intensity step along an edge (Binford & Horn 1971), and consistency of orientation along a straight edge (O'Gorman & Clowes 1973). For these to be used the operator must obviously supply estimates for the relevant properties.

Since some form of gradient operator is commonly used it is probably worthwhile looking at some of the inadequacies of this kind of operator. The version used in our edge finder (O'Gorman & Clowes 1973) is fairly typical. It involved

fitting a plane to the intensity samples of a 3 x 3 window, and taking the gradient of this plane as an estimate of the intensity gradient. Feature points were selected by thresholding the magnitude of the gradient. Figure 2 shows the feature points obtained by applying this operator to figure 1. The broad bands, which are due partly to the size of the window, represent a significant loss of resolution and also place a heavy burden on the edge finder which has to group the feature points. The window size cannot be reduced because of the presence of high frequency noise. Nor does raising the threshold solve the problem; it simply results in the loss of several edges. The difficulty is that the gradient magnitude depends on several factors, including the intensity step across the edge and how blurred the edge is. What is really required is some measure of "edgeness", and the gradient is only a crude approximation to this.

A more subtle problem with the gradient operator concerns the estimate it gives for the direction of the gradient, which one might expect to be normal to the edge. The operator was designed on the tacit assumption that the gradient is roughly constant within the window, but this assumption is clearly violated in the vicinity of edges and this produces a systematic error in the direction. To illustrate this, consider an ideal step edge passing through the centre of a square window and having a normal with direction θ. The direction θ' of the gradient of the plane which best approximates (in a least-mean-square sense) this edge in the window is given by:

$$\tan\theta' = \begin{cases} \dfrac{2\tan\theta}{3 - \tan^2\theta} & \text{if } |\tan\theta| \leqslant 1 \\ \dfrac{3\tan^2\theta - 1}{2\tan\theta} & \text{otherwise} \end{cases}$$

This gives a maximum error $(\theta - \theta')$ of approximately 6.6°. Since the gradient operator works with samples of the intensity function the actual error is much larger, and may exceed 20°, depending on the details of the digitization process.

It should be noted that other, ad hoc, variants of the gradient operator (e.g. Pingle 1969) suffer from similar problems; the only difference is that their ad hoc nature makes the defects a little more difficult to demonstrate. The operator designed by Hueckel (1971), which is not a gradient operator, does avoid these problems, but it does so at a considerable cost in speed: it is roughly 100 times slower than the 3 x 3 gradient operator. It does, however, provide a model on which to base a more efficient design, and this is what I

propose to describe, but first I shall explain in general terms the method used by Hueckel.

The method can be thought of as a form of template matching, in that an ideal edge (the template) is compared with the picture function in the window, and an edge is declared to be present if the difference is sufficiently small. But we have to be able to handle a range of edges with varying orientations, intensity steps, etc. and the template is therefore parametrized appropriately. We now have the problem of finding values for these parameters such that the difference between the template and the picture function is minimized. Although a direct solution of this minimization problem is possible in simple cases (e.g. our gradient operator was designed in this way, using a parametrized plane as the template), it is intractable in general. Hueckel was, however, able to simplify the problem by approximating both the template and the picture function with truncated orthogonal expansions. An orthogonal expansion of a function in a region R of the picture has the form:

$$h(x,y) = \sum_{i=0}^{\infty} a_i \, f_i(x,y) \qquad (x,y) \in R \tag{1}$$

where the f_i are an orthogonal sequence of functions. The familiar Fourier series is an example, in which the f_i are sine and cosine functions. The coefficients a_i are given by:

$$a_i = \iint_R h(x,y) \, f_i(x,y) \, dx \, dy \tag{2}$$

In the case where h is the parametrized template, the a_i would also be parametrized. For a digitized picture function p (2) assumes the discrete form:

$$A_i = \sum_{(x,y) \in R} p(x,y) \, f_i(x,y) \tag{3}$$

We can thus represent both the template and the picture function by vectors of coefficients, and the minimisation problem can be expressed as one of minimising the distance between these two vectors. Hueckel chose to use for this purpose the Euclidean distance:

$$\left[\sum_i (A_i - a_i)^2 \right]^{\frac{1}{2}} \tag{4}$$

As mentioned earlier, the series (1) is truncated for reasons of tractability. A further justification is that, for an appropriate choice of orthogonal basis, the truncation may be interpreted as a low-pass filtering operation which removes high frequency noise. It should also be noted that for a digitized (sampled) function only a finite number of terms of the series would be significant in any case. To summarise, the operator consists of two stages: firstly, a set of coefficients representing the picture function in the window is computed using (3), and, secondly, values are found for the template parameters such that the distance (4) is minimised.

We are now in a position to look at why Hueckel's operator is so slow. He managed to reduce the minimization stage to a one-dimensional search over possible orientations, so this is tolerably fast. Most of the operator's time is in fact spent computing coefficients. Hueckel chose as the orthogonal basis a polar version of the usual Fourier series, and truncated it after 8 terms. So even for the smallest window which he considered (32 intensity samples) the computation of coefficients requires over 200 multiplications and a similar number of additions. It is not feasible to use a smaller window since with this choice of basis the window should be circular; the mismatch between this assumed shape and the square digitization grid causes errors which become unacceptably large for small window sizes. In any case, Hueckel had to give less weight to intensity samples near the periphery of the window in order to minimise these errors, and this of course reduces the effective size of the window. It seems that Hueckel's choice of orthogonal basis, whilst it may have some theoretical advantages, is inappropriate for use with digitized images.

There is in fact an orthogonal set of functions which is far better suited to the task, namely the set of two-dimensional Walsh functions (Walsh 1923), the first few of which are illustrated in figure 3. They are defined on a square region, so there are no errors due to a mismatch with the digitisation grid. They take the values +1 or -1 in rectangular sub-regions which have sides with lengths $w/2^k$, where w is the length of the sides of the window and k takes non-negative integral values. Thus, if w, the size of the window, is chosen to be a power of 2, the boundaries of these sub-regions will coincide with those of the digitisation grid, and the computation of coefficients using (3) will involve multiplications by +1 and -1 only (which are not, of course, actually performed). So the only operations involved are additions and subtractions, and the number of these may be reduced substantially by factoring in a manner similar to that of the Fast Fourier Transform (Searle 1969). If a raster scan of overlapping operator applications is used, then the number of operations may be further reduced by saving partial sums

from previous applications. The overall effect is that the computation of Walsh coefficients is considerably faster than for the orthogonal basis chosen by Hueckel.

In order to illustrate the foregoing I will now describe a simple operator which was modelled on Hueckel's but which uses Walsh functions. The size of the window was chosen as 4 x 4 to meet the power-of-two requirement and to give an operator which is comparable in efficiency to the gradient operator. The class of ideal edges (i.e. the template) consists of step edges passing through the centre of the window, with orientation Θ, and with intensities b + d and b - d on either side of the edge. More formally, an ideal edge is a function of the form:

$$h(x,y) = \begin{cases} b + d & \text{if } x \cos\Theta + y \sin\Theta > 0 \\ b - d & \text{otherwise} \end{cases} \qquad (5)$$

I decided to use just the first six Walsh functions in order to obtain an analytic solution for the minimum distance, i.e. to avoid a search in the second stage of the operator. Applying formula (2) to the edge equation (5) we obtain the corresponding six coefficients for the template:

$$\begin{aligned} a_0 &= b \\ a_1 &= \frac{d}{2sc}\left[q(s + c) - q(s - c) - 2q(c) \right] \qquad (6) \\ a_2 &= \frac{d}{2sc}\left[q(s + c) + q(s - c) - 2q(s) \right] \\ a_3 &= a_4 = a_5 = 0 \end{aligned}$$

where $s = \sin\Theta$, $c = \cos\Theta$

and $$q(x) = \begin{cases} x^2 & \text{if } x \geq 0 \\ -x^2 & \text{if } x < 0 \end{cases}$$

By substituting these coefficients into the distance formula (4) and taking derivatives, we obtain the desired analytic solution for the minimum distance:

$$b = A_0$$

$$d = |A_1| + |A_2|$$

$$\tan\theta = \begin{cases} d\,\mathrm{sign}(A_1)/2A_2 & \text{if } |A_1| \geqslant |A_2| \\ 2A_1\,\mathrm{sign}(A_2)/d & \text{if } |A_1| < |A_2| \end{cases} \qquad (7)$$

The equations (7) represent the computations performed in the second stage of the operator, the A_i being the coefficients of the picture function which are computed in the first stage. In order that the operator can make a decision as to whether an edge is present we need a measure of the goodness of match. The distance function (4) is inappropriate since it depends on the intensity step d, so, following Hueckel, we use the cosine k of the angle between the vectors a_i and A_i:

$$k = \left[\frac{A_1^2 + A_2^2}{A_1^2 + A_2^2 + A_3^2 + A_4^2 + A_5^2} \right]^{\frac{1}{2}} \qquad (8)$$

For a perfect match k takes the value 1, and the operator rejects an edge if $k < 0.9$. The definition of an ideal edge (5) includes regions of uniform intensity (i.e. with $d = 0$), so to deal with this the operator also rejects an edge if $d < 1.0$. Figure 4 shows the feature points selected by the operator using these two thresholds.

The time taken to process this picture was about 10 secs (on an ICL 1906A); this roughly 1.5 times slower than our gradient operator. As can be seen from figure 4, most of the broad bands produced by the gradient operator have disappeared, and in fact the number of feature points has been more than halved compared with figure 2. The values obtained for the orientation θ are typically accurate to within $\pm 3^\circ$, and do not suffer from the systematic error noted for the gradient operator. Since edges are blurred by the digitizing equipment the values obtained for the intensity step d are generally too low, but they are typically consistent to within 10% along an edge. The values for the intensity b depend critically on whether the edge actually passes through the centre of the

window, and considered individually they are not accurate enough to be of much use. However, after an edge has been recovered the mean intensity of the feature points contributing to the edge is usually quite accurate, and could be very useful to higher level routines.

The fact that there are still broad bands on the curved body of the puppet and in the penumbra in figure 4 merits some explanation. Examination of equation (8) shows that the "edgeness" measure k takes its maximum value of unity if the coefficients A_3, A_4 and A_5 are all zero. This condition is satisfied by any picture function of the form:

$$p(x,y) = b + g(x \cos \theta + y \sin \theta)$$

where g is odd. Unfortunately a region of uniform gradient can be expressed in this form, and hence the bands. To make the operator more discriminating more coefficients would have to be used, and this would apparently require a search over possible orientations in the second stage. I avoided this because I thought that the search would slow down the operator too much, but I now realise that the search need rarely be performed. Most regions of a picture are of roughly uniform intensity and the Walsh coefficients for such a region are trivial: all except the first are nearly zero. Thus in most cases a simple test on the coefficients is sufficient to decide that no edge is present, and the search is then unnecessary.

REFERENCES

Binford, T.O & B.K.P.Horn (1971) The Binford-Horn Line Finder, M.I.T. A.I. Laboratory Working Paper 16.

Hueckel, M. (Jan. 1971) An operator which locates edges in digitised pictures, in J. Ass. comput. Mach. 18, 1, pp. 113-125.

O'Gorman, F. & M.B. Clowes. (1973) Finding edges through collinearity of feature points , Proc. IJCAI, 3, pp. 543-555, Stanford.

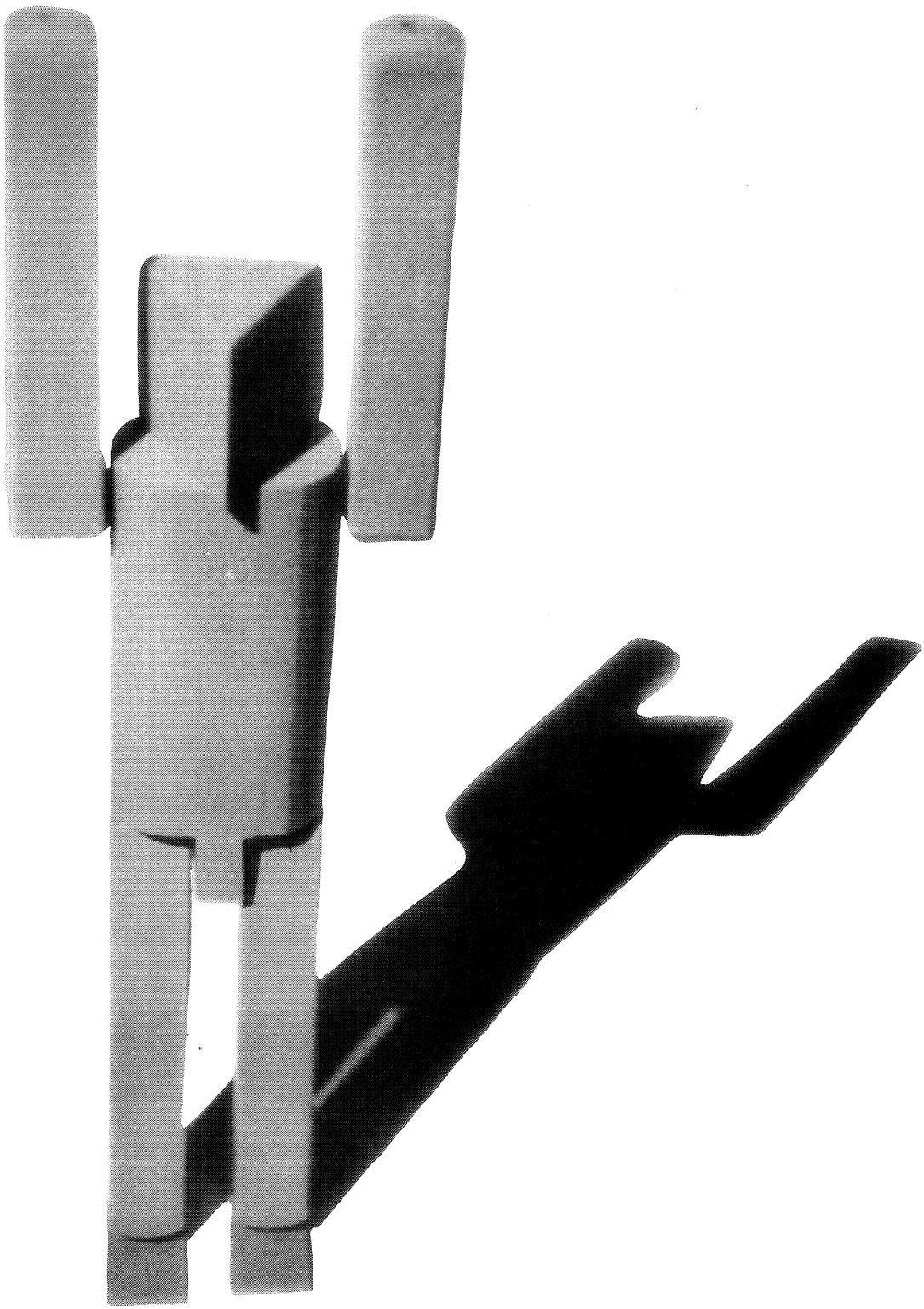
Pingle, K.K. (1969) Visual perception by computer, in Automatic interpretation and classification of images, (ed. A. Grasselli), pp. 277-284, Academic Press, New York.

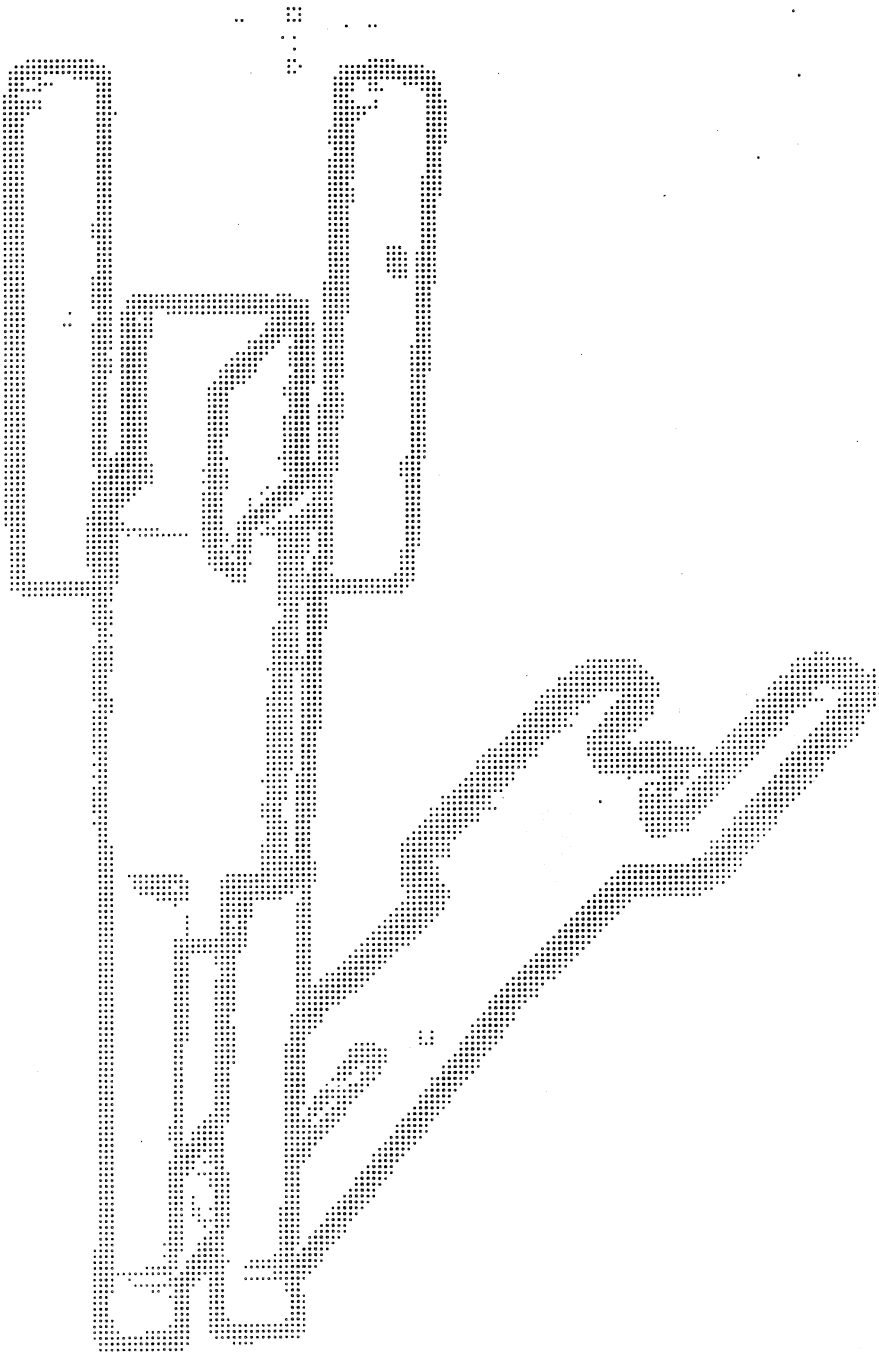
Roberts, L.G. (1965) Machine perception of three-dimensional solids, in Optical and electro-optical information processing (eds J.T. Tippett et al), pp. 159-197, M.I.T. Press, Cambridge, Mass.

Searle, N.H. (1969) Shape analysis by use of Walsh functions, in Machine Intelligence 5, (eds B. Meltzer & D. Michie) pp. 395-410, Edinburgh Press.

Shirai, Y. (1973) A context sensitive line finder for recognition of polyhedra, in Artificial Intelligence 4, 2, pp. 95-120.

Walsh, I.L. (1923) A closed set of orthogonal functions, in Amer. J. Math., 55, pp. 5-24.





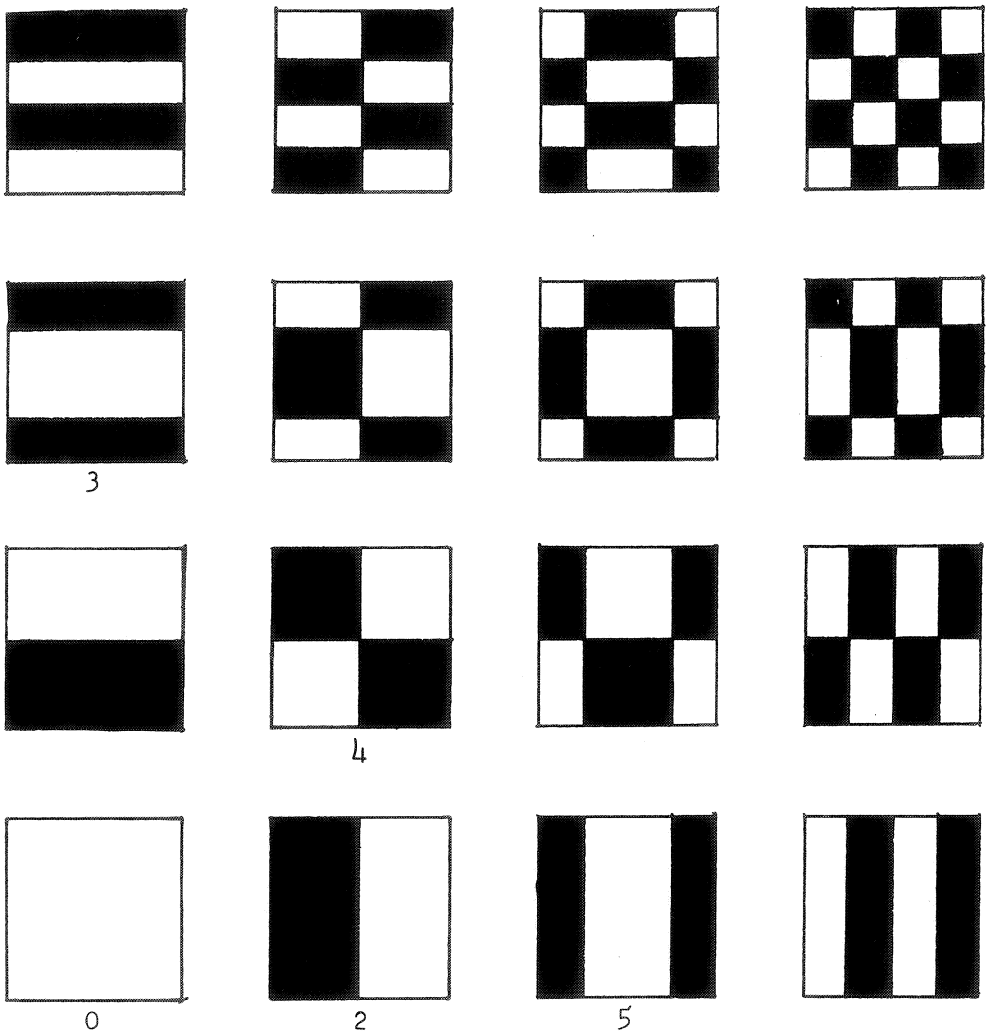
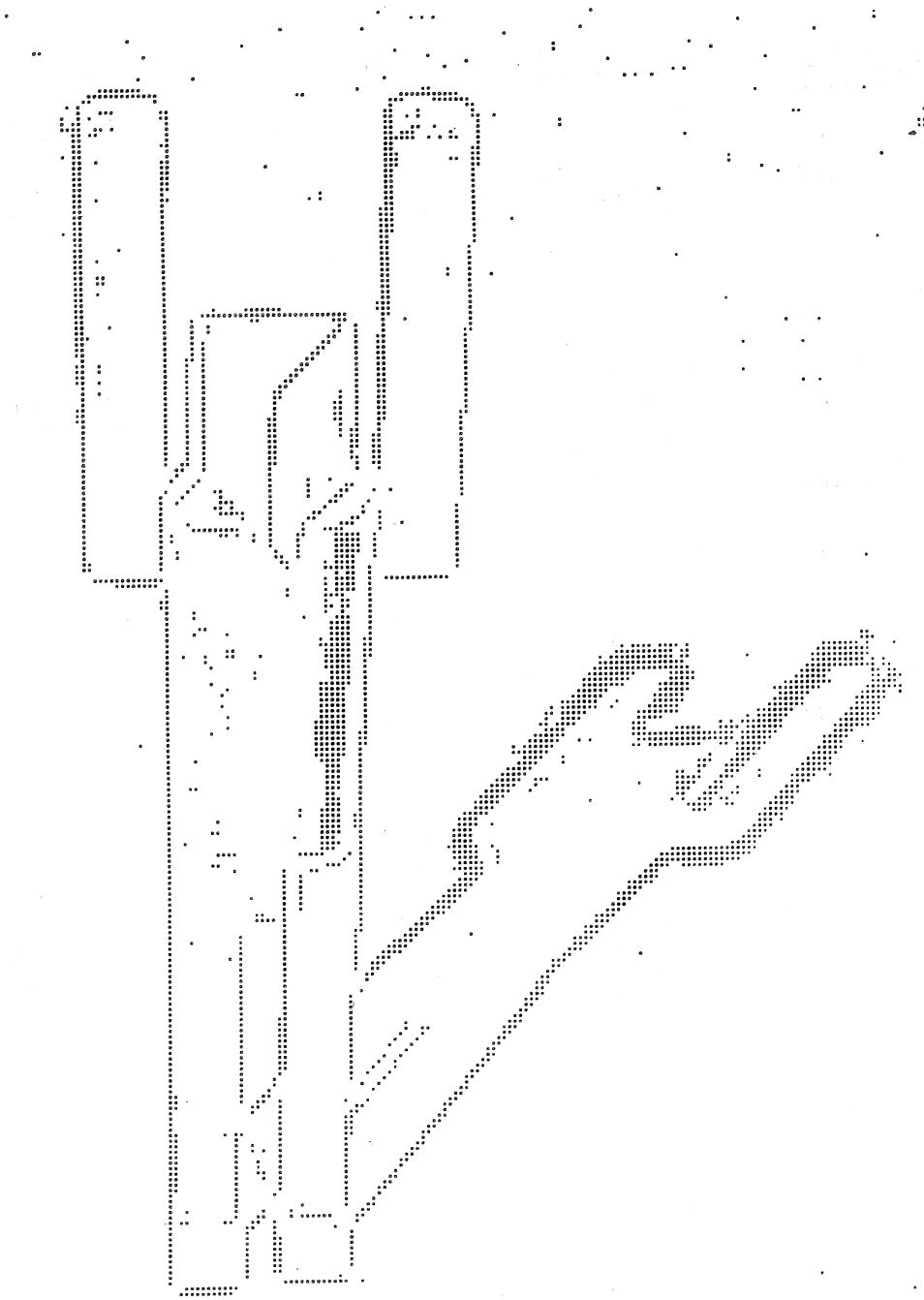


Figure 3. Two dimensional Walsh functions. White represents the value +1, and black -1. The numbered functions are those used in the operator described in the text.



M.Ornato
G.P.Zarri

AN APPLICATION OF ARTIFICIAL INTELLIGENCE IN INFORMATION RETRIEVAL : RESEDA PROJECT FOR MEDIEVAL BIOGRAPHIES

1. The RESEDA project was formed to provide specialists of the XIVth and XVth centuries with an interacting method of processing biographical data. A model is at present being worked out with the personages in Charles VI's so called Love Court, thanks to the joint financing of the "Centre National de la Recherche Scientifique" and the "Délégation Générale à la Recherche Scientifique et Technique" (contract 75.7.0456) .

The methodological interest of the project seems to lie essentially in the attempt to enrich the techniques of information retrieval by instruments endowed with a certain deductive power (Cros *et al.* 1968) and thus to assess on a concrete example the weight of the tools which have to be set up in order to simulate a series of quite complex intellectual processes.

Moreover, it is almost essential to have recourse to very sophisticated methods owing to the extremely complex and very often implicit nature of the relations between the personages who interest us. The target is in fact the creation of a system which would have not only a static function of recuperating a stoked information but also a dynamic function which would allow us to establish new links between the data and to increase as it were the initial knowledge. Therefore RESEDA must to a certain extent enter the sphere of A.I.

2. RESEDA is organised around a memory structure at two levels : the information "personages", which constitute the data-base, in the usual sense of the term, continually kept up to date during the feeding process of the network, and the "metalanguage", chosen in a pragmatic way in accordance with our aims and which serve as support to the expression of the information.

2.1. The "personages" data are stored in the form of a series of "planes" -understood in the sense of Quillian (1968; see also Scragg 1975).

2.1.1. These planes are organized in two ways :

2.1.1.1. physical organization is accomplished following a *sui generis* chronological criterion which has been developed specially to take into account the uncertainties and inexactitudes

characteristic of Medieval documentation;

2.1.1.2. Logical organization divides the planes among different "volumes", each volume being dedicated to a single personage, the "lead" ("vedette"). The criterion for attributing a given plane to a volume is the presence of the lead in formulating the episode that the plane relates. Since the episode may involve several personages at once, the plane as a whole will therefore be attributed to the various corresponding volumes (Scragg 1975). Each volume can thus be thought of as a list of addresses which point to the plane labels (inverted list).

2.1.2. The coding representing the episode related in a plane is divided into two parts, the circumstantial frame or "thematic" and the proper description. This cutting up is far from being arbitrary : the "historical" nature of our information renders inevitable references of time and place which we will thus always be able to isolate. From a more strictly practical point of view, this form of presentation allows us to make the coding of the second part lighter and one will readily find, without entering into the details of the episode, these two types of information which are very often used to guide research.

2.1.2.1. The thematic, with seven zones of recording separated in the examples which follow by slashes, takes over the information of dating and localization, and also the identification of the informant (in most cases bibliographical references). The first three zones allow us to mark the date of beginning, the dates of attestation, and the date of termination; the next three zones, the place of departure, the places passed through, the place of arrival or simply the place. By date of "attestation", we mean the certified statement that at that given time, the "action" -or the "state"- which is mentioned in the episode is taking place according to the testimony of the examined source. Regarding the coding of the localization, the organisation in three zones has naturally been adopted in order to take journeys into account. When the episode takes place only in one place the fourth and fifth zones remain empty while the sixth one contains the "simply place". None of these first six zones need necessarily be filled, unlike the seventh and last one which must always give the origin of the information stated in the episode. The dating information of the thematic are used for the chronological organisation of the planes which have already been mentioned.

2.1.2.2. The description of the episode -absolutely independent from its formulation in natural language- takes the form of a tree whose nodes are "correlators" (Ceccato 1962) and whose terminals are "predicates" -possibly preceded by "modulators"-, elements of the "lexicon" or leads (the "correlators", "predicates", "modulators" and terms of the "lexicon" are elements of the

metalanguage). This mode of representation meets with certain results obtained by linguist who are working on the comprehension of natural language with tools of the conceptual case-systems type (Schank 1973, 1975; Bruce 1975).

To illustrate the interaction of these different elements in the coding, we can give as an example a paragraph of Clamange's biography such as : "from Paris, Clamanges went to Avignon. There he met Jean Muret and Jean Moccia again. On 13 November 1397, he was appointed Secretary to the Pope by Benoît XIII and he stayed there until May 1398", which generates three planes (the coding is understandably incomplete here for the sake of clarity) :

thematic : /-/-/avant-13-novembre-1397/Paris/-/ Avignon/Bibl: Glorieux/ (1)
description : (*CLAMANGES* .SUJ. (*DEPLACER* .OBJ. *CLAMANGES*))

/-/13-novembre-1397/-/-/-/Avignon/Bibl: Glorieux/ (2)
(((*CLAMANGES* .COORD. *MURET*) .COORD. *MOCCIA*) .SUJ. *E/RESIDANT*)

/13-novembre-1397/-/mai-1398/-/-/Avignon/Bibl: Glorieux/ (3)
(*CLAMANGES* .SUJ. ((*soc+E/BENEFICIANT* .SPECIF. *secrétariat-pontifical*) .SOURCE. *BENOIT-XIII*))

where

- Glorieux, name of the author of the biography, is the symbol here of a coding which takes into account the nature of the work, its authour, its title, etc...
- SUJ, OBJ, COORD, SOURCE are "correlators".
- *DEPLACER* is a "predicate of action", *E/BENEFICIANT*, *E/RESIDANT* are "predicates of state" and *soc* is a "modulator".
- *CLAMANGES*, *MURET*, *MOCCIA*, *BENOIT-XIII*, are leads.
- *secrétariat-pontifical* is a "lexicon" term.

We stress the purely pragmatic nature of the classification of the metalanguage : the definition of the different categories is nothing else but a definition by list. For example, will be "correlators" all the units included in the close series : "ARGUMENT, CAUSE, CONDITION, COORDINATION, DESTINATION, FINALITE, MODALITE, OBJET, QUALIFICATION, RELATION, SOURCE, SPECIFICATION, SUJET".

As for the logical function of these different categories, it is sufficiently clear in itself for the correlators -which allow us to organise the coding systematically in groups of three interlocking elements- for the predicates -which correspond roughly with the verbs of natural language-, for the lexicon and the leads. The modulators' function is to restrict as much as possible the list of predicates by allowing us to specify the scope of the action or the state which is stated

in the episode with the indirect help of a limited number of prefixes (see also Mel'chuck, Zholkovskij 1970). It is quite obvious that the terms of French vocabulary used have only a mnemonic value. *E/BENEFICIANT*, for example, has the general sense of "to possess something during a certain period"; with the prefix *soc* (social) as in the previous example, this tenure places itself within the framework of the institutional interrelations of the period; with the prefix *ment* (mental) on the other hand, it is an intellectual enrichment which is made manifest.

2.1.3. A system of pointers is associated with the plans in order to memorize in RESEDA informations about the relations which exist between episodes stated in different planes. These pointers will be inserted either at the moment of the coding and when bringing the data up to date or as the result of inferences possibly triggered by certain system questions (cf. 4.3.). Thus, "short-cuts" are established which allow us to connect directly among them certain pieces of information whose initial connecting process has necessitated the creation of a long chain of inferences (Quillian 1968).

The pointers, from a conceptual point of view, are in fact only correlators of a particular nature. In the present organisation of RESEDA, the correlators CAUSE and FINAL are reserved only for the establishment of connections between separate episodes; COORD, which normally appears within the coded planes, has in this case an auxiliary function.

We must explain the coexistence of FINAL and of CAUSE in a function which appears symmetrical. In fact, these two correlators are distinguished in a most pragmatic way which allows us to keep FINAL for the insertion of certain episodes which -considering what we actually know at the moment of coding- seem to be the targets, without confirmed carrying into effect, of the predicate expressed in the first correlated. The functions of these two correlators may be illustrated by this sequence of episodes :

/après-11-août-1408/-/-/Paris/-/Cambrai/Bibl: Valois/ (4)
(*CHARLES-VI* .SUJ. (*DEPLACER* .OBJ. (*SAINT-POL* .QUALIF. *comte*)))

/après-11-août-1408/-/après-11-août-1408/-/-/Cambrai/ (5)
Bibl: Valois/
(*SAINT-POL* .SUJ. (*contre+exec+PRENDRE-POSITION* .OBJ. *PIERRE-D'AILLY*))

/après-11-août-1408/-/-/-/-/Cambrai/Bibl: Valois/ (6)
(*PIERRE-D'AILLY* .SUJ. *neg+phys+E/LIBRE*)

/-/11-août-1408/-/-/-/Paris/Bibl: Valois/ (7)
(*PIERRE-D'AILLY* .SUJ. (*neg+E/PRESENT* .SPECIF. *5e-concile-Eglise-de-France*))

((4 .FINAL. (5 .FINAL. 6)) .CAUSE. 7)

whose signification in natural language is : "noticing the absence of Pierre d'Ailly, on 11 August 1408, at the opening of the 5th Council of the Church of France in Paris, Charles VI sent the Count of St Pol to arrest him in Cambrai".

We must note that the modulator *exec* indicates that the action is "executed" on the order and responsibility of a person other than the agent.

Owing to the "non-confirmed" nature given to the episode which is the second correlated of FINAL, this pointer cannot in any way be the result of an inference, which in RESEDA is reserved for the setting up of connections between episodes whose historical reality has already been established. On the other hand, if a new element brings a testimony of the actual happening of the episode second correlated, it will be necessary, owing to the principle which has just been stated, to replace FINAL by CAUSE by inverting the order of the correlated.

2.2. The constituents of the metalanguage, which have just been presented in the way they function, can be classified in two categories with respect to the way they are represented in information processing.

2.2.1. In the first one, are placed those whose semantic function is expressed under the form of programmes (Winograd 1973). The "correlators", the "predicates" and the "modulators" come in this first category. Each predicate has as reference a particular program which describes its directions for use in processing terms; modulators and correlators play the role of differentiating elements by diverting the research towards different subroutines.

2.2.2. In the second category are placed the elements of the lexicon which, on the contrary, must be defined in relation to each other. This establishment of relations can have three aspects :

2.2.2.1. First we have the tree structures as in the traditional relation of the general to the particular (for example, the ecclesiastical classification in *parish, Deanery, Archdeaconny, Diocese, province*). It is possible to ensure the passage from one tree to another by setting up between certain of their nodes links of "equivalence". We can notice that a certain number of these terms correspond to a "collective of leads" characterized by the enumeration of the individuals which make it up. For example, the generic term *popes-of-Rome* has as specific elements *URBAIN-VI, BONIFACE-IX, INNOCENT-VII, GREGOIRE-XII, MARTIN-V*. In this one case, we insert in the volume of each of the leads concerned all the planes which contain the term "collective".

2.2.2.2. Next we have definitions of the type "item of dictionary", where are used one or more planes linked, if need be, by correlators-pointers. We can give as example the central cell of the definition of *grand-schisme-d'occident* :

/1378/-/1418/-/-/occident/Bibl: consensus/ (8)
(*église-catholique* .SUJ. ((*contre+PRENDRE-POSITION* .OBJ. *église-catholique*) .ARG. *papauté*))

/-/9-avril-1378/-/-/-/Rome/Bibl: consensus/ (9)
(*sacré-collège-1* .SUJ. ((*soc+DONNER* .OBJ. *papauté*) .DEST. *URBAIN-VI*))

/-/21-septembre-1478/-/-/-/Fondi/Bibl: consensus/ (10)
(*sacré-collège-2* .SUJ. ((*soc+DONNER* .OBJ. *papauté*) .DEST. *CLEMENT-VII*))

(8 .CAUSE. (9 .COORD. 10))

The signification in natural language of this is : "from 1378 to 1418, in the West, the Church is divided within itself about the occupancy of the Papal Seat. For the election in Rome, on 9 Avril 1378, of the Pope Urbain VI by certain members of the College of Cardinals was questioned during a new election made in the neighbouring town of Fondi for the benefit of Clement VII, on 21 September, by a College of Cardinals slightly changed".

Naturally, for these dictionary definitions, the pointers are not the result of an inference but are laid down in advance at the moment of the establishment of RESEDA.

When there are several planes in a definition, they are not necessarily all connected among themselves by pointers. For example, for *schism*, we will have, in addition to the "cell", a whole series of planes which bring information of a quite general nature on this event and which are part either of definitions or of the volume of certain leads. In this case, they contain very often *schism* as second correlated of SPECIF or of ARG, and that allow us to insert in the definition elements of recursivity (an example will be found in plane 14 -see 4.3.- which is classified under *UNIVERSITE-DE-PARIS* as well as under *schism*).

2.2.2.3. Finally we have examples of the *sui generis* structuration of lexicon elements whose most complex example is provided by the system of kinship, where different axes (sex, marriage, fraternity, consanguinity) are going to intersect in order to determine pairs of symmetrical terms (of the type father - son, uncle - nephew, mother-in-law - son-in-law) and their sequences (the son of the sister = the nephew).

The correlator RELATION is specifically reserved for *sui generis* structures, among which we are at present providing for, in addition to *kinship*, *sponsorship-stricto-sensu*, *university-*

sponsorship, protection.

3. To question RESEDA, the user formulates his questions indirectly (Carbonell, Collins 1974) in accordance with coding principles which are rigorously identical to those used to enter planes in the data base (see examples in 4.). In both cases, RESEDA does not involve setting up an automatic analyser for natural language : its realization seems utopian -at least initially- given the very erudite nature of the sources to be analysed and the level of complexity required in the metalanguage. On the other hand, it would appear indispensable to complete a series of conversational programs which will guide the analyst in coding planes and the user in formulating question. This prompting will notify him at each step of errors in formal syntax, lexical inaccuracies, and, in general, will suggest authorized procedures to him.

In order to specify the precise aim of his question, the user has four modes of questioning available to him :

3.1. The question mark is entered before the unknown term X which replaces a correlated item inside the parenthetization. The awaited reply will be the instantation inside the same structure of some term which will transform the question into a valid assertion;

3.2. The question mark is entered before the unknown term X which replaces the modulator of the predicate. This pratice will be exemplified under 4.3.

3.3. The question mark is entered in a correlation before a known term of the RESEDA lexicon. This possibility corresponds in particular to the search for specifics in the trees of the marked term (see 4.2.1.)

3.4. The question mark is entered before the entire parenthetization in order to show that the question regards the truth value of the formulation put forth;

3.5. The question mark can, of course, also appear in the thematic. It should be noted that RESEDA does not permit one to ask about the description and the circumstantial frame at the same time nor, in general, does it allow asking more than one question at once.

4. RESEDA gives answers of varying reliability by using its memory structure according to various strategies.

4.1. The first consists in seeking a term by term coincidence between what is stated in the question and all or part of the formulation of a plane ("direct match"). This can involve the thematic as well as the description; in the latter case, the analogy must first be set forth at the level of correlators and

of their parenthetic organization. The only variance permitted will be the absence of certain correlators contained in the plane in the question.

Here, we should also indicate the problem of disputed information, quite characteristic of Medieval documentation : each of the conflicting versions has to be placed in a separate plane, but the series of planes involved in the controversy is recovered in a special list. Whatever the mode of search used (direct match, inference, etc...), when the data obtained in the reply is contained, even partially, in a plane that is listed in this manner, the same search technique should be repeated on the series of planes associated with it.

Under these conditions, the reply along with its possible "divergent information" can be considered to be "sure in the present state of our knowledge".

In addition to the direct match as such, the search is made in accordance with the possibilities offered by various organizations (see 2.2.) of RESEDA metalanguage components on two levels : first, that of a simple extension of the direct match, then that of more elaborate inferences. These possibilities are of course compounded in the case of a complex search, but we are separating them here for reasons of clarity.

4.2. Extension of the direct match : we can (adopting the terminology of Schwarcz *et al.* 1970) call this way of proceeding "generalized direct match".

4.2.1. One version of this is based on elements of the question that in the RESEDA lexicon are connected to structure 2.2.2.1. We can consider, for example, the question :

/-/-/avant-1394/-/-/Paris/-/
(*CLAMANGES* .SUJ. (*ment+E/BENEFICIANT* .SPECIF. *?cours-fac-arts*))

the meaning of which is "which courses did Clammanges attend at the Faculty of Arts in Paris before 1394 ?" Let us give a piece of the *enseignement-public* arborescence :

. *enseignement-public*
.. *cours-d'enseignement*
... *cours-préparatoire*
.... *cours-petites-écoles*
.
.
.
.
.... *cours-fac-arts*
..... *morale*
..... *logique*
..... *grammaire*
.

.
... *cours-fac-supérieure*
.... *cours-fac-théologie*
.
etc...

At the *cours-d'enseignement* level, one finds a relationship of equivalence with *formation-privée*, which is the upper end of another arborescence :

. *formation-privée*
.. *formation-commerciale*
... *comptabilité*
.
.
.
.. *formation-érudite*
... *grec*
... *hébreu*
etc...

Through scanning the specifies of the term *cours-fac-arts*, as marked in the question, we get a series of elements on which the direct match should be attempted, and which might provide answers "sure in the present state of our knowledge" such as, for instance, the information that Clamanges followed a course on Ethics (*morale*).

If this should fail, one should not start another procedure without first scanning, within suitable threshold limits, the nodes placed above the marked term in the arborescence and their specifics. One will retain, for instance, that Clamanges had attended divinity lectures (*cours-fac-théologie*). One will also scan other arborescences for nodes related to the starting term or its generics by a relationship of "equivalence" (Cros *et al.* 1968; Schwarcz *et al.* 1970). One will then discover that Clamanges had some notions of Greek (*formation-privée, grec*). With *cours-fac-théologie* and *grec*, one will have thus retrieved valuable informations as to Clamanges' focuses of interest during the period considered.

In the latter examples, we did not get a real answer, but rather pieces of information related to the topic, and these should guide the user towards new pertinent questions.

4.2.2. A second type is based upon elements of the question that in the RESEDA lexicon are connected to structure 2.2.2.3.

Discovering a relationship which is supplied indirectly is of special importance given the nature of the documentation (biographical data) that RESEDA is to process.

Seeking the kinship that exists between X and Y, one thus finds no indication X .REL. (kinship) Y but one finds in the volume of X statements X .REL. (kinship) A, B, C, and in the vo-

lume of Y statements Y .REL. (kinship) M, N, O, where .REL. is the particular correlator reserved for structures of this type (kinship, godparents, protection...). Starting at the same time with roots X and Y, one then constructs two trees where the first level nodes are represented by A, B, C, and M, N, O, and where the succeeding levels are obtained in the same way based upon each of the first level nodes. If in constructing a level, one finds a node belonging already to the same tree, it is no longer considered; if, on the other hand, it already belongs to the other tree, one has then found the point of coincidence that was sought. An empirical threshold will be established in order to avoid the excessive extension of trees. If Z is the common point, there will finally be a string like :

X .husband-of. C .daughter-of. C1 ... .REL. Z .REL. ... M1 .REL. M .REL. Y

One then uses the *sui generis* structure of kinship in order to translate in turn each adjacent pair of the string in the terms of the corresponding kinship (X husband of the daughter of C1 = X son-in-law of C1 etc...). One will finally find the sought after tie.

4.3. RESEDA includes a certain number of "second-order" inference structures in order to compensate as much as possible for drawbacks resulting from our incomplete view of the Medieval world. Of course, the replies supplied by these inferences are of a "probable" rather than a "sure" type.

These procedures are essentially related to the definition of "predicates" in the form of programs (2.2.1.). We can get an idea of this by examining in its general outline the strategy adopted to reply to the question "What is the attitude of Pierre d'Ailly towards the University of Paris during the year 1395 ?" This question will be coded thus :

/début-1395/-/fin-1395/-/-/-/-/
(*PIERRE-D'AILLY* .SUJ. (*?X+PRENDRE-POSITION* .OBJ. *UNIVERSITE-DE-PARIS*))

As a first step of proceeding, RESEDA tries to accomplish a direct match : we will examine the volume of the particular person within the time limits defined by the thematic, by trying to find a plan which has the same minimal formulation as the question and one or several modulators instead of *X*. We will note that, if a user has a very precise idea of the context of *PRENDRE-POSITION* that he wishes to obtain, he can have the *X* followed by a list of modulators to be put aside, each preceded by a minus sign (for example *?X-exec+PRENDRE-POSITION*).

When the direct match does not give any result, we should then try a "generalised direct match" -that is, try to use the "tree" definitions of the lexicon (2.2.2.1.)- but these means are not given to us here (there are no lexicon terms in the question).

The algorithm of answer then calls the program *PRENDRE-POSITION* which checks in the question the presence of a modulator (optional), of the "object" (compulsory) and of the "argument" (optional). The absence of an argument and of a modulator diverts us to a subroutine which asks, in the first place, if the object is a lead. As this is the case here, we then look for the planes which are at the intersection of the volumes of the subject and the object -in the period defined by the thematic- and which contain between these two protagonists the predicate *PRENDRE-POSITION*. We will then find in our example "on the second of August 1395, the University of Paris forbids Pierre d'Ailly from being present at its assemblies about the Schism" in the form :

/-/2-août-1395/-/-/-/Paris/Bibl: Valois/ (11)
(*UNIVERSITE-DE-PARIS* .SUJ. (*contre+real+PRENDRE-POSITION* .OBJ. *PIERRE-D'AILLY*))

/2-août-1395/-/-/-/-/Paris/Bibl: Valois/ (12)
(*PIERRE-D'AILLY* .SUJ. (*neg+E/PRESENT* .SPECIF. (*assemblée-universitaire* .ARG. *schisme*)))

(11 .FINAL. 12)

A research is then started by the program to find the first occurence of the correlator ARG; it is continued, if need be, in all the planes connected to the first by correlators-pointers. If this research did not lead to anything, we would get out of the program *PRENDRE-POSITION* by giving the set of planes we have found as elements of information.

According to the hypothesis that a "manifestation" of A "towards" B can be caused by a previous "manifestion" of B on the same matter, the program then acknowledges in the elements taken from 11 + 12, the first part of a figure :

(*P2* .SUJ. ((*pour/contre+PRENDRE-POSITION* .OBJ. *P1*) .ARG. *a*) (α)

(*P1* .SUJ. ((*pour/contre+PRENDRE-POSITION* .OBJ. *P3*) .ARG. *a*) (β)

(α .CAUSE. β)

(see also the "demons" of Charniak 1972); the program then tries to complete this figure even if it may have to retrace its steps in case of failure, and go through the definition of the lexicon term *a*. *PRENDRE-POSITION* will thus look in the volume of *P1* (Pierre d'Ailly) for a plane bearing a date prior to that of α (borrowed from 11 + 12) and having the form of β. If this research is successful, it remains to look in the planes common to *P3* and *P2* (University of Paris) -by a "generalised direct match" research- for the following figure :

(*P2/P3* .SUJ. ((*pour/contre+PRENDRE-POSITION* .OBJ. (γ)
P3/P2) .ARG. *α*)

where the date is also prior to that of α and where the modulator must correspond to what this table requires :

α	β	γ
contre	pour	contre
contre	contre	pour
pour	pour	pour
pour	contre	contre

Thus, β will be satisfied by "about April 1395, Pierre d'Ailly openly adopts a definite position in favour of the Pope of Avignon Benoît XIII about the Schism." In the form :

/-/vers-avril-1395/-/-/-/Paris/Bibl: Valois/ (13)
(*PIERRE-D'AILLY* .SUJ. ((*pour+PRENDRE-POSITION* .OBJ. *BENOIT-XIII* .ARG. *schisme*))

The intersection of the volumes of *P3* and *P2* gives among others "as from 6 January 1391 and until 1418, the University of Paris is always in opposition with the Pope of Avignon about the Schism" in the form :

/6-janvier-1391/-/1418/-/-/-/Bibl: consensus/ (14)
(*UNIVERSITE-DE-PARIS* .SUJ. ((*contre+PRENDRE-POSITION* .OBJ. *papes-d'Avignon*) .ARG. *schisme*))

where the methode of generalised direct match enables us to recognise in 14 the figure γ (see "collective of protagonists" in 2.2.2.1.).

All the conditions having been met, the program *PRENDRE-POSITION* ends and gives to the algorithm of answer all the relevant planes which helped bring it to a successful issue, and which will thus make up the answer to the question that was asked.

If this result is judged to be feasible, the connection established between planes (11 .FINAL. 12) and 13 will be stored by a pointer-correlator .CAUSE. in such a way as to find it directly in the future, should the occasion arise.

5. Some final words, now, about the present state of the work and its perspectives. October 1977 has been fixed by the D.G.R.S.T. as deadline for our Team to work out a restricted operational model of RESEDA on a body of about a thousand biographies.

At the present time -April 1976- the general organisation of the project and its dynamics are fully defined and processing tests have been made on the logical coherence of certain complex parts, on the structures of kinship, on the predicates, on the hierarchy of correlators etc...

From May onwards, will start the phase of transcribing and

entering biographical data and the study of the definitive implantation of files on mass memory. This should enable us to launch an intensive campaign of tests on procedures for retrieving information in RESEDA and, more particularly, on inference algorithms.

RESEDA is constructed and operated from a 2741 *Ibm* terminal linked to the Centre Inter Regional de Calcul Electronique (C.I.R.C.E.) of Orsay. The absence, in the latter's instrumentation, of specialised languages for Artificial Intelligence -like MICRO-PLANNER and CONNIVER (Bobrow, Raphael 1974)- has led us to choose the language APL\360-TSO as programming support. Its mode of functioning, based on structuring data in arrays has proved particularly useful for the representation of the trees contained in the planes. In a more general way, its power and its flexibility should enable us to simulate certain characteristics of the programming languages proper to Artificial intelligence as well as to have at our disposal general frames for constructing discrimination nets, processes for the control of tree search, pattern matching mechanisms.

REFERENCES

Bobrow, D.G., Raphael, B. (1974) New Programming Languages for Artificial Intelligence Research, *ACM Computing Surveys*, VI, 155-174.

Bruce, B. (1975) Case Systems for Natural Language, *Artificial Intelligence*, VI, 327-360.

Carbonell, J.R., Collins, A.M. (1974) Natural Semantics in Artificial Intelligence, *American Journal of Computational Linguistics*, I, microfiche 3.

Ceccato, S. (1962) *NATO Seminar on Mechanical Translation - Course Notes*. Milano: Centro di Cibernetica dell'Università.

Charniak, E. (1972) *Toward a model of children's story comprehension* (A.I.-TR266 Memo). Cambridge (Mass.): MIT Artificial Intelligence Laboratory.

Cros, R.C., Gardin, J.C., Levy, F. (1968) *L'automatisation des recherches documentaires - Un modèle général, le SYNTOL*. Paris: Gauthier-Villars.

Mel'chuk, I.A., Zholkovskij, A.K. (1970) Sur la synthèse sémantique, *T.A. Informations*, n°2, 1-85.

Quillian, M.R. (1968) Semantic Memory, in *Semantic Information Processing*, M. Minsky, ed. Cambridge (Mass.): M.I.T. Press.

Schank, R.C. (1973) Identification of Conceptualizations Underlying Natural Language, in *Computer Models of Thought and Language*, R.C. Schank, K.M. Colby, eds. San Francisco: W.H. Freeman.

Schank, R.C., Goldman, N.M., Rieger, C.J.III, Riesbeck, C.K. (1975) Inference and Paraphrase by Computer, *Journal of the ACM*, XXII, 309-328.

Schwarcz, R.M., Burger, J.F., Simmons, R.F. (1970) A Deductive Question- Answerer for Natural Language Inference, *Communications of the ACM*, XIII, 167-183.
Scragg, G.W. (1975) *Frames, Planes and Nets : A Synthesis*. Castagnola (Lugano): Fondazione Dalle Molle.
Winograd, T. (1973) A Procedural Model of Language Understanding in *Computer Models of Thought and Language*, cit.

J. Lawrence Paul
Cognitive Studies Programme,
University of Sussex,
Brighton.

SEEING PUPPETS QUICKLY

We are trying to interpret scenes containing a puppet with moveable limbs in order to try to understand how we can get an interpretation of the scene without noticing all the details. This involves the use of several intermediate descriptions (e.g. humanoid postures, occlusion, support).

Keywords: intermediate representations, models, scene analysis

If you, the reader, quickly glance at the photograph of the puppet (figure 1), you are able to determine a lot of information about the puppet very easily, for example, his posture (standing, sitting, etc.), the direction he is facing, and the approximate angles between limbs and the trunk. If you look more closely, and examine the picture in greater detail, you see that the arms and legs are composed of rectangular prisms, the head of a triangular prism, and the trunk is rather elliptical in cross-section. Notice what has happened: we seem to see the puppet before, or without, noticing details of its construction. This is in the opposite direction to which a "classical" scene analysis program would operate. It is the ability to see these pictures "at a glance" (almost), without a detailed analysis of the picture, that I am attempting to give the computer.

Let us see what the ability to see these pictures as representations of puppets implies. We have somehow divided the image up into meaningful regions, which somehow suggested a humanoid figure, which helped us divide the image into meaningful regions. The result is, in the picture domain, a figure-ground separation (puppet and everything else), and a division of the figure area into meaningful regions (right arm, left arm, etc.). In the scene domain, the result is an interpretation in terms of humanoid posture, e.g. facing slightly towards left and sitting. The point is, perception of this sort is a complex process in which all of these interpretations are developing at the same time.

Why is the ability to get a quick interpretation of the picture without examining all the details important? This ability is one of the hallmarks of intelligent behaviour. Most

of our ordinary perception requires this ability. Otherwise, we should never see the forest for the trees, never see the house for the bricks. We would be lost in a barrage of details. Once we have done some fast processing, and identified the object, we can then ignore it, or examine it more closely, as we choose. It seems clear that an intelligent vision system will have to have this ability in order to cope. It is the same type of ability, to deal quickly and intelligently with sentences in a specific domain, that workers in natural language are trying to achieve, e.g. Schank (1975) and Charniak (1975). At a more general level, it is the question that Minsky addresses in his "frames" paper (Minsky 1975).

How does this research fit in or compare with other scene analysis work? It can be opposed to Winston (1970) who has to see very clearly each block which composes the arch before perceiving the arch. Closest in spirit to mine are Guzman(1971) and Adler (1965). Working respectively on children's story book illustrations and Peanuts cartoons they try to identify as well as possible different parts of the picture, but many shapes are ambiguous or unclear, and can only be identified in the light of global hypotheses, or relations to other objects in the scene.

Why work on pictures of a puppet? Historically, the answer is "Because it's there". Clowes (1972) created the puppet to provide meaningful scenes for the study of polyhedral objects. The polyhedral objects are not, now, just objects scattered on a table, but meaningfully related. (My research interests were formed in looking at the photographs of the puppet, and realizing that almost no knowledge of polyhedral objects was required to find a fairly rich interpretation of the scene). Intellectually, the answer is that the puppet provides a good vehicle for studying the issues outlined above, i.e. how can we use many different types of knowledge (in this case, humanoid postures, 3-D structures, occlusion, lines, regions, etc.) to interpret a picture quickly.

I have several points in mind while developing the system, which follow directly from the stated goals and motivation of the research, and manifest themselves in different ways. For example,

- Don't expect everything to be meaningful. Don't try for too much high level knowledge from simple low level details. And don't expect all details to be explained in terms of a high level interpretation.
- Make the system puppet specific. Create a data base which deals explicitly with puppets, e.g. handles limbs, trunks, etc, rather than any general form of model of which "puppet" is a specific example. Decisions are made as knowledgably as possible, and based on puppet-specific considerations.

Let me make clear the limitations of the system. First, it has a built-in "set" for seeing a puppet, and cannot recognize any other type of object. It tries its hardest to see a puppet, no matter what is in the picture. Second, the puppet must be "obvious" - one must be able to see it and identify it easily. This means, for example, that the puppet must be well differentiated from the background, the picture must not be too cluttered, the puppet should not be occluded, and we don't want any tricky camera shots (no view from directly above, no fish lens, no gross size distortions, etc).

$2\frac{1}{2}$-D MODEL

The puppet model that we use to find the puppet in the image is a $2\frac{1}{2}$-D model (2-D, plus occlusion of laminar regions). The model consists of six 2-D parts: head, trunk, two legs, two arms. The shapes of the regions are vaguely defined; the important point is that they all have two parallel sides. The pieces must be assembled so that they follow the rules of the model. These rules specify the relationships between different parts. For example: the head region should be parallel to the trunk region, adjacent to it, and near one end; the joint end of the arm should be near the head end of the trunk; either one arm is occluding and the other occluded, or they are both beside the trunk. It is the business of the program to find regions to satisfy the model, even it is has to alter regions found in the picture by elongating or widening them, or even hallucinate regions based on some sort of evidence in the picture.

This $2\frac{1}{2}$-D model only specifies what the puppet looks like, and even then it cannot reject all bad puppets. Why use it, rather than some sort of 3-D model?

- It is a much simpler model than any 3-D model I could think of. It allows a much quicker, though less reliable, recognition of the parts to be achieved. It is probably adequate for a wide range of pictures.
- It is very useful for thinking about occlusion of the parts of the puppet. The reason this works is that the puppet's limb motions are restricted to movements in parallel planes. Thus the occlusion of the puppet's parts must follow that of the $2\frac{1}{2}$-D model, and vice versa.
- The $2\frac{1}{2}$-D model is a good intermediate stage between the data and a 3-D model. Any sort of 3-D model we might want to fit could make use of the $2\frac{1}{2}$-D model.

We shall refer to the instantiation of the $2\frac{1}{2}$-D model as being a $2\frac{1}{2}$-D puppet.

SYSTEM OVERVIEW

This section gives a rough description of the program I am

writing. The example later on demonstrates how much of this plan exists in program form. In this explanation, problems of control, backtracking, etc. are not included. We can think of the program as having five stages.

1. Get line drawing from grey-level data.

The initial grey-level pictures are digitized to about 250 by 250 pixels. A program written by Frank O'Gorman is run on the data first to find the lines (O'Gorman and Clowes, 1973), and then to extend the lines to find junctions in the picture. The line drawing produced from this first stage forms the initial input to my program (e.g. figure 2).

2. Find parallel regions, and rough descriptions.

A parallel region (usually referred to as "region") is a set of more or less parallel lines that overlap to a great extent. For example, in figure 2, lines 37, 45, 49, 24, 56, 55, 52 form a parallel region, but not lines 38, 53 (the overlap is too small). The program definition is more complicated than the verbal one above, but that is basically what is wanted. A parallel region is usually defined in terms of its two outermost edges, and the parallel lines it encompasses.

The reason we look for clusters of parallel lines is that, in the types of scenes we are studying, in which all components have parallel edges, clusters of parallel lines are common in the derived image, and frequently can be associated with one of the components of the scene. We hope that in many cases we can easily identify a component by the cluster of lines; yet the fact that the lines found so often badly match the edges of the component, and our natural scepticism, and the fact that we don't know the actual size of a component in the scene, prevents us from making further deductions, e.g. concerning the component's orientation in 3-D.

The program joins collinear adjacent segments into longer lines, e.g. lines 37 and 44 of figure 2; then, starting at the longest lines, it looks for parallel regions. When it finds a region, it tries to give it an initial classification of "trunk", "head", or "limb" region, based on its length/width ratio. The region is then related to the other regions found. If, for example, a limb-shaped region is found, it compares this with head-shaped regions already found, making sure that the relative scales of the limb and head region are about the same, and then seeing if they are near each other. If so, the limb-shaped region is associated with the head region. Similarly, relative scales and nearness relations between limb and trunk, and relative scales, nearness, and collinearity relations between head and trunk regions, are checked. The

purpose of doing this is to make it easier to decide the best place to put serious efforts into finding the puppet, and to make the $2\frac{1}{2}$-D model matching easier later on.

There is a special-purpose heuristic which is also applied at this point. Large sets of parallel lines seem to be a common feature of a lot of these pictures, for example, the trunk and arms region, and legs region of figure 2. The main idea of the region finder was that it should pick up sets of lines which correspond to a single part of the body, e.g. the set 50, (46,59) correspond to the head. This does not happen in the trunk and arms region, or the legs region, so we make any set of parallel lines, if it has at least three lines in it, and is wider than a limb region, into a special parallel region. We try to break it down into smaller regions, as follows: We find the line closest to the center of the region, if this forms two new regions each of which could be a limb, we add these two regions to our list. If not, we start from each edge of the region, and try to peel off the best limb-shaped region we can find. For example, in figure 2 we pick lines 37, 45, 49 and 56, 55 52 as being nice limb-shaped regions, leaving the region defined by lines 49, 24, 56 in between.

3. Pick starting point, look for puppet.

Given the rough description found above, pick the most likely place to start looking for the $2\frac{1}{2}$-D model. For example, a trunk with three limbs near it seems a better first bet than a trunk with one limb near it. The starting location doesn't have to be a trunk region -- there may be none found; it could be a head region, or a collection of limb regions.

Once the starting point is found, try to fit as many of the regions as possible to the $2\frac{1}{2}$-D model. For example, if a trunk region is found, see if any of the head regions found nearby match it, see if any of the limb regions found could be its arms or legs. At this stage, we try only to use those regions tentatively identified in stage 2; we do not try to hallucinate, reinterpret, etc. The hypothesis is that for a large class of scenes and pictures this will enable a $2\frac{1}{2}$-D puppet to be completely or almost completely determined with no backtracking or hypothesis testing. Global relationships will have made it emerge.

4. Get interpretation, seek missing parts.

If we are lucky, we will have found enough parts to specify a good $2\frac{1}{2}$-D puppet at the end of stage 3. If not, we try to get an interpretation of the partial puppet found, and possibly use this to guide the search for missing parts. It is at this stage that we try to force things on the picture by

extending regions, hallucinating hidden regions on the basis of a line or two in the picture, etc.

5. Find further interpretations from $2\frac{1}{2}$-D model.

We can get a more detailed 3-D interpretation of the picture based on the $2\frac{1}{2}$-D puppet. The interpretation can include the posture, the direction it is facing, the viewing angle (are we looking straight across at the puppet, or perhaps slightly downward), the approximate angles between the limbs and trunk, and the approximate location of the light source.

There are a number of things we could do with this interpretation. For example, we could ask more detailed questions about the picture, such as "exactly where (in the picture) is the left arm", and use the interpretation to guide the matching to a 2-D block model, and a re-examination of the grey-level data to look for or confirm missing lines (a combination of Shirai (1975) and Grape (1973)). Or we could use this interpretation to guide a more demanding puppet finder/verifier, one that was not as slack as the $2\frac{1}{2}$-D model finder.

EXAMPLE

Figure 2 is the line diagram produced from the digitized version of figure 1, i.e. at the end of the first stage described above. Figure 3, shows the groups of parallel lines which were grouped together to form parallel regions, and the subregions derived from these. Table 1 shows the state of the system after stage 2 -- the parallel regions have been found, and the relationships between them described. The region numbers are not sequential -- the missing numbers represent regions tentatively created and then rejected. The "part of picture" column is not found by the computer -- they are my comments to help examine the picture. The "interp" column contains the initial classifications which the system does based roughly on length/width ratios. These regions (and their corresponding interpretations) with which the given region relates on the basis of relative scale, proximity, and, for head-trunk pairs, collinearity, are shown in the "relates to" column.

The first line in table 1, for example, indicates that region 1, defined by lines 53 and 47 in figure 2, lying in the shadow part of the picture, could be interpreted as a trunk. If so considered, regions 12 and 24 could be considered as limbs which are relatively the same scale as trunk region 1, and lie near it. Region 9 could also be a limb, but it is not near enough region 1. The interpretation of region 18 as a head and region 17 as a trunk is not allowed because of the difference in relative sizes (it would have been rejected by

the collinearity requirement between head and trunk anyway). Region 30 is rejected as head to region 17 by the collinearity requirement.

In the third stage, it picks trunk region 17 to be the most likely starting point; the current criterion for choosing is the number of limb and head regions which relate to it. It confirms region 29 as the head. For the arms, it considers those limb regions found to be near the top of region 17, i.e. regions 9 and 12. Each of these is confirmed as a possible arm, but there is no evidence from the picture for either arm that it either occludes or is occluded by the trunk. The type of evidence that would indicate occlusion would be if one of the parallel lines associated with an arm region was found to lie between the edges of the trunk region. Regions9 and 12 form a possible arm pair. The possible leg regions(those limb regions near the bottom of region 17), are regions 28, 24, 23, 9, and 12. Regions 28, 24, and 23 are accepted as possible leg regions. However, it is noted that 23 and 24 are both end regions from one of those special parallel region sets (region 18), which means that they form a more likely leg pair than any other combination of two. So that most likely puppet is composed of regions 29 for head, 17 for trunk, 12 and 9 for arms, and 23 and 24 for legs. Furthermore, because there is no evidence of occlusion, we assume that the trunk is not turned very much, so we can associate regions 12 and 24 as limbs on the same side of the body because they are on the same side of the trunk region; similarly, regions 9 and 23 are paired. If we square off the limb regions, and lengthen them to accommodate the longest line in the region, the final $2\frac{1}{2}$-D puppet is illustrated in figure 4. Figure 4 may look a bit strange; it's not the $2\frac{1}{2}$-D puppet one might have drawn from figure 2. But it is very quickly found, it is possible to say a lot about the interpretation of the picture from this, and it should prove to be a good guide to further examination of the picture.

This illustrates the current state of the program. Further efforts will extend it to cover more difficult pictures (e.g. no trunk region found, ambiguous parsing into regions, limbs partially or completely occluded), and will investigate criteria for choosing the order of processing, and will interpret the $2\frac{1}{2}$-D puppet as a 3-D puppet.

CONCLUSION

In terms of control of information processing, vision seems to be a "middle out" rather than the "bottom up" affair assumed by most scene analysis programs. That is, in examining a picture, we start at some intermediate level, higher than the lines and junctions level, yet below the full

interpretation, and use this level to find the interpretation (going "up"), occasionally looking at details, such as lines and junctions, (going "down") to guide the process. We do not start at the level of lines and junctions, then find the blocks, then find the relationships between the blocks, to identify an object such as the puppet. Once an interpretation has been found, the details of the scene can be seen in the light of the interpretation.

The puppet pictures provide an interesting class of pictures for investigating "seeing without details". The $2\frac{1}{2}$-D puppets provide a useful intermediate stage to help us quickly locate the puppet in the picture, to further interpret the scene, and to guide further exploration of the picture. The results so far indicate the methodology of giving a program knowledge of several "intermediate" levels of structure (lines, regions, region-clusters, $2\frac{1}{2}$-D model, 3-D model) instead of the usual 2 levels, i.e. 2-D lines and junctions, and 3-D vertices and edges.

ACKNOWLEDGEMENT

Funds for this work are being supplied by the National Research Council of Canada (for me), and the Science Research Council of Great Britain (for computing facilities). I wish to acknowledge the idea of flat puppet models as coming from Max Clowes, and thank Aaron Sloman for his help, advice, guidance and encouragement in all phases of this research.

REFERENCES

Adler, M (1975) Understanding Peanuts Cartoons, in Progress in Perception, Dept. of A.I. Research Report No.13, University of Edinburgh.

Charniak, E (June 1975) Organization and Inference in a Frame-like System of Common Knowledge, in Theoretical Issues in Language Processing: An Interdisciplinary Workshop. Cambridge, Mass. pp 42-51

Clowes, M.B (1972) Computer Interpretation of Pictorial Data, Laboratory of Experimental Psychology, University of Sussex

Grape, G.R. (May 1973) Model Based (Intermediate-Level) Computer Vision. Stanford Artificial Intelligence Memo AIM-201

Guzman, A (1971) Analysis of Curved Line Drawings Using Context and Global Information, in Machine Intelligence 6 (eds B.Meltzer & D.Michie)

Minsky, M. (1975) A Framework for Representing Knowledge in Psychology of Computer Vision (ed P.H. Winston) McGraw-Hill

O'Gorman, F. & M.B. Clowes (1973) Finding Picture Edges Through Collinearity of Feature Points, in Proc. IJCAI-3 pp 543-555.

Schank, R.C (June 1975) Using Knowledge to Understand, in Theoretical Issues in Natural Language Processing: An Interdisciplinary Workshop. Cambridge, Mass. pp 117-121

Shirai, Y (1975) Analyzing Intensity Arrays Using Knowledge about Scenes, in Psychology of Computer Vision (ed P.H. Winston) McGraw-Hill.

Winston, P.H. (1970) Learning Structural Descriptions from Examples, Ph.D. Thesis, Report MAC-TR-76, M.I.T.

Region No.	lines*	part of picture	sub regn	super regn	interp	relates to
1	53,47	shadow			trunk	limb 12,24
2	(44,37),45,[12,49],24,56, 55,52	arms & trunk	9,12, 17		head	limb 24
9	52,55,56	left arm		2	limb	trunk 17,28 head 18,29,30
12	(44,37),45,[12,49]	right arm		2	limb	trunk 1,17,28,29 head 18,29,30
17	56,24,[12,49]	trunk		2	trunk	limb 9,12,23,24,28 head 29
18	42,40,[10,36],(58,39),54	legs	23,24 28		head	limb 9,12
23	42,40	left leg		18	limb	trunk 17,28 head 34
24	54,(58,39)	right leg		18	limb	head 2,34,35 trunk 1, 17,28
28	40,36,(58,39)	mid-legs		18	limb trunk	trunk 17 limb 9,12,23,24
29	(59,46),50	head			trunk head	limb 12 trunk 17 limb 9,12
30	26,35	head			head	limb 9,12

Region No.	lines*	part of picture	sub reg^{n}	super reg^{n}	interp	relates to
31	48,28	shadow of head			head	
32	25,2,(27,3), (60,29,57)	feet				
34	15,33	left foot			head	limb 23,24
35	22,14	right foot			head	limb 24

*numbers in round brackets, e.g. (27,3). indicate that the lines are collinear and adjacent and are found before regions are found.

numbers in square brackets, e.g. [12,49] , indicate a composite line, found when the regions are found

TABLE 1

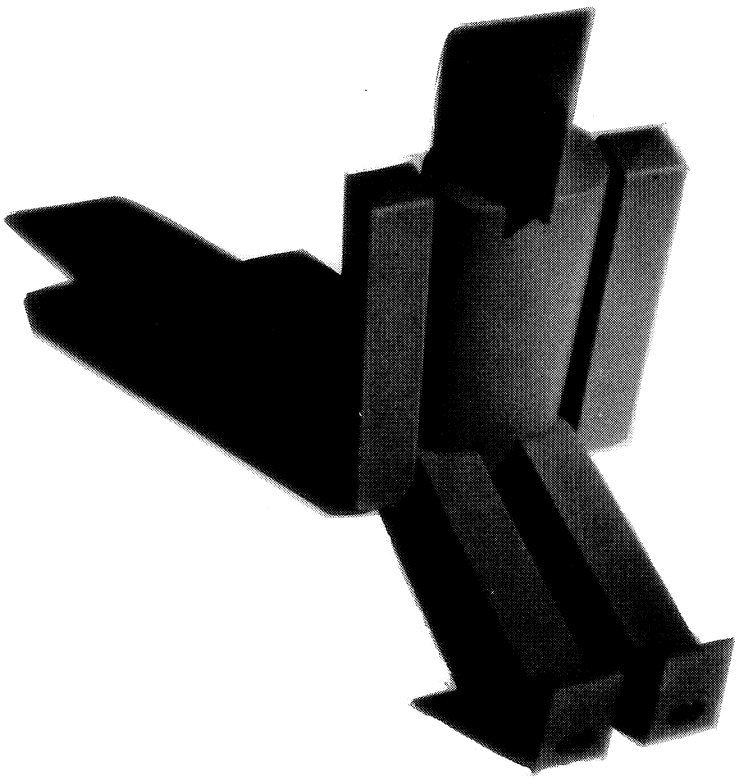


FIGURE 1

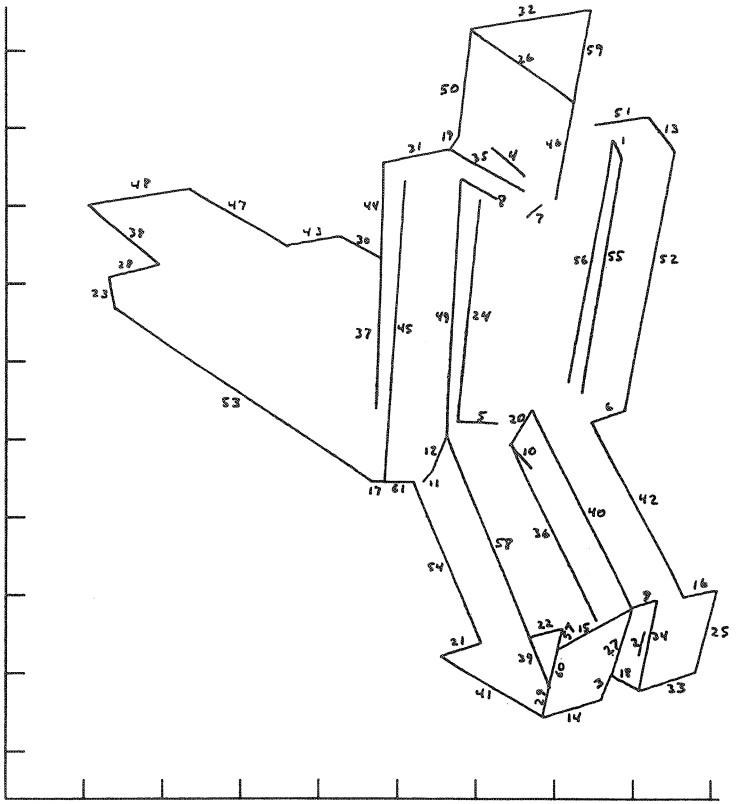


FIGURE 2

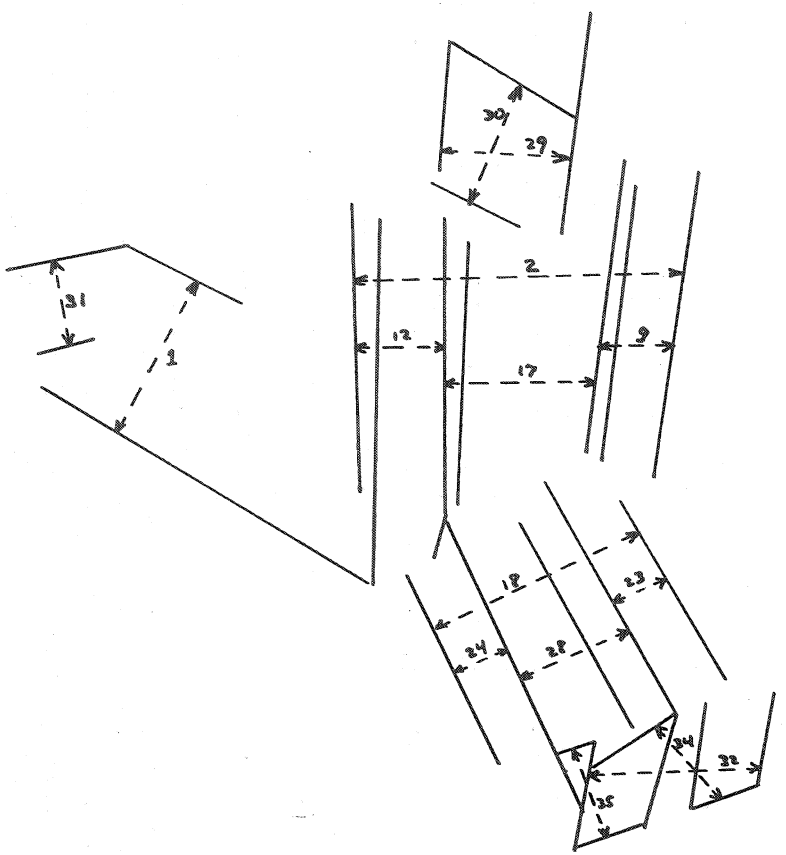


FIGURE 3

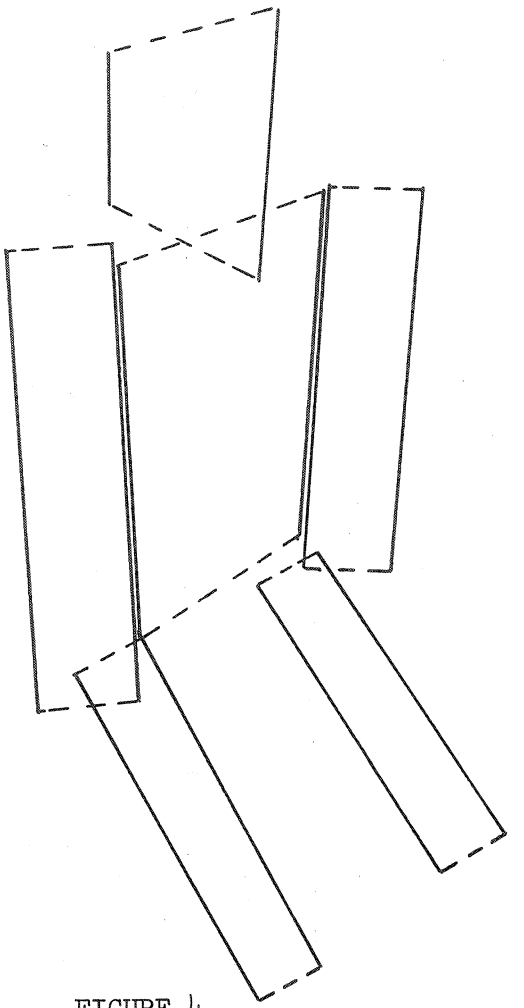


FIGURE 4

G.D.Ritchie

PROBLEMS IN LOCAL SEMANTIC PROCESSING

ABSTRACT

Winograd's SHRDLU program carried out full semantic processing, including computing of referents, as it parsed. If we try to generalise this mechanism, certain difficulties arise concerning inadequate information about contexts of reference. This suggests that we may need to perform much of the semantic processing on senses rather than referents.

Key Words

computational linguistics, semantic processing, sense and reference, parsing, time and tense, run-environment.

The program of Winograd (1972) showed how computational devices (in particular, the representation of a meaning as a procedure) could be used to describe the structure of English. I am engaged in trying to extend Winograd's model, by writing an English conversational program in which the semantic mechanism is more systematic and general (Ritchie (forthcoming)). My program carries out as much semantic processing as possible after each word in the input sentence (in left-to-right order). Attempting to perform immediate semantic processing brings up the question of whether there is adequate information available at such a low level to carry out all semantic processing. Although the problems involved have arisen while constructing a particular program, they are essentially theoretical issues which can be discussed using the concepts from Winograd's program.

Winograd's program performed semantic processing (i.e. conversion from a surface syntactic tree to a BLOCKS world structure) at the end of each noun group (as well as the end of certain other major constituents). One advantage of this was supposed to be that an analysis-path containing a semantically odd item could be discarded. The semantic processing covered two logically distinct stages. Firstly, the translation of the syntax tree for the noun group to a Micro-Planner procedure; secondly, the execution of this procedure in the BLOCKS world (which, if successful, should provide a pointer to some item in the BLOCKS world).

It is worth noting a relationship between these two steps

and the traditional notions of "sense" and "reference". The Micro-Planner procedure for a noun group is a complex description which will identify different items in different contexts. The successful execution of this procedure in a run-environment maps the description to a particular item. Hence the procedure could be looked on as a function from run-environments to BLOCKS world items. If we regard the run-environment as a "state of the world" or "point of reference", and the BLOCKS item(s) as the referent(s) of the noun group, then this is very similar to the "intension/extension" system of Montague (1968,1970,1972). The intension of a term is a function from points of reference to extensions. The extension is (roughly) the set of things referred to by the term. Montague regards this mechanism as formalising the distinction between sense and reference (Frege (1892)). What follows here does not depend crucially on the analogy between procedures and senses, and between results and referents, but the two approaches seem very similar. The terms "procedure-meaning" and "result-meaning" will be used for the Micro-Planner procedure and its result when executed, respectively.

1. LOCAL ANOMALY

One way in which semantically anomalous items were detected in Winograd's program was in the construction of the procedure-meaning. Semantic markers of the constituent meanings were checked to see if they were compatible (as in Katz and Fodor (1963)), and incompatible analyses discarded. In practice it is very difficult to exercise such subtlety in the small BLOCKS world, but the facility was there in principle. However, as McCawley (1968) pointed out (in criticising Katz and Fodor), a semantically odd item may appear in a sentence which, as a whole, is well-formed, as in (1) and (2).

It is nonsense to speak of a rock having diabetes. (1)

He says that he poured his mother into an inkwell. (2)

There need not be a single clear device like "It is non - sense to say that" to allow such effects:-

A round square is a puzzling idea. (3)

Hence, the tactic of aborting locally anomalous analyses seems to be a device which is not generally applicable.

The only way round this problem that has been suggested in the literature is Wilks' Preference Semantics (1975). If different possible analyses are not "discarded" or "retained" on an all-or-nothing basis, but are ranked in order of preference (using either Wilks mechanism, or some other device with this purpose), then examples like (1)-(3) are covered automatically. In each of these cases, only one analysis is possible,

and so it is retained despite the semantic oddity.

Hopefully, such an approach is viable, so this problem is not as disconcerting as those which are to follow.

2. REFERENCE EVALUATION

The second stage of semantic processing (let us call it "reference-evaluation") provided a check on anomaly as follows. If the procedure-meaning did not return a result when executed, then the noun group was less preferred (Winograd, p.14). One possible question to ask in this framework is - in what environment should the procedure-meaning be executed? A run-environment, or context, can be regarded, in this model, as a "state of the world". A phrase like (4) might refer (as Winograd points out (p.146)) to someone sitting on the table now, or to someone sitting on the table at some other specified time; the context of utterance of (4) should make it clear what the necessary environment is.

The man sitting on the table. (4)

It might be thought that the procedure-meaning of a noun group should be run on the completion of that group, so that information about the result-meaning (or lack of one) can guide the analysis process. However, the question is: does the context of use provide this information early enough in the analysis process to enable reference-evaluation to occur as soon as a noun group is parsed? If we identify run-environments with states of the world, then tense-markers and/or time-adjuncts provide relevant clues, as they indicate at what point in time a phrase denotes its referent(s). These clues often occur outside the noun group in question, e.g.

When you worked for ITT, was your boss in the CIA ? (5)

The president of the United States signed the test-ban treaty in 1962. (6)

The Tay Bridge was blown down in 1879. (7)

Such clues or pointers allow the explicit switching of the focus of attention from one environment to another in the course of a dialogue (see Isard (1974) for discussion of such manipulation). Unfortunately, the tense-marker or time-adjunct may occur in a constituent which is parsed after the noun group in question (as in (6) and (7)). This casts doubts on the possibility of immediate reference-evaluation.

If we assume (for the sake of a pessimistic argument) that such environmental information is not always available for each noun group as it is parsed, at least two possibilities arise, which can now be considered (2a. and 2b. below).

2a. Backtracking Semantics

The problem of making a processing decision on inadequate information is not a new one in AI, and is particularly familiar in syntactic parsers. Two classic approaches can be described as "depth-first with backtracking" or "breadth-first", and need no elaboration here. In view of the problem raised in 2. above, perhaps some such measures could be introduced in the area of reference-evaluation. However, these exhaustive strategies are normally employed in a situation where there is a limited set of choices, so that either one option can be chosen (depth-first), or all options can be developed in parallel (breadth-first). When the choice is among alternative evaluation environments, this may not be the case - how can arbitrary points of time be chosen from an indefinitely large set ?

This difficulty may be surmounted by assuming that there is a limited set of possible points of time (or "states of the world") available at a given point in a dialogue. Isard and Longuet-Higgins (1973) suggest that a clause can be described as being related to 3 points of time - the time of utterance, the time used as focus of attention, and the time of occurrence of the event described. The English tense system can be used to manipulate these points of time (with associated states of the world) so that phrases can be understood in selected environments. The tense system allows two selected time-points to be kept in prominence ("PAST" and "PRESENT"), and the time of utterance is always available as a ("PRESENT") time point. Thus it may be that, using the tense system, a parser/analyser could narrow the possible options (for environments for reference-evaluation) to about three, and use some exhaustive strategy on these.

Even then, the breadth-first approach could not be completely correct, as the time/tense system in English allows the creation of completely new foci of attention (using "When"-clauses, for example - see Isard (1974)).

In such situations, this newly-created environment might be the one needed for reference-evaluation, even though it was not available at the start of the utterance as one of the options. However, if we allow some form of "undoing" of semantic processing (i.e. try backtracking), this retro-active choice of an environment might be feasible. Such a mechanism would not be trivial, either in principle or in implementation.

2b. Semantic Processing Using Procedures

One possible solution to the problem of choosing an evaluation-environment for noun groups is to postpone all reference-evaluation until some suitably late stage in the analysis, where the information has become available. This might, for example, be possible on the completion of each clause. Any semantic processing would have to involve manipulation (other than

running) of procedure-meanings. Instead of evaluating a procedure-meaning immediately and using the result-meaning from then onwards, the analyser would have to perform semantic processing on the unevaluated procedure-meanings. Semantic anomaly could be detected only to the extent that the procedure-meaning's own structure (or some explicit labelling) displayed its semantic aspects. If we adopt this approach, two preliminary observations are necessary.

Firstly, one way in which semantic information can be used without locating the particular referent has already been mentioned in 1. - if we accept that there is some notion of semantic anomaly in the internal structure of a noun group, then it can be detected in some cases from the semantic markers of that noun group while the procedure-meaning is being built (e.g. (1) and (2)). However, semantic oddity can also arise from subjects, objects, etc., failing to meet the selectional restrictions (or preferences) of the main verb, and this kind of semantic checking is more complicated. As McCawley (1968) observed, the selectional restrictions on a verb are really restrictions on the <u>referents</u> (in this model, the result-meanings) of the participants of the verb (subject, object, etc). If we try to operate entirely with procedure-meanings, verbs will have to have selectional restrictions which classify procedures, if at all. That is, instead of stating that "see" requires an object which is [VISIBLE] , we really need to say that it requires an object which is [EVALUATES-TO VISIBLE] , since this restriction will have to be true of the procedure-meaning, not of the result-meaning. (In the analogy suggested in the introduction, markers like [EVALUATES-TO T] correspond to Montague intensional types $\langle s,t \rangle$ (Montague (1970))). For each verb relation R, with argument domains D1,D2,....,Dn , (defined on the result set) there is an associated verb-meaning R' with argument domains D1' ,.....,Dn' , where Di' = set of procedures which evaluate to items in Di, so that verb-meanings are relations between procedures.

Secondly, selectional restrictions are generally a weak form of compatibility checking, since many noun groups (e.g. "something", "what it is", etc.) do not carry much information about their referents. McCawley (1968) comments that the verb "diagonalise" has a very specific selectional restriction on its object - the referent of the object should be a matrix. However, this restriction cannot be invoked unless the object noun group is specified in sufficient detail:

He diagonalised something. (8)

* He diagonalised his parrot. (9)

The only testing possible is of the form "<u>could</u> this term refer to an item which meets the restriction of the verb-relat-

ion ?" If the analyser has access to the result-meaning (as happened in Winograd's program) then it has full information; procedure-meanings, on the other hand, are often somewhat inscrutable.

One possible approach to processing of procedure-meanings is as follows. Winograd's semantic specialists operated on surface structures, combining meanings of constituents hierarchically (as in Katz and Fodor (1963) projection rules). As commented in 2. above, these rules also performed reference-evaluation, and produced a result-meaning as their output. Suppose we stipulate that these combining-rules generally operate on procedure-meanings, and that reference-evaluation occurs at some late stage (perhaps at the end of the sentence). If we wish to detect semantic oddity using selectional restrictions at an early stage in the analysis, before reference-evaluation occurs, then we need some way of making predictions about the semantic details of the referents. That is, the procedure-meanings must somehow display to the analyser some information about the properties of their results. For a common noun like "parrot" this may seem fairly straightforward - something referred to as "the parrot" should be predicted to be [ANIMATE], etc. But life is not so simple - there are "toy parrots", and "dead parrots". The internal structure of a complex expression may make non-trivial contributions to the predictions about the properties of the referent. Let us associate with each combining rule a "semantic classification rule" which states how the properties of the output item (a procedure-meaning) will be related to the properties of the input procedure-meanings. For example, the relative-clause semantic rule might have an associated classifying rule roughly as follows:

> The output will meet the selectional restriction of the verb-slot from which the relativising occurred. (10)

This is intended to express the fact that a clause like

> What I saw (11)

will refer to an item which is [VISIBLE] since the object of "saw" has been relativised, as the object of "see" has the selectional restriction [EVALUATES-TO VISIBLE]. Similarly, if we assume that "break" prefers a [PHYSOB] (Winograd's marker for a physical object) as its object referent, then (12) will have a referent which is a [PHYSOB]. (This ignores the slight problem that, since we are dealing with preferences, even this prediction is not cast-iron).

> What you broke. (12)

4. CONCLUSIONS

The aim of this paper has been to examine the semantic processing that was carried out in Winograd's program, and to point to certain modifications that may be necessary. The terminology used has implied that syntax and semantics are separate, classifications like "subject" and "object" are used rather than Fillmore-style deep cases (Fillmore (1968)). It might be felt that the difficulties raised here are consequences of this approach, and would not arise in a more obviously semantically based model. This is not the case, since the points mentioned here have arisen while attempting to work with a model in which the semantic rules have totally taken over the prominence previously afforded to syntactic rules, and in which deep case structures are used. The problems are inherent in the task of performing all processing (including reference evaluation) as the analysis proceeds.

The relevant points are as follows. Semantic anomaly cannot be decided once and for all at a local level and it is probably best handled by some kind of ordering or "preference" system. If we represent noun group meanings as procedures, then semantic processing requires environments in which to run those procedures. These environments may not be determinable at a local level, and so it may be impossible to find a result at once. This suggests that, in at least some cases, semantic rules should handle procedure-meanings rather than result-meanings. If we wish to carry out semantic preference at an early stage using procedure-meanings, we will need a device (such as the semantic classifying rules of Section 2b. above) to label procedure-meanings with information about their results.

Examining the limits of semantic processing is a necessary preliminary to attacking the problems involved. Asserting that Winograd has shown how to use semantic processing during sentence-analysis obscures the real contribution of Winograd's program - namely, providing the foundations of a framework in which substantive questions can be examined in detail.

ACKNOWLEDGEMENTS

This paper was written while I was supported by a Science Research Council research studentship. I should like to thank Alan Bundy, Hamish Dewar, John Knapman, Arnold Smith, Martha Stone and Yorick Wilks for comments on an earlier draft.

REFERENCES

Bach, E. & R.T.Harms (1968) Universals in Linguistic Theory. London: Holt Rinehart and Winston.

Fillmore, C.J. (1968) The Case for Case, in Bach and Harms (1968), 1-88.

Frege, G. (1892) On Sense and Reference, in Translations from the Philosophical Writings of Gottlob Frege, 56-78. Oxford: Blackwell.
Isard, S.D. (1974) What would you have done if ..... Theoretical Linguistics Vol.1, No.3, 233-255.
Isard, S.D. & H.C.Longuet-Higgins (1973) Model Tic-tac-toe (in Logic, Language and Probability (eds. R.J.Bogdan and I.Niiniluoto), 189-195. Dordrecht: Reidel.
Katz, J.J. & J.A.Fodor (1963) The Structure of a Semantic Theory. Language 39, 170-210.
McCawley, J.D. (1968) The Role of Semantics in a Grammar, in Bach & Harms (1968), 125-169.
Montague, R. (1968) Pragmatics, in Contemporary Philosophy (ed. R.Klibansky), 102-122. Florence: La Nuova Italia Editrice.
Montague, R. (1970) Universal Grammar. Theoria 36, 373-398.
Montague, R. (1972) Pragmatics and Intensional Logic, in Semantics of Natural Language (eds. G.H.Harman & D.Davidson), 142-168. Dordrecht: Reidel.
Ritchie, G.D.(forthcoming) Computer Modelling of English Grammar. Ph.D. Thesis, University of Edinburgh.
Wilks, Y. (1975) Preference Semantics, in Formal Semantics of Natural Language (ed. E.L.Keenan). Cambridge: Cambridge University Press.
Winograd, T. (1972) Understanding Natural Language. Edinburgh: Edinburgh University Press.

Aaron Sloman
Steven Hardy

GIVING A COMPUTER GESTALT EXPERIENCES

POPEYE is a vision program currently being developed by a small group at Sussex University. The aim is to explore the problems of interpreting messy and complex pictures of familiar objects. Familiarity is important because knowledge of the objects helps to overcome the problems of dealing with noise and ambiguities. Figure 1 gives an example. Pictures are presented to POPEYE in the form of a two-dimensional binary array, representing scenes containing overlapping letters made of "bars". Pictures are generated by programs either from descriptions or with the aid of an interactive graphics terminal. We are using POP2, the programming language developed for A.I. at Edinburgh University. However, we have found it useful to extend the language, and this paper describes some of the extensions. POPEYE's domain-specific knowledge will be described on another occasion.

POPEYE should process the picture in a sensible, flexible way, so that the main features to have emerged at any time can redirect the flow of attention. This applies at all levels. For instance, instead of doing the usual complete scan of a picture before starting higher-level processes, POPEYE builds histograms while doing the scan, and uses them to indicate the need to switch to something more important. But if higher level processes run out of things to do, the scan is continued. Similarly, the higher-level processes themselves may generate something still more important, in which case they have to be suspended. So, many different kinds of processes need to be able to coexist. Starting higher processes before lower level surveys are complete generates horrific problems to do with not jumping to conclusions too quickly, and being prepared to undo decisions, occasionally. However it seems that the human ability to make something meaningful emerge quickly out of a mess of data depends on this. To make it easier to write programs which generate and control many different processes we have implemented a "process" package, including scheduling facilities. These will be described below.

Another kind of problem is concerned with storage and retrieval of large amounts of information. For instance, the same item may be relevant in different ways in different contexts: sometimes a line-segment is relevant to a subproblem because of its orientation, and sometimes because of its loca-

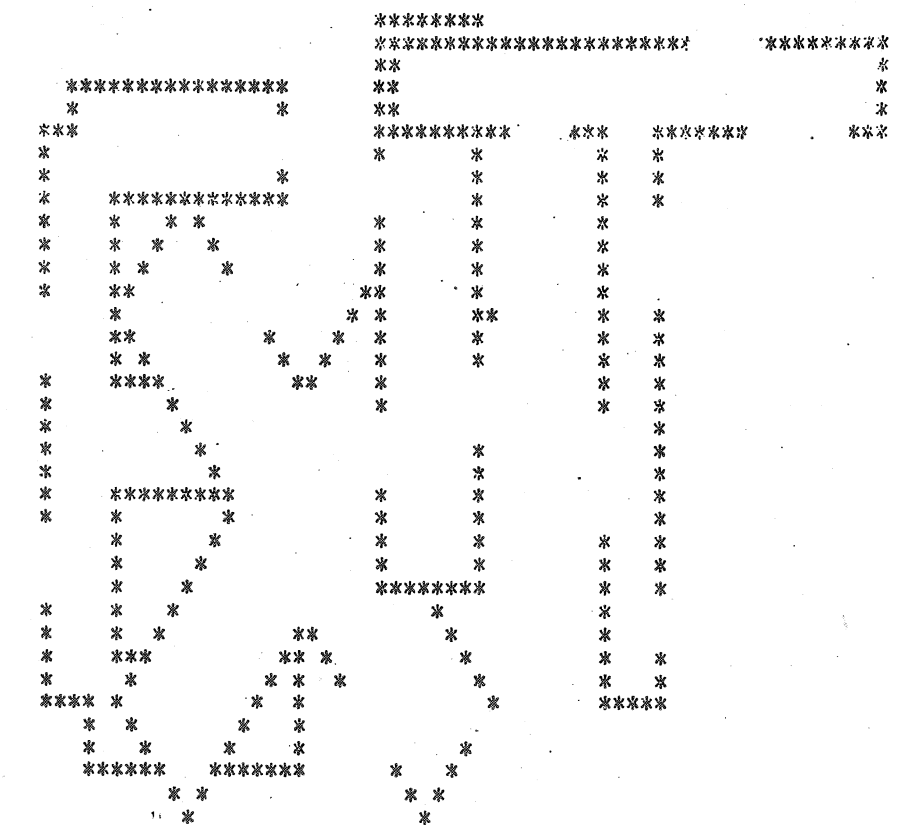


Figure 1.

This is a simple example of the kind of picture the program will be expected to analyse and interpret. Most people find that familiar letters, and possibly also a word, emerge very quickly.

tion in the picture. So a lot of cross-indexing is needed. Moreover, the program will eventually have to have a very large number of different concepts, relating to different kinds of structures and relationships which can be found in the picture and the scene, so an efficient retrieval system is needed for matching picture-fragments and scene-fragments with previously stored structural descriptions. To deal with this problem we are using a pattern-directed data-base package, with demons which can react to attempts to access or modify the data-base.

THE DATA BASE

The data base contains many features found in PLANNER-like languages. It is a modified and extended version of the HBASE system designed by Harry Barrow as a POP2 library package. The data base consists of a network of patterns, with an indexing mechanism for finding a pattern quickly, given a template for it. There is a POP2 library package for associating different values with the same data base item in different "contexts". The main extension to HBASE has been the provision of a "demon" mechanism. A demon is a procedure invoked by the occurrence of a specific type of event involving a pattern, e.g. when ASSERT, DENY or RETRIEVE is applied to the pattern. A "possibilities" list mechanism is provided, so that real or simulated data-base items may be generated and tried one at a time, if necessary. Generators use the process package, described below. Possibilities lists are themselves generators, so that explicit lists of possibilities don't have to be built up.

Eight standard types of pattern-invoked demons are provided, whose names should indicate their functions: PREASSERTED, IFASSERTED, PREDENIED, IFDENIED, PRENEEDED, IFNEEDED, ALREADY-ASSERTED, ALREADYDENIED. The last two types are for trapping redundant data-base alterations - especially useful for debugging. The system makes it easy to define and use further types of demons. For instance, the function RUNALL takes a pattern, and a list of types of demons, and causes all demons of the types listed, whose patterns match the given pattern, to be executed. This is used in the processes package. Processes may be sent "messages" in the form of patterns. Each class of processes uses a specialised set of demons to react to such messages. Later we shall illustrate the use of the data base in combination with the process package.

THE PROCESS PACKAGE

The process package consists of (a) a scheduler, (b) process-creating facilities and (c) a message-broadcasting system, which uses the data base. These will be described in that

order.

The scheduler is our attempt to overcome the following problem. POPEYE will be analysing a wide variety of picture fragments, yet wanting to switch attention to the most important features as they emerge. This suggests the need for some central intelligent process with an overview of everything happening, so that informed decisions can be taken. Yet no central administrator can manage all the complex different kinds of knowledge required for comparing different possible developments. We have therefore tried to decompose the scheduling task as follows.

The central scheduler knows only about the major categories of jobs which may be ready to run. These categories are listed in SCHEDULE. Lower-level processes know how to decide, on the basis of detailed evidence available to them, when to create a job, and which category to assign it to, or more generally which job-description to associate with it. For instance, processes concerned with linking line-fragments into larger clusters might be able to decide to create a hypothesis that a junction between two bars has been found. That hypothesis will be a process which may collect further evidence, and perhaps link itself with other bar-hypotheses to generate a letter hypothesis.

As each process is either initially created, or given some new evidence, it can put itself into an appropriate category of runnable jobs, leaving it to the central scheduler to decide which category is currently most important. So all the scheduler has to do is find the most important non-empty category, and select a job from it, either arbitrarily or by asking a manager for that category. By allowing schedulers themselves to be processes which can run for a while, then halt, then continue, we can allow some of the jobs to have their own local versions of the scheduler, with their own local SCHEDULE. Since POP2 does not provide a time-sharing facility, the programmer has to take care to design modules so that they don't run for too long. For instance, whenever a process does something which may have created or reactivated more important processes, then it should suspend itself soon after. The DETACH facility, described below, makes this possible.

The use of job-descriptions instead of numerical priorities has a number of advantages. For instance if the task changes (e.g. "find all vertical bars"), then the order of importance represented by the schedule can be altered on the basis of knowledge of which kinds of processes are relevant to the task. Further, if a relatively important process needs a specific low-level process to be run (e.g. a bar process needs a line-segment process to get information about junctions at its ends), then instead of juggling with priorities the first process can look in the appropriate job category, find what it

needs and run it.

The process-creating facilities make extensive use of "partial application", one of the features of POP2. A POP2 function can be partially applied to a set of arguments, forming a "closure". For example, the following function assigns X to be the second element of L.

```
FUNCTION PUTSECOND X L;  X -> HD(TL(L)); END;
```

If L1 is a list, we can make a closure which behaves like a function whose execution assigns the word "cat" to be the second element of L1, thus:

```
PUTSECOND(%"CAT",L1%) -> PUTCAT;
```

Every time PUTCAT is executed, it runs the function PUTSECOND in an environment in which X and L are given the two "frozen values" specified. Thus PUTCAT has both program and data stored within it. It can be executed, like a function, but it can also be accessed, like a data structure. Both the frozen values and the frozen procedure may later be altered. By giving a POP2 closure a pointer to itself, we can allow it to update its own frozen values. So it can behave like a process with a memory, unlike ordinary functions. Doing this in POP2 requires defining functions which are ugly and obscure. So we have defined POP2 "macros" which alter the syntax, hiding the ugliness. The central ideas come from Steve Hardy's POPCORN system, currently under development.

The most basic mechanism is the creation of a "process" consisting of a function and a variable-binding environment which, unlike an ordinary POP2 closure, will remember assignments to its variables. If FOO is a function in which the variables P Q and R are not declared as locals, then the brackets "/::" and "::/" can be used to create a process whose "manager" is the function FOO and whose environment uses the values of P Q and R at the time of creation.

```
FOO /:: P Q R ::/  -> PROC;
```

PROC is now a process which will run FOO each time it is executed. But if FOO assigns new values to P, Q or R, then they will be remembered for subsequent executions.

Such processes can be accessed from outside. For instance PROC("P") will produce the current value of P in PROC. Processes automatically get updaters, so that `4 ->PROC("P");` will change the stored value of P. ENVEVAL can be used to evaluate a function inside a process. Each process gets a local variable SELF, so the code for the manager function can include instructions involving SELF. The function SAVESELF allows a process to make a copy of its current state. The function SHOWSELF enables any process to print itself out in a neat format. The operation SETMANAGER may be applied to a

function or lambda expression and will give the current process a new manager function.

Similar effects could have been obtained using the state-saving functions of POP2. However, this would have been much more awkward to use, and much less efficient.

We have provided additional macros and functions, using the process-creating brackets, to provide features analogous to the "classes" of SIMULA67. We use the term "processmaker". This denotes a function which creates a process, does some initialising computations, such as setting up data-structures for the process, then returns the process. The process has the features mentioned above. Further, the macro DETACH, illustrated below, makes it possible to write code which can be executed, left for a while, then continued. The process-initialising instructions are distinguished by occurring between the words "INITIALISE" and "BEGIN".

Here is a simple example: a processmaker which produces generators for pairs of numbers.

```
PROCESSMAKER PAIRS XMAX YMAX;
 INITIALISE;
   VARS X Y;
 BEGIN;
   FOR X FROM 1 BY 1 TO XMAX DO
    FOR Y FROM 1 BY 1 TO YMAX DO
        CONSPAIR(X,Y)
        DETACH;
    ENDDO;
   ENDDO;
 TERMIN;
END;

VARS PAIRGEN;
PAIRS(5,7) ->PAIRGEN;
```

PAIRGEN is now a process which will generate a new pair each time it is executed, until it is exhausted, in which case it produces the POP2 "terminator" TERMIN.

A macro GENERATE, which uses DETACH, is provided to enable one generator to create and use others, on the assumption that an exhausted generator will produce TERMIN as its result. The following is a recursive processmaker which creates a generator for the atoms of a tree. (In POP2, HD and TL correspond to CAR and CDR of LISP).

```
PROCESSMAKER FRINGE TREE;
 INITIALISE;
 BEGIN;
  UNLESS TREE=NIL THEN
   IF ATOM(TREE) THEN TREE; DETACH;
   ELSE
        GENERATE FRINGE(HD(TREE));
        GENERATE FRINGE(TL(TREE));
   CLOSE;
  CLOSE;
 TERMIN;
END;

VARS TREEGEN;
FRINGE([A [B C] [[D] E F]])->TREEGEN;
```

TREEGEN is now a process which when first called will produce "A", then the next time "B", and eventually TERMIN. GENERATE used like this is rather inefficient, but illustrates the facilities.

Neither TREEGEN nor PAIRGEN takes arguments. However, a process manager may be a function which takes arguments, in which case it is easy for other processes to communicate with it, without using ENVEVAL. For instance we are currently experimenting with a class of processes which all use the following manager function:

```
FUNCTION PROCESSMANAGER MESSAGE;
  IF ISFUNC(MESSAGE) THEN MESSAGE()
  ELSE
   RUNALL(MESSAGE,DEMONTYPES)
  CLOSE;
END;
```

A process with this manager will run only if given an argument, MESSAGE. If it is a function it will be executed in the environment of the process, otherwise it is assumed to be a pattern which will invoke one or more demons of the types specified in DEMONTYPES. The data-base index is used to find relevant demons quickly. Different classes of processes may use different versions of DEMONTYPES. For instance, if some

have demons which know how to understand English, then they can talk to one another in English. By having two types of messages, functions and patterns, we cater for two cases. When communicating with a process, if you know exactly what you want it to do, then send it a function to execute. Otherwise send it a pattern and let it use its own expertise. By allowing processes to have their initial versions of DEMONTYPES set up by the relevant processmaker, we allow instances to "inherit" procedural attributes from their species. However, individual processes may modify their own versions.

The concept of a processmaker is still evolving. We are currently experimenting with ways of allowing assignments to process-variables to trigger suitable actions. Similarly, it is possible to attach "exit" demons to a process, which will automatically run whenever control is about to leave the process. Thus a process which needs to be careful about something, for a short time, can give itself such a demon, for as long as is necessary.

A process may be located at an "address" in the data base, or possibly at several addresses. The function STOREAT takes a pattern and locates the current process at the address specified. The address is a public description of some important facts about the process, e.g. its type and location in the picture, or maybe some of its relations to other things. Inside the process is the more detailed information it needs to do its stuff, but which it would be wasteful to have represented in the data-base index. We could have represented everything by data base items, and done without processes, but that would have too many disadvantages, of the sorts documented by Bornat, Brady and Weilinga in their contributions to this conference.

The message broadcasting mechanism is provided to enable processes to communicate with one another, using their addresses in the data base. A message may be either a pattern, which will invoke appropriate frame demons, or a word or a function. In the latter two cases it will be evaluated in the environment of each recipient. The target should be a description of the intended recipients. For instance

```
LAMBDA;
   IF SIZE>10 THEN PR(SEGMENTLIST) CLOSE
END
   -->«LINE $$DIRECTION ==>>;
```

causes every line whose orientation matches the current value of DIRECTION ($$ means "use current value"), no matter what its location (== will match anything), to print its segment-list if it contains more than 10 points.

```
<<NEWPOINT $$POINT>>
 --> <<LINE $$DIRECTION $$LOCATION>>
```

will tell the appropriate line process that a new point has been found for it. This should activate appropriate demons in the environment of that process. To postpone message sending until there's nothing more important to be done, use the following syntax:

<message> --> (<target description>, <job category>);

This will cause the sending of the message to be a job to be activated by the scheduler. The sender will presumably detach,

and hope for a reply later.

When one process runs another by calling it explicitly, the second can reply by leaving results on the stack, as sub-routines do in POP2. But when message broadcasting is post-poned, the sender may not be active when the message is re-ceived, so the stack cannot be used for replies. If a reply is needed, the message must include some data-structure which the sender can examine later. For instance,

```
VARS LETTERBOX; CONSREF(NIL) ->LETTERBOX;

LAMBDA LETTERBOX;
  IF SIZE > 10 THEN
    SELF :: CONT(LETTERBOX) ->CONT(LETTERBOX)
  CLOSE
END(%LETTERBOX%)
```

uses partial application to create a message which can be sent to a lot of lines. By looking at CONT(LETTERBOX) from time to time, the sender will discover which of the lines has a size greater than ten. Similarly a letterbox can be included as part of a pattern message. Incidentally this shows how pro-cesses which at first only know of one another by description can get direct pointers to one another.

This message-sending mechanism, combined with other POP2 features, such as interrupts and incremental compilation, enables the programmer to communicate with processes in much the same way as they communicate with one another. This is indispensable during debugging.

Here is an example showing how data base demons can use process brackets to save a portion of the environment in which they were created. Demons of type IFNEEDED are activated when a RETRIEVE command is unsuccessful. We want an IFNEEDED demon to try to answer a question, and if it fails, to plant an IF-ASSERTED demon which will watch out for the answer. If the answer turns up later, the second demon will record the fact, which may trigger off other demons, then kill itself. The problem is that the second demon may run long after its creator has exited, so that it needs to save relevant parts of its binding environment, using the process brackets. In the example, "%%" means use the current value of, and "%*" means give this variable a value during matching.

```
IFNEEDED <<SUM $*X $*Y $*Z>> ;
  VARS X Y  Z X1 Z1;
  IF X <1  THEN QUIT(FALSE)
  ELSE
   X-1 ->X1;   Z-1 ->Z1;
   IF RETRIEVE <<SUM  $$X1 $$Y $$Z1>> THEN
        SUCCEED()
   ELSE
        IFASSERTED <<SUM $$X1 $$Y $$Z1>>;
          PR('FOUND THE ANSWER');
          KILLSELF();
          ASSERT <<SUM $$X $$Y $$Z>>;
        END /:: X Y Z ::/
   CLOSE
END
```

A call of RETRIEVE<<SUM 3 5 8>> when <<SUM 3 5 8>> has never been asserted, will activate the IFNEEDED demon. If, in addition, <<SUM 2 5 7>> has never been asserted, which will produce a recursive call of the demon, and it cannot be proved, then the IFASSERTED demon, in the form of a process remembering the values of X Y and Z, will be added to the data base. If it gets triggered later, it will remove itself from the index and store the solution to the problem, which may activate other demons waiting for the solution. Demons can be given names, making it easy for one to access another and kill it if it becomes redundant. Thus chains of demons in the data base can provide some of the functions of a multiple stack mechanism. The lack of efficiency is perhaps compensated for by the ease of inter-process communication.

OVERVIEW

POPEYE sets a number of different processes going in parallel when presented with a picture to interpret, Some collect global statistics about the picture, some search for dot configurations suggesting lines. These may trigger off other processes, some deciding whether parallel lines should be linked to form "tubes", some keeping track of junctions, some trying to link "tube-sections" into larger structures, etc. This kind of "breadth first" approach is required mainly because, with a large amount of information available for analysis and interpretation, it may not be easy to decide what to do next, e.g. which configurations to look for, and where to look for them. Deciding between such alternatives itself requires analysis of evidence, and it will not be obvious what the important clues are, nor where they are. So initially many possibilities are sampled, until items both unambiguous and relatively important begin to emerge, such as a long line, an unambiguous clue to the location of a line, an aspect of the style of the picture,

or a set of linked fragments which uniquely identify a known letter. What counts as important will depend on the stage of analysis. The scheduler will notice the emergence of new important jobs and run them before others. This approach seems to be very similar to that used in the HEARSAY II system at Carnegie Mellon.

We are guided by several principles. One is to use descriptions wherever possible instead of numerical weights or priorities, so that the program has adequate information for taking decisions. Another is to select hypothese not on the basis of their support, or probability, but on the basis of their explanatory power (as recommended in Popper's philosophy of science). For instance, work on a large picture fragment rather than a small one, but work on a scene fragment rather than a picture fragment. But this requires a further principle, which is not to let any hypothesis be generated unless there is good reason to do so and one is not simultaneously generating large numbers of rival hypotheses. When there is no way of choosing between alternatives on the basis of current evidence, don't generate either. Instead there should be a description of what is common to both. Hope that either new detailed evidence will emerge to decide the issue (or look for it if you know it can be found quickly), or else global relationships between ambiguous, intermediate structures will enable larger, unambiguous clusters to emerge without combinatorial searches. For example, as Larry Paul's paper for this conference shows, global relations between a cluster of ambiguous limb-like regions may determine which are arms and which legs, when there's little hope of finding local details to discriminate them.

Making all this work requires the program to have a large store of concepts corresponding to various "intermediate" levels of structure, so that it never needs to take large leaps from what it knows to shaky hypotheses. In relation to dots and letters, intermediate concepts include "line segment", "bar", "bar junction". We view Grape's work as illustrating the importance of using intermediate structures between line-junctions and pictures of whole objects. This kind of expertise will work only in a "friendly world". If pictures are too noisy, or objects are piled up so that most things are almost entirely occluded, or if letters are juxtaposed so that gaps between them form too many spurious clues, then POPEYE, like a person, will get confused. However, there are cases where alternative hypotheses need to be coped with, for instance in pictures where there are quite large chunks with globally consistent dual interpretations. Geoffrey Hinton's conference paper suggests a way of using relaxation to deal with this.

Currently POPEYE manages to see important bars, and

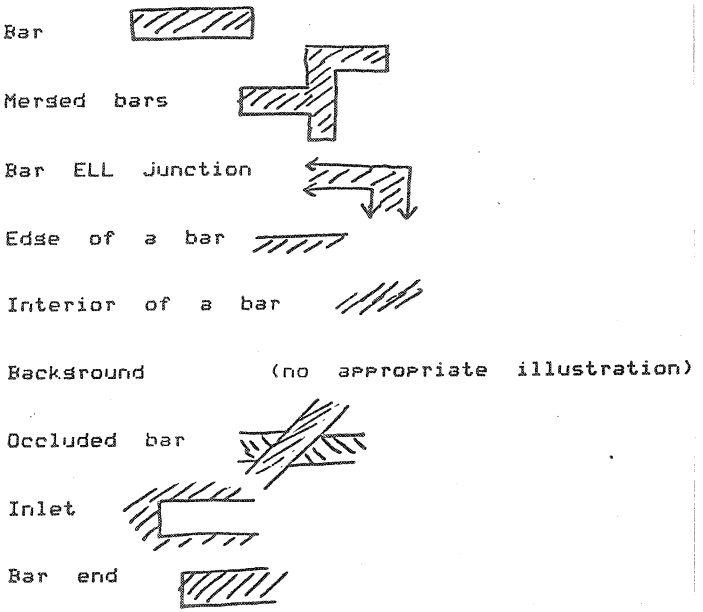


Figure 2

Some concepts from the domain of bar-scenes.

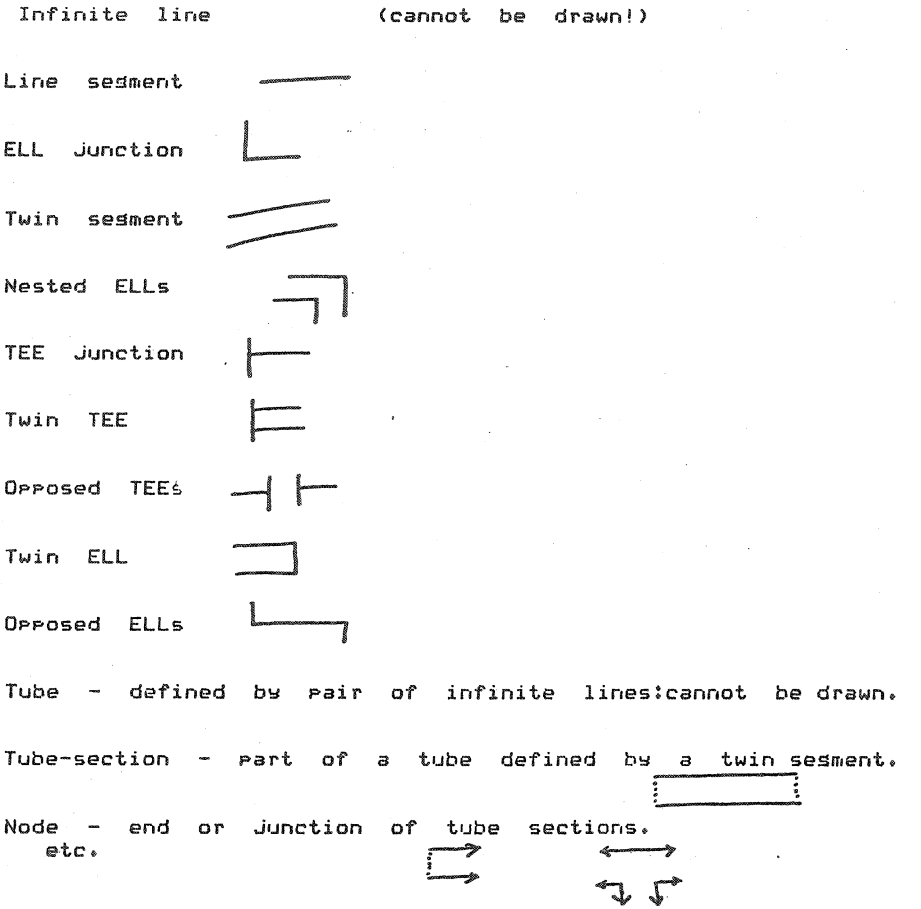


Figure 3

Some concepts from the line-picture domain.

junctions between them, without exploring all points and line-segments in the picture. A fair amount of positive or negative noise (spurious or missing dots) can be added without upsetting the process much. We have begun to work on the concepts required for dealing with significant clusters of bar fragments, so as to enable whole letters to emerge from the mess. The figures illustrate some of the concepts involved.

ACKNOWLEDGEMENTS

The POPEYE project is funded by the Science Research Council. Much of POPEYE'S domain-specific code is being written by David Owen, who joined the project in September 1976, followed by Geoffrey Hinton in January 1975. The idea of using histograms and other global picture descriptions to control processing came from the work of Max Clowes and Frank O'Gorman. The latter has been closely involved in our theoretical discussions. The design of the project also owes much to interactions with Sylvia Weir, Alan Mackworth, Mike Brady and Richard Bornat. The process package is influenced by Carl Hewitt's work on "actors", and the paper by Bobrow and Norman. Instead of standard POP2 we are using POP10, a dialect developed by Julian Davies and maintained on the Edinburgh PDP10 by Arnold Smith.

But for Pat Norton's speed, accuracy and patience this paper would never have been typed on time.

CORRIGENDUM

P246 ninth line from bottom should read ENVEVAL ("P", "PROC") and not PROC ("P").

RELEVANT REFERENCES

Bobrow,D. & A. Collins (1975) Representation and Understanding, Academic Press.

Bobrow,D. & D. Norman (1975) Some principles of memory schemata in Bobrow and Collins.

Clowes,M.B. (1971) On seeings things in Journal of Artificial Intelligence.

Davis,L.S., A. Rozenfeld. & S.W. Zucker (1975) General purpose models: expectations about the unexpected, Computer Science Technical Report, TR347, University of Maryland.

Erman,L.D., R.D.Fennell, V.R.Lesser & D.R.Reddy (1973) System organisations for speech understanding in Proc.3rd I.J.C.A.I. Stanford.

Grape,G.R. (1973) Model based (intermediate level) computer vision, Stanford A.I. Memo AIM-201.

Hewitt,C., P.Bishop & R.Steiger (1973) A universal modular ACTOR formalism for A.I. in Proc.3rd I.J.C.A.I., Stanford.

Hinton,G. (1976) Using relaxation to find a puppet, this conference.

Magee,B. (1974) Popper, Fontana Modern Masters.

O'Gorman,F. & M.B. Clowes Finding picture edges through collinearity of feature points in Proceedings 3rd I.J.C.A.I. Stanford, 1973.

Paul,J.L. (1976) Seeing puppets quickly, this conference.

Shirai,Y. (1975) Analysing intensity arrays using knowledge about scenes in Winston.

Stansfield,J.L. (1974) Active descriptions for representing knowledge in Proceedings AISB summer conference.

Reddy,D.R., L.D.Erman, R.D.Fennell & R.B.Neely The hearsay speech understanding system in Proc.3rd I.J.C.A.I., Stanford (1973).

Turner,K.J. (1974) Computer perception of curved objects using a television camera in Proceedings A.I.S.B. summer conference.

Waltz,D.(1975) Understanding line drawings of scenes with shadows in Winston.

Winston,P. (1975) The psychology of computer vision, McGraw-Hill.

Hardy,S. The POPCORN75 reference manual, forthcoming.

A.M. Stanier

PLANNING TO MAKE TRICKS AT BRIDGE

Playing contract Bridge, there is a bidding stage, the first card is led, the dummy-hand is exposed, and the declarer then knows exactly what ti (see footnote) has to play with to make tis contract. It is usual and advisable, to pause for thought at this point, and make a rudimentary plan as to how the contract should be made. This paper examines the task of making such a plan, and describes a program which attempts it.

The program is the second stage in a three-stage program which plays the entire game. The first stage (which bids a hand, and interprets all the other bids), is described in (2). The third stage, which plays the cards, will hopefully be written in the near future.

This paper assumes a reasonable knowledge of Bridge; this is regrettable, but unavoidable. The A.I. philosophy behind this research has been that to produce an intelligent program, it is necessary to give it all the benefit of human knowledge in the particular field.

Thus the bridge program has coded into it a great deal of Bridge-specific knowledge. It would be impossible, in this paper, to describe the game in sufficient detail for an absolute beginner to appreciate it. A reasonable introduction is in (1). This attitude, that one ought to deploy all the knowledge at ones disposal, is accepted in fields such as language understanding, or vision, probably because most people have a deeply ingrained appreciation of the tasks involved: I wish to apply what appreciation I have of Bridge.

The first section of the paper describes the task to be tackled, using an extended example, in which we bring out the

---

* Impersonal Pronouns
English contains no elegant way of expressing impersonal pronouns. Early papers would use the word 'he' at this point, thus making all Bridge players male. Lately, to balance out the effects of male chauvinism, some writers have taken to using 'she' in such contexts, thus making all players female. To avoid this, and to avoid the cumbersome 'he or she', we use the following system of impersonal pronouns: TI = he or she TIS = his or hers TER = him or her. These pronouns were first suggested by Cullen, Giles, Lloyd and Mooney, and are recommended as standard usage.

various schemas used by human players to make tricks. This example is given in detail, which may seem excessive: this is necessary, as there is a great deal of detailed reasoning underlying the choice of schemas, which we wish to formalize. The second part of the paper describes the operation of the program, again using an extended example, and shows how it discovers possible schemas, and chooses from these a best plan.

STATEMENT OF THE PROBLEM

After the bidding sequence, the declarer knows the trump suit if any, the number of tricks needed to make the contract, and - if there has been a contested auction - ti will also have some indication of the distribution of the missing cards. Ti will also have seen the lead card, and will, perhaps, have been able to make deductions from it.

We contend that the declarer builds up a plan for playing the cards by recognizing certain schemas for producing tricks, and by considering these schemas, and tis knowledge of the interactions between schemas, gathers together a self-consistent set of instantiated schemas sufficient to satisfy the contract. The difference, in our view, between an expert player and an average one is the number of schemas recognized, and the perception of interactions between them.

Example

Suppose West is the declarer, after the following bidding sequence.

N	E	S	W
3D	DBLE	NO	3S
NO	4S	NO	NO
NO			

The card led by North is the diamond King, and the hands in the partnership are

S: A K J 9	S: 1Ø 7 5 4 3 2
H: K 1Ø 9 8	H: Q J
D: Q 6 2	D: J 8
C: 4 2	C: A K J

West now has the task of taking a minimum of 10 tricks with spades as trumps. From the bidding, ti can tell various things about the hidden hands. Norths opening bid of 3D may be assumed to be conventional, and shows at least 6 diamonds with

no aces or kings in any other suit. Therefore South probably holds the missing ace of hearts (note that this is only probably: North may have been bluffing when ti opened. This is highly unlikely (as North would probably not open the bidding with a false bid, since ti would fool tis partner as well as opponents), but it is possible. But should West want to place the HA, then a priori, ti should assume that it is held by South). The lead of the king of diamonds indicates that North probably also holds the ace.

West should then consider the suits in order: In spades: the partnership holds a total of ten cards, including the ace and king. Thus the opposition hold only three spades, and so it is possible that six spade tricks can be taken. The missing cards must be split between North and South 0-3, 1-2, 2-1, or 3-0. If it happens that North holds all three missing cards, there is no way to prevent ter taking the queen: any other distribution can be cleared. The play of the ace will reveal the distribution (at least, it will find out whether the cards are in fact split 3-0 or 0-3, in which case the void player must discard: should both follow suit, it is not possible to infer who holds the third, but it is not necessary to know). Suppose both opponents do follow: there is then only one outstanding trump, which will have to be played when West plays off the king. There will then be 4 cards in the East hand, which will all win, as any card that might have beaten them has been drawn. Suppose, however, that South does hold all three missing cards, this will be revealed when North discards. If the East hand is then entered, and a low spade played, South is trapped: should ti play the queen, the king will take it, promoting the jack. But should ti not play the queen ti is known to hold, West merely plays the jack to win the trick. This procedure (a finesse) is frequently used to establish low cards. Thus, for spades there are three mini-plans which may be considered: the play of the top two cards, a possible finesse, and running any long cards which are not needed for ruffing. Since spades are trumps, it will not be necessary to protect these mini-plans against being ruffed away. However, it is possible that plans in other suits based on ruffing will vitiate the long tricks plan in spades, as each ruff will remove a low spade from the long trick list.

In hearts: the partnership must lose a trick to the ace, but can then win three top tricks. Thus the mini-plan for hearts is to clear the missing cards, then run the suit. It is obvious that this mini-plan will fail if the suit is being ruffed by either of the opponents. Thus it is probably advisable to clear trumps if possible before attempting this plan.

In diamonds: the partnership has problems. North probably holds AKxxxx or longer, so South holds at most two low diamonds and will thus ruff the third round. East can ruff high, but

there remains the danger of an over-ruff should South hold the queen of spades. Thus the contract is in danger of being lost on the first 4 tricks: king of diamonds, ace of diamonds, low diamond ruffed and over-ruffed, ace of hearts. To avert this, West must avoid a third round of diamonds by throwing the queen. Hopefully, North will switch to another suit, thinking West is void. Note that this requires West to have an idea of North's plan: North will probably have a plan based on playing top diamonds, then lead a low diamond to be ruffed. West must anticipate this mini-plan.

This plan may well go wrong; should North hold 7 or more diamonds, South will be discarding and giving North a full count of the diamonds. West could try to avert this by throwing the queen on round one, but North will probably not believe West started with a singleton, and will play a second diamond round anyway. Finding the deception, ti will then play the third round because West doesn't want ter to. Alternatively North may fall for the pløy, and switch to a heart, won by Souths ace. South then returns tis last diamond, and thus asks for the killing diamond return.

On balance West is best advised to throw the queen on round two, hoping that North made a pre-emptive bid holding AKxxxx in diamonds and that South does not signal for a third diamond round.

In clubs: the partnership holds the ace and king, and the possibility of a finesse should one be needed. However, the finesse almost certainly won't be needed, as opposite the AK is a holding of xx, so after playing off the top cards, the jack may be ruffed.

So West can build up a composite plan of (1) persuading North to switch from diamonds on the third round by throwing the queen on round 2. (2) winning the return (or Souths return should North switch to a heart won by Souths ace), making one trick, then:

Drawing trumps (making 5 tricks)
Forcing out the ace of hearts if it hasn't already appeared
Winning the second return (another 1)
Taking the top clubs (another 2)
Ruffing a club (another 1)
Running the hearts
Running the spades.

West can expect exactly to make the 10 tricks needed: 6 spades, 2 top clubs, club ruff, and a heart. Ti will lose the DA, DK and HA.

In the above discussion, we may distinguish 5 distinct schemas for producing tricks:

(1) Taking top tricks
(2) Ruffing

(3) Establishing the low cards in a suit
(4) Finessing
(5) Losing tricks to missing top cards, to win low ones later.

It may, however, not be possible to use all of these schema on any given suit, as there will be interaction between the separate schemas: both between schemas in the same suits, and schemas in different suits, especially the trump suit - for example, should the plan for clubs involve establishing long cards, discards will have to be found in the other hand: the discards cannot take part in any other schema. One must, therefore, consider the pre- and post-conditions of the schemas

TOP TRICKS

Top tricks are tricks won with master cards - ace, king, etc. downwards in continuous sequence such that the partnership holds all cards capable of bearing the master. The preconditions for taking such tricks are either that the lead is in the hand with the master and there is a suitable card to play with it in the other hand, or that one is in the hand opposite the master, and has a suitable lead. The postconditions (apart from the obvious one that the cards will have been played) is that the lead is now in the hand that held the master. It sometimes happens that, since a card from both hands is needed, some masters cannot be made and have to be thrown on other masters. But more often, there is an order of taking the masters that avoids this: for example, should the partnership hold, in some suit

A K J Q 4 2

Then, although the partnership holds four masters, only three can ever be made. If the holdings were slightly different, such as

A K J 4 Q 2

Then, while the naive way of playing ace, king, etc. downwards will only take three masters (the queen would have to be played on the king), a re-ordering to take the queen first, playing the 4 on the trick from the opposite hand, will then take 4 top cards.

RUFFING

The prerequisites for being able to take tricks by ruffing are that

(1) the contract is not being played in notrumps.
(2) the suit in question is not the trump suit.
(3) one, and only one, of the hands in the partnership is void in the suit, and has at least one spare trump.

(4) the lead must be in the hand with cards in the suit

A card in the suit is led, and a trumps played, taking the trick. Thus the post-condition of a ruff is that the lead is now in the hand void in the suit, so to repeat the ruff, one must cross back to the leading hand.

Ruffing tricks may well affect the plan for dealing with trumps - as, for example, the ruff will remove a low trump, that could perhaps have taken a trick anyway, by being established as a long card. Should this occur, it is probably better to use a non-ruffing schema.

Should neither hand in the partnership be void in the suit, ruffing chances may still be present, although the suit must be cleared first to create a void. This may involve losing a trick to do so, and again, a non-ruffing schema would be preferred.

Ruffing should also be done before clearning trumps - this imposes a linear ordering on the execution of schema.

ESTABLISHING THE LONG TRICKS

Suppose the partnerships holdings in some suit is:

A K 8 6 5 2 Q 9 7

There are three obvious top tricks. Taking them will probably establish three more tricks: the two hidden hands contain 4 cards in the suits, and, except in the unlikely event of their all being in one hand, the top three will drive them out. Thus the 8, 6 and 5 will win tricks, as no card is left to beat them. There are thus, probably, three long tricks available in the suit, and even in the worst case, there are two.

The pre-condition for long tricks are a long suit with sufficient masters to take out most (or preferably all) of the missing cards, and no outstanding trumps. This second is a sine qua non of long tricks: whereas it is possible, although perhaps undesirable to attempt any other schema while there are trumps outstanding, the establishing of long tricks, relying as it does on voiding the opponents in the suit, forces the drawing of trumps first. There is also a pre-condition that one must have possible discards in the short hand.

FINESSING

Finessing is the process of taking a trick with a card which is not a master, due to the favourable location of the cards that could beat it. There are two distinct forms, with differing advantages and disadvantages: consider this holding

A Q 9 4 2 J 10 3

Should the king be in the hand over A Q, nothing can be done,

but should the ace-queen be over the king, the king is useless. The first form of finesse involves playing the 3, and playing the queen if the king hasn't appeared. This wins a trick if the king is favourably placed. The second form of finesse involves playing the jack or 10, then throwing the 2 unless the king has appeared. This has the advantage that the finesse can be repeated should it work the first time, since the lead stays in the leading hand: on the other hand, it has the disadvantage that the holder of the king, knowing it to be useless, will play it anyway, thus removing two of the partnerships high cards at the cost of one useless card to ter.

With the holding above, three finesses are possible, and should the cards split no worse than 4-1, and the king be favourably placed, the holding will bring in 5 tricks.

As might be imagined, the pre-conditions for a finesse are somewhat complex to define. The partnership must hold a master card. The opposition must have the card below this, and the partnership the card below that. (This ignores the possibility of a deep finesse, one that relies on the favourable location of more than one missing card). Should the master and the card below the missing card be in the same hand, the opposite hand must have a lead card in the suit: should they be in opposite hands, the master hand must have a low card in the suit.

FORCING LOSERS

The fifth schema we recognise is the situation where the opposition holds master cards in the suit. Thus declarer must lose tricks in this suit (or not tackle it at all). Cards must be played to force out the masters, in the hope of being able to make low cards later. This schema should only be attempted as a last resort.

THE PROGRAM

The program initially finds which top tricks it could take, and any contradictions in the planned order of taking them, as an example, consider this deal:

S: A K 10 7	S: Q J
H: A Q 7 5 4 2	H: J 10 3
D: 4 2	D: A Q J 10 3
C: 3	C: 10 4 2

Declarer is North, in the contract of 5 hearts, after this bidding sequence:

N	E	S	W
1H	3C	DBLE	NO
3S	NO	5D	NO
5H	NO	NO	NO

and the lead of the diamond king.

The program generates a list of its top tricks, and tags each master with the card it intends to play on that card: since it must play such a card on each round, it does not consider more than N masters in each suit, where N is the length of the longest hand in that suit.

The two lists are:

MASTER	OTHER CARD
Diamond Ace	Diamond 2
Diamond Queen	Diamond 4
Diamond Jack	NONE
Diamond Ten	NONE
Heart Ace	Heart 3
Spade Ace	Spade Jack
Spade King	Spade Queen
Spade Queen	Spade 7
Spade Jack	Spade 10

Any card which appears in both lists represents a contradiction These contradictions may be removed by swapping round the entries in which the card appears twice, and setting the 'other card' field in which it appears to NONE. This produces a list of top tricks with no contradictions, although the other cards are a little eccentric.

MASTER	OTHER CARD
Diamond Ace	Diamond 2
Diamond Queen	Diamond 4
Diamond Jack	NONE
Diamond Ten	NONE
Heart Ace	Heart 3
Spade Jack	Spade 10
Spade Queen	Spade 7
Spade King	NONE
Spade Ace	NONE

This is then a workable list of master tricks. The tricks need not necessarily be taken in the order listed: indeed, since drawing trumps has a high priority, it is likely that the order will not be as listed, but no trick may be taken while there is an untaken trick in the same suit earlier in the list.

The program next checks whether it has already found a suitable plan: it may not need to make any more tricks. The

program will produce simple plans quickly (as do human players). Difficult plans require more processing. It could be that there are enough top tricks to satisfy the contract (although it would be a very poor contract). If it has not yet found enough tricks, it proceeds to examine the suits in order, to discover schemas.

Should the partnership not hold a master in the suit, any schema would involve losing tricks. Thus a schema to clear the suit is generated, together with data as to how many rounds must be forced. If it does hold masters, a positive schema can be used.

The program next checks for ruffing chances, should the two holding differ in length, and the suit not be the trump suit, and the contract not be in notrumps, a ruffing chance exists. A schema is recorded, with notes on the number of ruffs available, and the number of losing rounds needed to establish the needed void.

Next, the possibility of drawing out all the missing cards to establish the long tricks is examined. If this is reasonable, a schema to do so is added, with information on the number of long tricks available in the best and worst cases.

Lastly, finesses are tested for. At the moment, the program is only capable of one-level finesses: deep finesses are beyond it. Should a possible finesse be found, its schema is added, with a note of the required lead, and an estimate of the likelihood of success. This estimate is based on the believed distribution of missing cards supplied by the bid-interpretation routines in bidder.

For example above, the mini-plans found are:

```
Clubs - Ruff 2 tricks, after losing 1
Diamonds - Ruff 3 tricks, after losing 1
           Lead the 2 and take the finesse.  Chance of success
           is 1.0.
Hearts - At least 2 long tricks, and at most 4.
         Lead the 3 and take the finesse. Chance of success
         is 1.0.
Spades - Ruff 2 tricks, after losing 0.
```

Having thus generated the schema, a 'best plan' is found by repeatedly testing if enough tricks are available by applying schemas in order. First the number available using just long tricks is considered (pessimistically, the program uses worst-case expectations for this: it can always make overtricks, but it may not always be able to recover from a bad split if it creates a plan relying on an even split that isn't there). If not, it considers long tricks, plus ruffs, etc.

Finally, the best plan is re-ordered so that ruffs are taken first, and a flag inserted in the plan where all ruffing schemas are finished. This flag marks the point at which trumps may be,

and probably should be if possible, drawn. This should ensure a normalized plan that protects itself against its schemas being disturbed by ruffing.

For the example, this best plan is

Drawtrumps Longtricks hearts 4 2

This is a very simple plan, meaning that in addition to taking the top tricks, there are two necessary mini-plans: first that of drawing out all the missing trumps (note that the plan found does not involve any ruffing, so drawing the trumps in no way inconveniences the program), then attacking the hearts for the long tricks it knows are available.

The resulting plan is very crude: deliberately so. This plan will be interpreted by the playing routines, which will take into account the vast amount of additional information discovered as the cards are played. Too detailed a plan at this stage would reduce the flexibility of the playing routines to take advantage of this additional information. For this reason, the toptricks are not included in the plan: since they may nearly always be taken, they may be kept in reserve, and used, for example, to cross from one hand to another to help carry out the schemas.

REFERENCES

Hervey, G.F., (1973) The Hamlyn Illustrated Book of Card Games 125-144. Hamlyn Publishing Group.

Stanier, A.M., (1975) BRIBIP: A Bridge Bidding Program, Advance papers of IJCAI-4, Vol. 1.

Olga Štěpánková
Ivan M. Havel

INCIDENTAL AND STATE-DEPENDENT PHENOMENA IN ROBOT PROBLEM SOLVING

After years of development of specific methods and programs in artificial intelligence it is worthwhile to assume a more theoretical viewpoint and try to use mathematical tools for unifying, generalizing and comparing the work that has been done up to now. Such attempts may eventually result in a "mathematical theory of artificial intelligence" which would be able to formulate and prove "metatheorems" about various methods used in AI, as well as to yield "metaheuristics" helping to discover good ideas for new methods.

The aim of our contribution is to discuss a particular attempt for a unifying formal approach to robot problem solving, especially to methods based on predicate-calculus representations (as e.g. STRIPS) and predicate-calculus reasoning (as e.g. the situation calculus). We have proved some metatheorems elsewhere (Štěpánková and Havel 1976) and in the present paper we demonstrate how our formal approach may suggest certain new ideas and extensions.

1. THE IMAGE SPACE

Our formal structure for robot world representation is based on the idea of an "image space", a collection of world images (or shortly images) of the same role as the world models in STRIPS (Fikes and Nilsson 1971; we have reasons for avoiding the term "model"). Formally an image is a formalized theory (or just a set of well-formed formulas, its axioms) in a first-order predicate calculus with a fixed language; intuitively it expresses a selection of assertions about a particular state of the world.

It is convenient to take apart axioms that represent state-independent facts and to group them into a single theory T_I, called the core (of the image space I). A particular image T is then obtained by adding a specific axiom , say F, to T_I;

we write $T = T_I[F] = T_I \cup \{F\}$.*

The image space is defined implicitly as a pair

$$I = \langle T_I, \Phi_I \rangle$$

where T_I is the core of I and Φ_I is a set of operator schemata of I. An operator schema $\varphi \in \Phi_I$ may be understood just as a name of one of robot's capabilities, e.g. "push", "goto", etc. With each $\varphi \in \Phi_I$ we associate a pair $\langle C_\varphi, R_\varphi \rangle$ of formulas called the condition of φ and the result of φ, resp. The free variables in C_φ and R_φ (if there are any) are called the parameters of φ. We write φ also in the form $\varphi[x_1, \ldots, x_m]$ to exhibit the parameters $x_1, \ldots, x_m$ of φ. We then obtain operators of I as instances of operator schemata by considering a variable-free term (in particular a constant) instead of each parameter and making the corresponding substitution in C_φ and R_φ. If $\varphi \in \Phi_I$ and ψ is an instance of φ, we denote by C_ψ and R_ψ the obtained instances of C_φ and R_φ, resp. The operators represent particular actions of the robot, e.g. "push box a from room k into room l".

The application of an operator ψ can be explained as follows: if $T \vdash C_\psi$ (C is provable in T) for an image T then ψ is applicable in T and yields a new image $T_I[R_\psi]$. Thus the problem solver can in its imagination wander through the image space until it finds a path (a sequence of operators) that solves the problem, i.e. leads from an initial to a goal image. Formally a problem is specified by a pair $\langle X, Y \rangle$ of variable-free formulas; the initial image is $T_I[X]$ and a goal image is any extension T of T_I such that $T \vdash Y$.

2. OPERATORS WITH INCIDENTAL PHENOMENA

The image space in its "pure" form described above is suitable and more or less sufficient for theoretical investigations (cf. Štěpánková and Havel 1976). It is also relevant to some questions suggested by Simon (1972, p. 415). However, from the point of view of practical applications, it is not very

---

* If convenient we treat a finite set of formulas as a single formula -- their conjunction.

satisfactory for two main reasons. First, the specific knowledge comprised in a new image depends only on the operator result and all other specific facts known in the previous image are lost even if they may not be influenced by the operator (this leads to the well-known frame problem). Second, all known side effects of any operator ψ, whether relevant to the problem in question or not, have to be permanently included in the formula R_ψ, and moreover, if these side effects have their own extra conditions, the operator has to be split, in general, into several new operators with distinct conditions C_ψ.

These drawbacks can be avoided by associating with each operator schema $\varphi \in \Phi_I$ a special well-defined (syntactical) procedure that generates -- or recognizes -- certain pairs of formulas $\langle A, B \rangle$, called the incidental phenomena of φ. The formulas A and B are assumed to have no free variables except possibly the parameters of φ and if an operator ψ is an instance of φ we obtain the incidental phenomena of ψ by substituting the corresponding variable-free terms for these parameters in incidental phenomena of φ.

The application of an operator ψ is now defined as follows. If

$$T \vdash C_\psi \ \& \ A$$

for a given image T and an arbitrary incidental phenomenon $\langle A, B \rangle$ of ψ, then the application of ψ yields a new image

$$T_I [R_\psi \ \& \ B] .$$

Any incidental phenomenon $\langle A, B \rangle$ represents a possible side effect changing the world (if A an B are distinct), or an element of the "frame" (if A equals to B)*, and can -- but may not -- be taken into consideration during the plan formation according to the problem solver's knowledge of the particular problem and his anticipation of useful facts for later stages of planning.

The collection (let us denote it by Inc_φ) of all incidental phenomena for a given operator (schema) φ, in practice the procedure for their recongition, depends on the nature of

---

* The "image space with frames" of (Štěpánková and Havel 1976), Sec. 7, considers only the latter case.

φ , on the environment, and on the expected class of problems. Typically it is specified by certain syntactical restrictions on participating formulas (cf. Example 1 below). To allow using more incidental phenomena at the same time it is useful to require that for any $\langle A, B\rangle$ and $\langle A', B'\rangle \in \mathrm{Inc}_\varphi$ we have also $\langle A \,\&\, A', B \,\&\, B'\rangle \in \mathrm{Inc}_\varphi$.

Let $\langle X, Y\rangle$ be a problem in I. We define the (straight-line) plan for $\langle X, Y\rangle$ as a sequence $\gamma = (\psi_1, \psi_2, \dots, \psi_n)$ of operators for which there exist incidental phenomena $\langle A^{(i)}, B^{(i)}\rangle \in \mathrm{Inc}_{\psi_i}$ $(i = 1, \dots, n)$ such that

(i) $\quad T_I[X] \vdash C_{\psi_1} \,\&\, A^{(1)}$;

(ii) $\quad T_I[R_{\psi_i} \,\&\, B^{(i)}] \vdash C_{\psi_{i+1}} \,\&\, A^{(i+1)}$;

(iii) $\quad T_I[R_{\psi_n} \,\&\, B^{(n)}] \vdash Y$.

The pure image space of the previous section comes out as a special case with $\mathrm{Inc}_\varphi = \{\langle \mathrm{true}, \mathrm{true}\rangle\}$ for all $\varphi \in \Phi_I$.

Let us illustrate our formalism on a simple example.

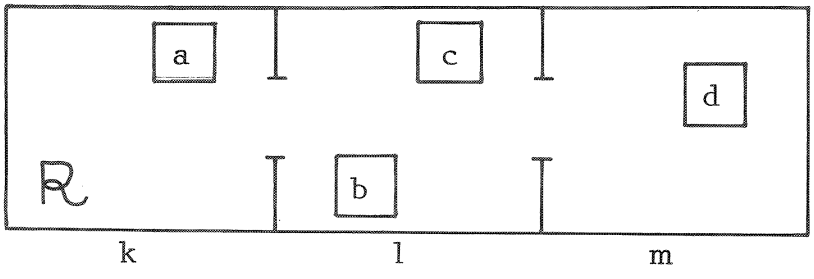


Figure 1

Example 1. The robot's world consists of three rooms k, l, m (all connected) and four boxes a, b, c, d distributed in the rooms according to figure 1 (formally described by the predicate IN, e.g. IN (b,l)). Robot itself is also in one of the rooms (ROBOT IN (k)) . The state-independent properties of the environment like $\forall x \exists y\, \mathrm{IN}(x,y)$ or $\mathrm{IN}(x,y_1) \,\&\, \mathrm{IN}(x,y_2) \longrightarrow y_1 = y_2$ compose the core theory T_I. There are two operator schemata:

push [x,y,z] (robot pushes box x from room y into room z) with condition IN (x,y) & ROBOT IN (y) and result IN (x,z) & ROBOT IN (z) ;

goto [y,z] (robot goes from room y into room z) with condition ROBOT IN (y) and result ROBOT IN (z)

As the incidental phenomena of both operator schemata we take first of all the frame: all pairs $\langle A, A \rangle$ where A is any formula without an occurence of the predicate ROBOT IN and, in the case of push [x,y,z] , also of the atomic formula IN x,y ;(thus e.g. $\langle$ IN (b, 1) , IN (b, 1)$\rangle$ is an incidental phenomenon of push [c, 1, m] but not of push [b, 1, m]) .

It can be easily seen that in such an image space one can solve the problem $\langle X, Y \rangle$ where X specifies the position of boxes and the robot as in figure 1 and the goal Y has the form

$$\exists v \ (IN (a,v) \ \& \ IN (b,v) \ \& \ IN (c,v) \ \& \ IN \ (d,v)).$$

An example of a solution sequence is

$$\gamma = (\text{push}\,[a,k,1] , \text{goto}\,[1,m] , \text{push}\,[d,m,1]).$$

Note that the incidental phenomena were necessary in order to use the fact that the boxes b and c remained in their original position.

Another example of an incidental phenomenon of push [x,y,z] may be

$$\langle y \neq z \ \& \ \forall u \ (IN (u,y) \longrightarrow u = x), \quad EMPTY (y)\rangle$$

(a room remains empty when the robot pushes away the last box). While useless for the above problem it helps to solve a problem with the goal, say, EMPTY (1).

The described idea of using incidental phenomena suggests a new interesting research topic: while the usual application of heuristic methods in problem solving concerns the selection of operators (as e.g. the GPS-like strategy in STRIPS), here one can also investigate special heuristic rules of controlling the flow of information from one image into another by a clever choice of incidental phenomena. Note that it is not apparent from the final plan, which particular incidental phenomena were used: they just serve as catalysts for planning.

3. RELATIONSHIP TO THE IDEA OF STRIPS

To give a rigorous logical meaning to the set-theoretical operations in STRIPS we have to agree on a certain class of basic formulas (for instance all literals) which may be used as elements of world descriptions. Any operator φ specified in STRIPS by the triple C_φ (precondition), Del_φ (the set of basic formulas to be deleted) and Add_φ (the set of basic formulas to be added -- cf Fikes and Nilsson 1971) can be in the image-space formalism specified by the condition-result pair $\langle C_\varphi, R_\varphi \rangle$ where R_φ is the conjunction of formulas in Add_φ, and by incidental phenomena of the form $\langle A, A \rangle$ where A is any basic formula not in Del_φ, or a conjunction of such formulas.

In general, every STRIPS problem description can be in this way converted into an image space I such that the solution obtained by STRIPS to a problem is also a plan (for the same problem) in I.

Conversely, let us define a STRIPS-like image space as any image space with the set of incidental phenomena including only pairs of the form $\langle A, A \rangle$ and closed under conjunction (the set of all formulas that do not participate in any incidental phenomenon of φ may then serve as an analogy to Del_φ). For any image space $I = \langle T_I, \Phi_I \rangle$ one can construct a STRIPS-like image space $\hat{I} = \langle T_I, \Phi_{\hat{I}} \rangle$ such that $\varphi' \in \Phi_{\hat{I}}$ iff there exists $\varphi \in \Phi_I$ and $\langle A, B \rangle \in Inc_\varphi$ such that $C_{\varphi'} = C_\varphi \,\&\, A$ and $R_{\varphi'} = R_\varphi \,\&\, B$ (thus $\hat{I}$ may have much larger and often infinite set of operator schemata). It can be shown that a problem has a solution in I iff it has a solution in $\hat{I}$.

As concerns the class of tractable problems, the image space is thus, in the above sense, equally powerful as STRIPS. Its conceivable advantage can be found in its generality (lesser dependence on a specific problem) and in the mentioned idea of selective choice of facts to be remembered from one image to another and thus giving the possibility of saving some memory space.

4. THE SITUATION CALCULUS AND BRANCHING PLANS

The question of relationship of STRIPS-like methods to historically older and more logically oriented situation calculus was rised several times (for instance by Nilsson (1971),

pp. 211-212). Due to the untractability of the frame problem the SRI group has after some experiments abandoned the situation calculus completely, while other authors (as Kowalski) strongly argue in favour of predicate logic as a tool for representing knowledge and for problem solving: Kowalski himself suggests a new variant of situation calculus for robot plan formation (Kowalski 1974 , Chapter 3).

It is our belief that the situation calculus, and formal logical tools in general, are and will remain relevant to AI, if for nothing else then for its important theoretical role. This will be partly documented in this and the following section.

To make our arguments clearer we shall consider the situation calculus in the simple form used by Green (1969)(cf. also Nilsson 1971 , Section 7-7). Instead of giving its formal definition we shall only sketch how an image space can be converted into a formal theory in situation calculus. Consider an image space $I = \langle T_I, \Phi_I \rangle$ as introduced in Section 2 and let L be the underlying logical language. We extend L in such a way that each formula A of L is "parametrized" by a special situation argument: a new formula, denoted by $A[\![s]\!]$ expresses the same as A but only for situation s . Furthermore, the operator schemata are used as additional function symbols in the new language and are interpreted as mappings from situations with objects as parameters into situations. Thus the term $\varphi(x_1, \ldots, x_m, s)$ corresponds to the operator schema $\varphi[x_1, \ldots, x_m]$ applied in situation s. This is an example of <u>situation term</u>, while the terms with value of type object are called <u>object terms</u>.

Two alternative axiomatic systems in situation calculus can be associated with the image space I, both with three basic types of axioms. The first system is denoted by K_I and involves:

(i) for each formula A in T_I a <u>core axiom</u> of the form
$$\forall s \; A[\![s]\!] ;$$

(ii) for each operator schema $\varphi[x_1, \ldots, x_m]$ with condition C_φ and result R_φ an <u>operator axiom</u> of the form
$$\forall s (C_\varphi[\![s]\!] \rightarrow R_\varphi[\![\varphi(x_1, \ldots x_m, s)]\!]) ;$$

(iii) for each $\varphi[x_1, \ldots, x_m]$ and each $\langle A, B \rangle \in Inc_\varphi$ an <u>incidental</u> axiom of the form
$$\forall s (C_\varphi[\![s]\!] \; \& \; A[\![s]\!] \rightarrow B[\![\varphi(x_1, \ldots, x_m, s)]\!])$$

(in particular, when A equals B we call it the frame axiom).

The second system, denoted Ins K_I, differs only in one point: axioms of type (ii) and (iii) are replaced by all their ground instances (variable-free object terms of L are substituted for all parameters of the axioms).

To avoid degenerate cases we shall assume all theories used in the sequel to be consistent.

Let $\langle X, Y \rangle$ be a problem in I and let K be either K_I or Ins K_I. We shall say that the problem $\langle X, Y \rangle$ is solvable in K iff

$$K \vdash \quad \exists s \, (X[\![s_o]\!] \rightarrow Y[\![s]\!]) \qquad (1)$$

where s_o is a constant denoting the initial situation.

It appears as a consequence of Herbrand's theorem, that (1) holds iff there is a finite set $\mathcal{T}$ of variable-free situation terms such that

$$K \vdash \quad X[\![s_o]\!] \rightarrow \bigvee_{t \in \mathcal{T}} Y[\![t]\!] \, . \qquad (2)$$

We shall call $\mathcal{T}$ a solution set for $\langle X, Y \rangle$ in K.

In the case when K is Ins K_I the solution set can be interpreted as a plan for robot's behaviour. If it contains only a single term then it corresponds to a straight-line plan. For instance in our Example 1 we obtain the term

$$\underline{\text{push}}\,(d,m,1,\underline{\text{goto}}\,(1,m,\underline{\text{push}}\,(a,k,1,s_o))) \, .$$

If, on the other hand, it contains two or more terms then it corresponds to a branching plan. This is a very important generalization of plans that enables to postpone certain decisions to the execution stage.

A typical reason for branching in the image space occurs, e.g., when in certain image T we have $T \vdash C_{\varphi_1} \vee C_{\varphi_2}$ for two distinct operators φ_1 and φ_2, but neither $T \vdash C_{\varphi_1}$ nor $T \vdash C_{\varphi_2}$ holds. According to the previous definition neither of the operators is applicable in T; nevertheless, the real world is logically complete and thus one of the two alternatives can be in execution time followed. The nature of branching plans and their role in problem solving is discussed in a more detail in (Štěpánková and Havel 1976), cf. also (Kowalski 1974, p. 41.)

An important metatheorem, proved in (Štěpánková and Havel 1976), shows the correspondence between plan formation in image space and theorem proving in situation calculus. It asserts that a problem $\langle X, Y \rangle$ has a branching plan in I if and only if it is solvable in the associated situation calculus Ins K_I. Moreover, the solution set can be effectively extracted from the proof in Ins K_I.

5. STATE-DEPENDENT FUNCTIONS AND THE EXPLICATION ABILITY

The theory Ins K_I was shown to have the same problem-solving power as the original image space I. Are there cases when the stronger theory K_I can be used as a tool for problem-solving? To prepare for demonstrating such a case we shall somewhat modify Example 1 from Section 2.

Example 2. Let the robot's three room world contain an unknown number of boxes of a varying size. Suppose there is an ordering relation for all boxes represented by predicate SMALLER (x,y). The situation calculus K_I for this world contains among its core axioms also the following:

$$\forall s\ \forall y\ ((\text{ROBOT IN}(y,s)\ \&\ \neg\text{EMPTY}(y,s)) \rightarrow \exists x\ (\text{IN}(x,y,s)\ \&\ \forall u\ (\text{IN}(u,y,s) \rightarrow \text{SMALLER}(x,u,s)))) \qquad (3)$$

i.e., each nonempty room visited by the robot contains the smallest box (i.e., the box smaller than all other boxes in the same room).

What makes the theories K_I and Ins K_I different in general? During a theorem-proving procedure, when the existential quantifiers from the prenex form of the axioms are being removed, the language of the theory under consideration is enriched by new functions (Skolem functions). In a solution suggested by Ins K_I these new function symbols play only a passive auxiliary rôle and never appear in a solution set since they do not occur in the place of parameters of operators - cf. (Štěpánková and Havel 1976) for details.

On the other hand, the terms in a solution set in K_I may easily contain object-valued Skolem functions dependent on situation. In our above example it will be the function replacing " $\exists x$" in axiom (3) (let us denote this function by smallest(y, s)).

How such a function should be interpreted?

Consider a robot with a preprogrammed ability to find and identify in any actual state of the environment a specimen of the object, the existence of which is claimed for this state by an axiom (or equivalently, to determine the value of an object function with one situation argument) and remember it for a later use. In particular, the robot can evaluate its Skolem functions during execution of its plans. This is an activity oriented to a better understanding of the environment rather than to its actual change.

If a robot has the described property for all existential quantifiers occuring in the prenex form of the core axioms and of the consequents of the operator and incidental axioms of K_I we say that it is endowed by the explication ability (for K_I). In the rest of the paper we shall consider only robots with this ability.

All possible actions of the robot are desribed by the theory K_I or even better by a theory K_I' obtained from K_I after the existential quantifiers in question are eliminated using the Skolem functions.

In Example 2 the theory K_I' is obtained by replacing axiom (3) by

$$\forall s\, \forall y\, (\mathrm{ROBOT\ IN}(y,s)\ \&\ \neg \mathrm{EMPTY}(y,s) \rightarrow (\mathrm{IN}(\underline{\mathrm{smallest}}(y,s),y,s)\ \&\ \forall u\, (\mathrm{IN}(u,y,s) \rightarrow \mathrm{SMALLER}(\underline{\mathrm{smallest}}(y,s),u,s)))). \quad (4)$$

Consider a problem $\langle X, Y \rangle$ where X is $\mathrm{ROBOT\ IN}(k)\ \&\ \mathrm{EMPTY}(m)$ and Y is

$$\exists x\, (\mathrm{IN}(x,m)\ \&\ \forall y(\mathrm{IN}(y,m) \rightarrow y = x)\ \&\ \forall y\, \mathrm{SMALLER}(x,y)).$$

It is intuitively obvious that a robot with explication ability can solve this problem iff

$$K_I' \vdash X[\![s_o]\!] \rightarrow \exists s\ Y[\![s]\!] \quad (5)$$

But can a concrete plan for robot's behaviour be extracted from the proof in (5)? If there are no existential quantifiers in axioms of K_I' and in X and no universal quantifiers in Y then (5) holds iff there is a solution set $\mathcal{T}$ for $\langle X, Y \rangle$ that consists of variable-free situation terms of K_I' such that

$$K_I' \vdash X[\![s_o]\!] \rightarrow \bigvee_{t \in \mathcal{T}} Y[\![t]\!]$$

(a consequence of Hilbert-Ackermann's theorem).

For instance, a solution set for the problem in our running example may consist of two terms of the form

$$\underline{\text{push}}\,(\underline{\text{smallest}}\,(l,t),l,m,t)$$

where t is either $\underline{\text{push}}\,(\underline{\text{smallest}}\,(k,s_o),k,l,s_o)$ (if k is nonempty) or $\underline{\text{goto}}\,(k,l,s_o)$ (if k is empty). This describes the plan "if k is nonempty push the smallest box from room k into room l (or else, if k is empty, just go to l) and then push the box which happens to be smallest in l from l into m".

However, not every solution set can be immediately interpreted as a well-defined plan. The difficulty is caused by the new object functions depending on situation argument, which may appear in terms of $\mathcal{T}$ in a rather peculiar way. For example the situation term

$$\underline{\text{push}}\,(\underline{\text{smallest}}\,(m,\underline{\text{goto}}\,(l,m,s_o)),m,k,\underline{\text{push}}\,(a,l,m,s_o))$$

does not represent a feasible command because the value of the object argument $\underline{\text{smallest}}\,(m,\underline{\text{goto}}\,(l,m,s_o))$ can be explicated only in situation obtained from s_o by applying goto from l to m -- but the robot never passes through this situation!

We shall call a situation term regular iff it represents a feasible command (a formal definition is in (Štěpánková 1976)). The following theorem can be proved.

Let K_I' describe the capabilities of a robot with the explication ability and let $\langle X, Y\rangle$ be a problem, where X is a formula without existential quantifiers. Then

$$K_I' \vdash X[\![s_o]\!] \rightarrow \exists s\ Y[\![s_o]\!]$$

iff a set $\mathcal{T}$ of regular terms of K_I' can be effectively found, such that $\mathcal{T}$ is a solution set for $\langle X, Y\rangle$ in K_I', i.e.,

$$K_I' \vdash X[\![s_o]\!] \rightarrow \bigvee_{t\in\mathcal{T}} Y[\![t]\!]$$

(the proof is in Štěpánková 1975).

Certain subterms of situation terms should be distinguished. We shall call them $*$-subterms and define them by induction: The only $*$-subterm of s_o is s_o. Let t be a term of the form $\varphi(\ldots,t_1)$; a situation term t' is a $*$-subterm of t iff t' is a $*$-subterm of t_1 or t' = t.

An execution procedure for a solution set $\mathcal{T}$ from the above theorem may have the following form:

Step 1. Set PRESENT := s_o , HINT := $\mathcal{T}$.

Step 2. If Y is satisfied in the actual environment then exit with success, otherwise continue.

Step 3: For any term of the form $f(\ldots,\text{PRESENT})$ which is a subterm of some term in HINT and where f is a Skolem function, detect in the actual environment the value of f with the same arguments and remember it.

Step 4. Find an operator schema $\varphi[x_1,\ldots,x_m]$ and object terms $a_1,\ldots,a_m$ such that

(i) there is a term $t \in$ HINT for which $\varphi(a_1,\ldots,a_m,\text{PRESENT})$ is a $*$-subterm of t, and

(ii) the condition C_φ with parameters replaced by the values $\alpha_1,\alpha_2,\ldots,\alpha_n$ of the terms $a_1,\ldots,a_m$ (computed using the values of Skolem functions determined in previous steps) is met in the actual environment.

If such an operator schema and terms do not exist then exit with failure; otherwise set OPERATION := := the operator obtained from φ by considering $\alpha_1,\alpha_2,\ldots,\alpha_n$ in the place of its parameters. Set NEXT := $\varphi(a_1,\ldots,a_m,\text{PRESENT})$.

Step 5. Execute the OPERATION.

Step 6. Set PRESENT := NEXT , HINT := the set of all $t \in$ HINT having PRESENT as a $*$-subterm, and go to Step 2.

This procedure ends with failure (in Step 4) only if the theory K_I, which yielded the solution set $\mathcal{T}$, was an inadequate representation of the world.

It is important to note that, as far as state-dependent functions are concerned, the situation calculus appears as a more suitable framework than the image space. It seems likely, that such functions may serve as a proper tool for representation of changing environment in the case when the robot is not the sole agent responsible for the changes.

REFERENCES

Fikes, R.E. & N.J. Nilsson (1971) STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189-208.

Green, C. (1969) Application of theorem proving to problem solving. Proc. IJCAI'69, Washington, D.C.

Kowalski, R. (1974) Logic for problem solving. Memo No.75, Department of Computational Logic, University of Edinburgh.

Nilsson, N.J. (1971) Problem-Solving Methods in Artificial Intelligence. New York: McGraw-Hill.

Simon, H.A. (1972) On reasoning about actions, in Representation and Meaning: Experiments with Information Processing Systems (eds H.A. Simon & L. Siklóssy), 414-430. Englwood Cliffs: Prentice-Hall.

Štěpánková, O. (1975) Skolem functions and the planning in the situation calculus. A collection of papers 1975, Inst. of Computation Techniques, Technical University of Prague.

Štěpánková, O. & I.M. Havel (1976) A logical theory of robot problem solving. Artificial Intelligence (to appear).

L. Steels

A FORMALISM FOR CASE SYSTEMS

Many recent proposals for natural language processing systems use in one way or another case systems. See e.g. Simmons (1973), Schank(1975), Rummelhart, Lindsay and Norman(1972), and the overviews in Bruce(1975) and Samlowski (1975). Also in structural linguistics there is a renewed interest in case systems, see e.g. Fillmore(1968).

This active interest in case systems has not lead to an investigation of what type of formal systems they really are. Most people have thought that no new formalism was needed, and they have expressed their results in terms of phrase structure grammars (or similar systems such as recursive transition networks), with an additional component mapping the phrase structures into case structures.

We think however that phrase structure grammars do not represent adequately the information expressed in case systems, and that much can be gained by studying case systems as peculiar entities. In this paper we present some results of such studies. In § 1 we sketch briefly the basic aspects of case systems and give a broader motivation to search for a new type of system. In § 2 we state a model in which the basic aspects of case systems are incorporated. In § 3 we give some more results and further references.

§ 1 INTRODUCTION

There exist several proposals for case systems. These proposals differ mostly in the cases that are adopted and in the 'level of deepness' that is aimed at. What they all appear to have in common are the following ideas:

(i) A case is a (binary) relation between a predicate and one of its arguments. We call the name for the case the case indicator or simply indicator. Examples of commonly found indicators are agent, object, instrument, source, goal, range, etc...

(ii) Case relations affect language in two ways:
(a) They are expressed (or recognized) by means of surface signals such as prepositions, case affixes, word order, intonation, etc... . These signals are called case markers.
(b) The system if further refined by the idea that semantic properties of the unit under consideration act as selection restrictions.

(iii) A case structure or case frame for a particular predicate is a set of case relations that occur with that predicate together with information about how they are expressed (or recognized) in language, i.e. the case markers and selection restrictions involved.
(iv) A case system is a series of case frames for a given language and a definition of the mechanism by which these case frames are set at work.

Depending on the point where this case analysis level appears, on can make a distinction between syntax directed and semantics directed systems. (Also called first and second generation systems, see Winograd (1973)). In a syntax directed system, such as the systems of Woods (1972), Winograd (1972), Petrick (1973), and many others, one typically finds:
(i) a component in which the sentence is parsed into its immediate constituents (phrases), on the basis of a phrase structure grammar (or basic transition network),
(ii) a component in which the obtained structures are mapped into other (but still constituent) structures on the basis of a transformational component (or an augmented transition network containing register setting actions and conditions on the arcs),
(iii) a component in which the syntactic structures are mapped into semantic structures (i.e. case structures). An example of such a component is Woods' interpreter system (see Woods(1972,section 2.3.) where semantic rules are applied. These rules have two parts: a left part with 'templates consisting of a (syntactic) tree fragment plus additional semantic conditions'(ibid,2.18) and a right part with 'forms or schemata' upon which the evaluation takes place. The mapping proceeds by matching a syntactic structure with the left part of a rule, and if succesful the result is the right part.

In a semantics directed system, examples of which are Riesbeck's parser (Riesbeck 1975) and Wilks'analyzer (Wilks 1975), no phrase structure analysis is undertaken. Instead one starts directly on the level of constructing case frames which contain the information for a semantic evaluation.

The main reasons for building semantics directed systems are
1. A simplification of the whole system occurs because one (very complex) level of analysis is simply skipped.
2. Syntax directed systems are only possible for languages where the order of the constituents is very strict. It appears however that English is quite an exception on this, and even in English the order is not 100% strict.
3. The main arguments is however that by constructing semantics directed systems, it is possible to introduce semantic knowledge at an earlier level. (A more detailed argumentation is given in Wilks(1975),Riesbeck(1975) and Schank(1975))

On a closer examination of the currently existing semantics directed systems, it turns out that their constructors have defined a data structure in which case relations can be expressed (e.g. a dependency graph) and a program to map natural language sentences into these structures. In this paper we take another path of investigation, our interest will be in a neutral characterization of the mechanisms for generating or recognizing language expressions on the basis of case frames. For this purpose we will use techniques of formal language theory and design a rewriting system that works on case frames instead of co-occurring constituents.

§ 2 THE PROPOSED MODEL

2.1. Completion Grammars

It is well known that a formal system in general and a rewriting system in particular consists of

(1) a finite alphabet
(2) a series of rules or patterns (the production rules)
(3) a mechanism to put the rules at work (the derivation relation)
(4) a definition of the language that is generated.

Let us systematically deal with these aspects.

Alphabet

As we stated before a case structure contains a predicate together with its arguments: (Pred arg1 ... argn), therefore we will have an alphabet consisting of (i) a set of predicates or procedure names (Vp) and (ii) a set of arguments (Va).

The predicates are the actual words of the language. From the point of view of generation, the arguments are a set of semantic features (which can act as selection restrictions) and case markers. An argument is 'rewritten' until it is a predicate. From the point of view of recognition arguments are the type of the result that appears as output of the predicate.

Note that the 'binding principle', i.e. the way in which predicates of the language expression are interrelated is not a series of nonterminal symbols, but rather the results of the respective interpretations, where the output of one predicate is input for another one, etc... .

The indices of the arguments are the case indicators. We will leave them out if they are obvious.

Productions

From this conception of input-output relationships, it follows that a rule in the grammar will have to specify a relation between a result and the case frame from which this result is

considered to be the output.

We do this by placing on the left of the arrow the result and on the right the frame itself. Note that by 'the result of a predicate' we do not necessarily mean that a new object is introduced, it can be that the result is simply the argument to which certain information is attached. Consider e.g. the sentences

(1) John translates a book.
(2) The translation of a book by John.
(3) A book was translated by John.

In these sentences the (deep) case frame is the following:

(TRANSLATE
(person)$_{agent}$ (text)$_{patient}$ (lang.)$_{source}$ (text)$_{product}$ (lang)$_{goal}$)

Where the product case denotes the product obtained after doing the translating.

In sentence (1) the case frame of TRANSLATE is attached to the agent, in (2) the result of the case frame introduces a new object, namely the product of translating,and in (3) the frame is attached to the patient.

Let us now give the first basic definition:

Definition 1.

A *completion grammar* is a 5-tuple G = ⟨ Va,Vp,P,AX,K ⟩ where

(1) Va,Vp are two finite disjoint sets called the set of arguments and the set of predicates respectively.

(2) V = Va ∪ Vp

(3) P is a finite subset of Va x VpVa* , the set of productions of the form

$a \rightarrow A\ a_1 \ldots a_n$ with $a, a_1, \ldots, a_n \in$ Va and A ∈Vp.

(4) AX ⊂ Va is the axiomset

(5) K is a mapping to be explained in the next paragraph.

The derivation relation

The main task of the derivation relation in this system is (i) to control the rewriting of the arguments, and (ii) the regulation of the order in which the arguments and predicates will appear in the language expression. As regards the order of case frames, it is well known from the work of Greenberg (1966) and many others, that there are certain basic word order patterns such as VSO, VSO, SOV, etc... , where S stands for subject, V for verb and O for object. If we generalize these insights and consider V as the predicate and S and O as the arguments of a case frame, then we obtain basic word order patterns for case frames such as:

(1) prefix: the predicate comes first followed by all its argument

(2) infix: the predicate comes after the first argument and

all other arguments follow the predicate,
(3) postfix: the predicate follows all its arguments,
etc...

The derivation process control whether a case frame is to be realized in a prefix, infix, postfix or other manner. For this purpose we associate with each type of word order a particular derivation relation. In order to specify for a production rule itself in what way it is to be applied in the rewriting process, we define a mapping $K \subseteq P \times I$ where $I = \{\text{prefix, infix, unspecified, etc...}\}$. We denote this by writing an index on the arrow in the production.
E.g. $a \xrightarrow{\text{pref}} A\, a_1 \ldots a_n$
Now we can define the derivation relation.

Definition 2.

(1) The relation $\Rightarrow$, i.e. *direct prefix derivation* is defined as follows:

$$(\forall x,y)_V^{\star}\ (x \Rightarrow y) \text{ iff } (\exists a)_{Va}\ (x = x_1 a x_2,\ a \xrightarrow{\text{pref or unsp}} A\alpha \in P,\ y = x_1 A \alpha x_2)$$

(2) The relation $\Rightarrow\!\!\!\!\Rightarrow$, i.e. *direct infix derivation* is defined as follows:

$$(\forall x,y)_V^{\star}\ (x \Rightarrow\!\!\!\!\Rightarrow y) \text{ iff } (\exists a)_{Va}\ (x = x_1 a x_2,\ a \xrightarrow{\text{inf. or unsp}} A a_1 \alpha \in P,\ y = x_1 a_1 A\, \alpha x_2)$$

If $a = a_1$ then $\Rightarrow\!\!\!\!\Rightarrow$ is called a strict infix derivation.

Other orderings are defined in a similar way, but as an example these two will do.

(3) The relation $\overset{\star}{\Rightarrow}$, called *prefix derivation*, and $\overset{\star}{\Rightarrow\!\!\!\!\Rightarrow}$ called *infix derivation*, denote the reflexive and transitive closure of $\Rightarrow$ and $\Rightarrow\!\!\!\!\Rightarrow$ respectively.
Let $\Rightarrow = \Rightarrow \cup \Rightarrow\!\!\!\!\Rightarrow$ and $\overset{\star}{\Rightarrow}$ be the reflexive and transitive closure of $\Rightarrow$.

The language definition

Definition 3.

Let $G = \langle Va, Vp, P, AX, K \rangle$ be a completion grammar
(1) The language of G denoted as L(G) is defined by
$L(G) = \{x \mid (\exists a)_{AX},\ a \overset{\star}{\Rightarrow} x \text{ with } x \in Vp^{\star}\}$
(2) We say that G is a prefix completion grammar iff
$(\forall p)_P\ (K(p) = \text{prefix})$

(3) We say that G is an infix completion grammar iff
$(\forall p)_P\ (K(p) = \text{infix})$

(4) Let CG denote the class of all completion grammars.

Before we discuss the natural language application, we introduce a simple grammar for the propositional calculus.

Example 1.

Let G = ⟨ Va, Vp, P, AX, K ⟩ ∈ CG with V = {⟨ log ⟩},
Vp = {AND, OR, IMPLIES, NOT, p,q,r,...} , AX = {⟨ log ⟩}
and P:

1. $\langle \text{log} \rangle \xrightarrow{\text{pref}} \left\{ \begin{matrix} \text{OR} \\ \text{AND} \\ \text{IMPLIES} \end{matrix} \right\} \langle \text{log} \rangle \langle \text{log} \rangle$

2. $\langle \text{log} \rangle \xrightarrow{\text{pref}} \text{NOT} \langle \text{log} \rangle$

3. $\langle \text{log} \rangle \xrightarrow{\text{pref}} \left\{ \begin{matrix} \text{p} \\ \text{q} \\ \text{r} \end{matrix} \right\}$

(Note that p,q,r,... are procedures with no input arguments, they compute a logical value, e.g. by retrieving it from the memory)

Some derivations:

(i) $\langle \text{log} \rangle \Rightarrow \text{AND} \langle \text{log} \rangle \langle \text{log} \rangle \Rightarrow \text{AND} \langle \text{log} \rangle \text{NOT} \langle \text{log} \rangle \Rightarrow$
$\text{AND p NOT} \langle \text{log} \rangle \Rightarrow \text{AND p NOT q}$

(ii) $\langle \text{log} \rangle \Rightarrow \text{p}$

Clearly what we obtain are expressions in the propositional calculus in Polish notation.

Example 2.

Let G = ⟨ Va, Vp, P, AX, K ⟩ with Va, Vp, AX as in example 1.
P:

1. $\langle \text{log} \rangle \xrightarrow{\text{inf}} \left\{ \begin{matrix} \text{AND} \\ \text{OR} \\ \text{IMPLIES} \end{matrix} \right\} \langle \text{log} \rangle \langle \text{log} \rangle$

2. $\langle \text{log} \rangle \xrightarrow{\text{pref}} \text{NOT} \langle \text{log} \rangle$

3. $\langle \text{log} \rangle \xrightarrow{\text{pref}} \left\{ \begin{matrix} \text{p} \\ \text{q} \\ \text{r} \end{matrix} \right\}$

Some derivations:

(i) $\langle \text{log} \rangle \Rightarrow \langle \text{log} \rangle \text{ AND } \langle \text{log} \rangle \Rightarrow \langle \text{log} \rangle \text{ AND NOT} \langle \text{log} \rangle \Rightarrow$
$\text{p AND NOT} \langle \text{log} \rangle \Rightarrow \text{p AND NOT q}$

(ii) $\langle \text{log} \rangle \Rightarrow \text{p}$

Clearly what we obtain are expressions in the propositional calculus but now in infix notation.

These examples show already that (i) the 'communication' between procedures goes through their output arguments, (ii) the same form of a rule gives rise to different realizations depending on the derivation mode.

Now we turn to natural language again. First a remark should be made on the way in which the case markers are realized. If the case marker is a case affix, then it is realized by the morphological component, if this marker is a preposition the argument is again to be rewritten, simply because a preposition is also a predicate (and it can serve other purposes than being a case marker).

Let us now give a very small natural language grammar. It serves no other purpose than being an illustration of our ideas.

Example 3.

Let $G = \langle Va, Vp, P, AX, K \rangle \in CG$ with $Va = \{\langle text \rangle, \langle person \rangle, \langle language \rangle, \langle of,text \rangle, \langle from,text \rangle, \langle by,person \rangle, \langle from,language \rangle\}$, VP = {TRANSLATION, TRANSLATES, TRANSLATOR, ARTICLE, DUTCH, READS, AUTHOR, JOHN, OF, FROM, BY}
$AX = \{\langle person \rangle\}$
P:

(1) $\langle text \rangle \xrightarrow{pref}$ TRANSLATION $\langle of,text \rangle \langle from,text \rangle \langle by,person \rangle$

(2) $\langle person \rangle \xrightarrow{inf}$ TRANSLATES $\langle person \rangle \langle text \rangle \langle from,language \rangle$

(3) $\langle person \rangle \xrightarrow{pref}$ TRANSLATOR $\langle of,text \rangle \langle from,language \rangle$

(4) $\langle text \rangle \xrightarrow{pref}$ ARTICLE

(5) $\langle language \rangle \xrightarrow{pref}$ DUTCH

(6) $\langle person \rangle \xrightarrow{infix}$ READS $\langle person \rangle \langle text \rangle$

(7) $\langle person \rangle \xrightarrow{pref} \begin{Bmatrix} AUTHOR \\ JOHN \end{Bmatrix}$

(8) $\langle of,text \rangle \xrightarrow{pref}$ OF $\langle text \rangle$

(9) $\langle from,text \rangle \xrightarrow{pref}$ FROM $\langle text \rangle$

(10) $\langle by,person \rangle \xrightarrow{pref}$ BY $\langle person$

(11) $\langle from,language \rangle \xrightarrow{pref}$ FROM $\langle language \rangle$

(Note that we left out case indicators and make abstraction of determiners)

Some derivations:

(i) $\langle person \rangle \Longrightarrow \langle person \rangle$ TRANSLATES $\langle text \rangle \langle from,language \rangle$
$\Longrightarrow$ AUTHOR TRANSLATES $\langle text \rangle \langle from,language \rangle$
$\Longrightarrow$ AUTHOR TRANSLATES ARTICLE $\langle from,language \rangle$
$\Longrightarrow$ AUTHOR TRANSLATES ARTICLE FROM $\langle language \rangle$
$\Longrightarrow$ AUTHOR TRANSLATES ARTICLE FROM DUTCH

(i.e.'(the) author translates (an) article from Dutch')

(ii) ⟨ person ⟩⟹⟨ person ⟩ READS⟨text ⟩⟹ JOHN READS⟨text⟩
⟹ JOHN READS TRANSLATION ⟨ of,text ⟩⟨from language ⟩⟨by,person ⟩
⟹JOHN READS TRANSLATION OF⟨ text ⟩⟨from,language⟩⟨ by, person ⟩
⟹ JOHN READS TRANSLATION OF ARTICLE⟨from,language⟩⟩⟨by,person⟩
⟹ JOHN READS TRANSLATION OF ARTICLE FROM ⟨ language⟩⟨by,person⟩⟩
⟹JOHN READS TRANSLATION OF ARTICLE FROM DUTCH ⟨ by, person ⟩
⟹JOHN READS TRANSLATION OF ARTICLE FROM DUTCH BY ⟨ person ⟩
⟹ JOHN READS TRANSLATION OF ARTICLE FROM DUTCH BY AUTHOR

(i.e. 'John reads (the) translation of (an) article from (the) Dutch by (the) author')

Comment:

1. It should be clear that a whole language expression is constructed by the interplay of different 'case frames' which all have their own specific word order.
2. Also, each case frame 'hangs' on one particular predicate and as a consequence the parsing process is completely guided by the predicate. Once we know the predicate we know the rule in the grammar to be applied (this omits costly search through the grammar !), also we know what other elements, that will fill up the rest of the frame, are to be expected. Note that we do not know their 'phrase structure' appearance, but only what semantic properties those arguments will have, including the case markers.
3. A predicate in a case frame is not only realized as a verb (as is mostly assumed), but ALL words in the sentence are considered as predicates.
4. It is possible to associate with each derivation a structure in the same way as constituent structures are associated with phrase structure derivations. The important thing is that these structures can be direct input for the semantic evaluation, because all information for this process is provided by the case structures specified in the grammar.
5. The term completion grammar denotes the activity of completing patterns. Due to the fact that this process is partially guided by the semantic properties of the arguments, a freeer attitude towards word order is possible.

We now deal with a second part of the theory of completion grammars.

2.2. META-GRAMMARS

The careful reader must have noted that the same deep case frame (such as the one for TRANSLATE) is realized in different surface case frames depending on the argument to which the case frame is attached.

It seems therefore a natural extension to have instead of an explicit set of surface structure rules a mechanism to construct such rules. Input for this mechanism are the abstract (or deep) case frames and an indicator to which argument the surface case frame is to be attached. The output of this mechanism is a rule of the type we have been discussing, that means the rule specifies (i) the morphology of the predicate, (ii) what arguments are allowed and by what case markers each of these arguments is to be realized, (iii) the basic word order (prefix, infix, etc...) for the surface case frame. This rule is then applied to yield the desired natural language expression.

Example 4.

(TRANSLATE (person)$_{agent}$ (text)$_{patient}$ (text)$_{result}$

(language)$_{source}$ (language)$_{goal}$)

If the pattern is attached to the agent we can have:

⟨ person ⟩ $\xrightarrow{infix}$ TRANSLATES ⟨ person ⟩⟨ text⟩⟨ from,language ⟩

Note that the result can not be expressed in this type of realization, the order is infix, the morphological ending of the predicate is that of a main verb and the case marker telling that a preposition is present, is added to the source case.

If the pattern is attached to the result:

⟨ text ⟩ $\xrightarrow{prefix}$ TRANSLATION ⟨ of,text Xby,person Xfrom,lang.⟩

Note that the same information is expressed, only the 'point of view' changed. Note the different word order, the nominal form of the predicate, the prepositions indicating the case relations.
The reader should construct some other surface case frames that can be derived from the above deep case frame.

Comment

1. It should be clear that such a meta-grammar serves the same purpose as a transformation component or an augmentation of a transition network. The way in which the problem is solved however, is completely different. In particular instead of changing already obtained structures, we have a productive component which acts even before the structures are constructed. This method is made possible by starting with deep case frames as underlying basis instead of co-occurring units.

2. During parsing the morphological endings of the predicate indicate to the parser what 'meta-rules' have been applied. This omits again in a natural way the costly trying out of all sorts of transformations.
3. Less storage is required because we do not need to store a surface grammar. Also by generalizing the strategies for surface patterns production, a greater 'habitability' of the system is obtained.

How 'deep' this meta-component can work is shown by our final example: Consider the deep case frame:

(WRITE (person)$_{agent}$ (text)$_{product}$ ...)

If the case frame has as result the agent, we have:

$$\langle \text{person} \rangle \xrightarrow{\text{prefix}} \text{AUTHOR} \langle \text{of,text} \rangle$$

$$\langle \text{of,text} \rangle \xrightarrow{\text{Prefix}} \text{OF} \langle \text{text} \rangle$$

If the case frame has as result the product:

$$\langle \text{text} \rangle \xrightarrow{\text{prefix}} \text{TEXT} \langle \text{by,person} \rangle$$

$$\langle \text{by,person} \rangle \xrightarrow{\text{prefix}} \text{BY} \langle \text{person} \rangle$$

If the case frame is attached to the agent:

$$\langle \text{person} \rangle \xrightarrow{\text{infix}} \text{WRITES} \langle \text{person} \rangle \langle \text{text} \rangle$$

to the product

$$\langle \text{text} \rangle \xrightarrow{\text{infix}} \text{WRITTEN} \langle \text{text} \rangle \langle \text{by,person} \rangle$$

etc...

§ 3 Further references

Although it would be instructive to give many more examples, due to space limitations we have to close our short discussion and we do this by reviewing current research on completion grammars. This involves two approaches: (i) theoretical, a study of the systems from a formal point of view, (ii) practical: the application of the theory in computational systems.

Theoretical

Completion grammars are at present formally worked out and investigated. (Results appeared in Steels & Vermeir (1976a), Steels(1976)). It is known that they have a weak generative capacity of type 2.

In addition a related class of completion automata was developed. These automata have a finite control and additional stacks to store states and arguments. Various results could be obtained by studying the degree of complexity in terms of operations of the automata and required memory, of each

basic word order pattern. (See Steels and Vermeir (1976b)). Also there have been some results on the generalization of the systems to make them free from word order. The meta-grammars are as yet not completely worked out formally.

Practical

Completion grammars developed from attempts to design computational models of language analysis and synthesis. In this respect they formed the underlying theory for small experiments in language processing (see Steels (1975)). Also in the context of research in automatic translation, a language producing system was designed and implemented, which contains a meta-component to produce surface structure rules from abstract deep case frames. This production system takes as input expressions in an 'interlingua' and,after consultation of the meta-grammar,produces language expressions in the language specified by the meta-grammar.

Conclusions

We hope to have made clear that it is possible to develop other means of analyzing and producing natural language than by the common phrase structure grammars approach. In particular we have presented a model that incorporates the basic aspects of case systems.

In linguistic theory a case is considered to be a relation between a predicate and one of its arguments, in a case structure we define the particular case relations that are found with a given predicate together with information about how these particular case relations are expressed (or recognized) in natural language. This includes surface signals, such as prepositions, case affixes, word order, etc... , the so called case markers, and semantic properties of the argument which act as selection restrictions.

In the paper we introduced a formal model, called a completion grammar in which the sort of information normally found in case systems can be expressed. This model consists of a rewriting system where the alphabet contains predicates and arguments and the productions define case structures. Order is being regulated by the generation and recognition mechanisms.

As a further extension of the theory a meta-grammar was proposed that constructs the rules for realizing surface case structures on the basis of deep case structures and information by what way the case frame should be introduced.

Although there remains a lot of work to be done, we are convinced by the results so far obtained, that the model discussed in this paper will once form an efficient tool in

It seems therefore a natural extension to have instead of an explicit set of surface structure rules a mechanism to construct such rules. Input for this mechanism are the abstract (or deep) case frames and an indicator to which argument the surface case frame is to be attached. The output of this mechanism is a rule of the type we have been discussing, that means the rule specifies (i) the morphology of the predicate, (ii) what arguments are allowed and by what case markers each of these arguments is to be realized, (iii) the basic word order (prefix, infix, etc...) for the surface case frame. This rule is then applied to yield the desired natural language expression.

Example 4.

(TRANSLATE (person)$_{agent}$ (text)$_{patient}$ (text)$_{result}$

(language)$_{source}$ (language)$_{goal}$)

If the pattern is attached to the agent we can have:

⟨ person ⟩ —infix→ TRANSLATES ⟨ person ⟩⟨ text⟩⟨ from,language ⟩

Note that the result can not be expressed in this type of realization, the order is infix, the morphological ending of the predicate is that of a main verb and the case marker telling that a preposition is present, is added to the source case.

If the pattern is attached to the result:

⟨ text ⟩ —prefix→ TRANSLATION ⟨ of,text Xby,person Xfrom,lang.⟩

Note that the same information is expressed, only the 'point of view' changed. Note the different word order, the nominal form of the predicate, the prepositions indicating the case relations.
The reader should construct some other surface case frames that can be derived from the above deep case frame.

Comment

1. It should be clear that such a meta-grammar serves the same purpose as a transformation component or an augmentation of a transition network. The way in which the problem is solved however, is completely different. In particular instead of changing already obtained structures, we have a productive component which acts even before the structures are constructed. This method is made possible by starting with deep case frames as underlying basis instead of co-occurring units.

C.F.Schmidt
N.S.Sridharan
J.L.Goodson

RECOGNIZING PLANS AND SUMMARIZING ACTIONS

ABSTRACT

The structural aspects of an information processing theory of how persons understand the actions of others is presented. The process described is a constructive process which uses a plan schema to recognize the plans which account for the observed actions. The logical and psychological rules of plan consistency are presented and their use in plan recognition is described. One measure of the understanding of observed actions is the ability to summarize the observations. Properties of summaries are discussed and it is shown that these properties result from the plan recognition process.

Key Phrases. plan, summary, belief systems, theory of actions.

If persons are asked to describe or explain the observed actions of others, such explanations invariably make reference to the beliefs, intents, and goals of the actor. The BELIEVER system discussed in this paper is an AI system that is being developed to give a psychological account of how persons use their general beliefs about other persons to arrive at an intentional explanation of the actions of others. The BELIEVER system is an explicit attempt to use the tools of AI to construct a formal psychological theory of the processes involved in understanding the actions of others. The theory is not completed. However, in this paper we will present the major structural components that are used to understand observed actions. Then we will show the way in which these assumptions lead to testable predictions about the form of persons' summaries of their observations of an action

basic word order pattern. (See Steels and Vermeir (1976b)). Also there have been some results on the generalization of the systems to make them free from word order. The meta-grammars are as yet not completely worked out formally.

Practical

Completion grammars developed from attempts to design computational models of language analysis and synthesis. In this respect they formed the underlying theory for small experiments in language processing (see Steels (1975)). Also in the context of research in automatic translation, a language producing system was designed and implemented, which contains a meta-component to produce surface structure rules from abstract deep case frames. This production system takes as input expressions in an 'interlingua' and, after consultation of the meta-grammar, produces language expressions in the language specified by the meta-grammar.

Conclusions

We hope to have made clear that it is possible to develop other means of analyzing and producing natural language than by the common phrase structure grammars approach. In particular we have presented a model that incorporates the basic aspects of case systems.

In linguistic theory a case is considered to be a relation between a predicate and one of its arguments, in a case structure we define the particular case relations that are found with a given predicate together with information about how these particular case relations are expressed (or recognized) in natural language. This includes surface signals, such as prepositions, case affixes, word order, etc... , the so called case markers, and semantic properties of the argument which act as selection restrictions.

In the paper we introduced a formal model, called a completion grammar in which the sort of information normally found in case systems can be expressed. This model consists of a rewriting system where the alphabet contains predicates and arguments and the productions define case structures. Order is being regulated by the generation and recognition mechanisms.

As a further extension of the theory a meta-grammar was proposed that constructs the rules for realizing surface case structures on the basis of deep case structures and information by what way the case frame should be introduced.

Although there remains a lot of work to be done, we are convinced by the results so far obtained, that the model discussed in this paper will once form an efficient tool in

Table 2. Coherent and Non-Coherent Summaries

Coherent Summaries

A) Tom got out a cone and went to the refrigerator to get some ice cream but there wasn't any ice cream so he wasn't able to make an ice cream cone to eat.

B) Tom wanted to make an ice cream cone to eat. He thought that there was ice cream in the freezer, but when he looked he found that there wasn't any ice cream.

C) Tom wanted to make an ice cream cone to eat, but there wasn't any ice cream.

Non-Coherent Summarries

D) Tom took a cone out of the cabinet. Then he went to the refrigerator and opened the freezer. Then he closed the freezer and put the cone away.

E) Tom walked to the cabinet and then he walked to the refrigerator and then he walked back to the cabinet.

F) Tom opened the cabinet and opened a box of cones, then he opened the freezer, but there was no ice cream so he put the cone away.

G) Tom opened the cabinet and took out a cone and got a cone. Then he opened the refrigerator and then closed it.

H) Tom walked to the cabinet, opened it and put the cone away.

I) Tom got a cone and closed the cabinet door. Then he went to the refrigerator to get some ice cream and closed the door. Since there wasn't any ice cream he put the cone away.

Within the work on the BELIEVER system we have constructed a theory which assumes that these

properties of a summary of intentional action are an inherent property of the understanding process. The process of understanding is assumed to be an active process of recognizing the intentions and goals of the actor. This process is essentially a process of recognizing the plans that generated the observed actions. Therefore, before returning to an explicit consideration of the problem of summarization we will first develop some of our ideas on Plan Recognition.

PROPERTIES OF ACT SEQUENCES AND PLANS

The problem of Plan Recognition is to take a sequence of observed actions and to hypothesize plans in order to explain the observed actions. The hypothesized plans have to be consistent with respect to a specification of plan consistency, and be correct in the degree to which they match the observed actions. This problem of Plan Recognition can be elucidated by contrasting some of the properties of plans with some of the properties of the observed actions (execution traces of plans) from which the plans must be inferred.

Some of the properties of the observed actions are:

(1) Actions are events which occur in and affect an actual world (The Physical or External property of actions);

(2) Sequences of observed actions are non-hierarchical and linearly ordered in time (Temporal Linear Ordering property of actions);

(3) An action immediately before or after a particular action sequence can be added to that action sequence and this extension yields an action sequence (Non-boundedness property of actions).

Some of the properties of recognized plans are:

(1) A plan is a partial model of a possible course of action executed in a possible world (The Psychological or Internal property of plans);

(2) A plan is a hierarchical structure of units each of which is also a plan (Composition property of plans);

(3) This hierarchical structure of a plan is logically ordered. This logical ordering creates a partial ordering of the subplans of which a particular plan is composed. (Logical Ordering property of plans);

(4) A plan has a logical beginning and end

(Boundedness property of plans).

This brief sketch and comparison of plan and observed action properties serves to point out the degree to which a process that attempts to perform a mapping from observations into plans involves a change in representation. The problem of plan recognition is to accomplish this shift in representation.

The necessity for such a shift can be better appreciated by comparing the action sequence of table 1 with the coherent summarizations of table 2. These coherent summaries refer to Tom's goal of making and eating an ice cream cone. However, no action of making or eating was actually observed since Tom's plan failed because there was no ice cream available. The coherency of Tom's actions derives from the plan that we as observers attibute to him. The actions themselves are in no way defective or unusual. Nonetheless, this sequence has no meaning unless the assumption is made that Tom wanted an ice cream cone and he believed that he would be able to make one when he initiated the observed course of action.

In the BELIEVER system the accomplishment of this representational shift is guided through the use of a plan schema. The plan schema is a structural representation of the properties of the plan units out of which a particular plan structure is constructed. The plan schema also includes a logical representation of each of the relations that a plan enters into which serves to define the rules of plan consistency. Hypotheses about the plans which underlie the observed actions are asserted as instantiated plan schema and these hypotheses may be merged to form composed plan structures by specifying the rules of composition on these plan units. Thus, plans are composed but actions are not. In this framework the reasoning involves a representational shift that is accomplished through the use of a schema which is part of the internal domain knowledge only. This exemplifies what psychologists might refer to as constructive reasoning and AI workers would call reasoning in problems of formation.

THE PLAN SCHEMA AND PLAN RECOGNITION

The first property mentioned above is the psychological or internal nature of the propositions

that constitute a plan. This property of plans is particularly important in plan recognition since these plans are inferred structures which are attributed to the agents of the observed actions. Consequently, a hypothesized plan must not only be consistent with a model of the physical world, but also consistent with the observer's psychological model of a person.

Since plan recognition is a predictive process (cf. Schmidt, 1976), the status of the propositions that constitute a plan will differ depending upon whether the plan is an explanation of observations already made by the system or a prediction about future observations. If the goal of a plan has been supported by observations already made by the system, then the configuration of propositions that constitute a plan are presumed true. Any contradiction, psychological or physical, would serve to invalidate the plan hypothesis. The propositions which are related to a predicted but as yet unobserved plan must be plausible, that is, not contradicted by what is known. However, once the goal of a predicted plan is supported by observations then these propositions are also presumed true.

A graphical representation of the concept of a plan schema is displayed in figure 1. The plan schema consists of the major relations that are defined between the box labelled Plan (*) and other nodes of this graph. The representation of the psychological sense of a Plan can be described by first considering those relations that hold between the Plan box and the various circles which represent sets of propositions.

Consider first the Opportunities relation. For non-social actions such as 'walk' and 'make,' these propositions are derived from the physical preconditions for the act. For social actions such as 'buy,' 'give,' 'ask' and 'help,' the preconditions for the act may also refer to mental states of the participants. The psychological property of this Opportunities relation is represented by associating with the relation the constraint that:

(*) The convention will be followed that when words appearing in the text refer to technical terms, the first letter of the word will be capitalized.

```
((PLAN L) opportunities (PROPOSITION S)) =>
        ((L planner:believes S) before L)
```

The logical formalism used in this paper is a variant of that designed for use in the MDS system (Srinivasan 1973). The statement above provides the plan recognition process with the information that the assertion of an Opportunities relation requires that the Planner of L Believes all of the propositions, S. The confirmation of this belief constraint only needs to be carried through to insure freedom from contradiction. Absolute confirmation is not required. This is facilitated by the three-valued (True, Unknown, False) logic available in the MDS system.

A use of this belief constraint can be clarified by considering an opportunity for 'taking' ice cream from the refrigerator. If one such proposition is that the ice cream is in the refrigerator, then the system has to verify that the Planner Believes that the ice cream is in the refrigerator. If this 'take' were an observed action specified as input to the system, then the satisfaction of this constraint can be presupposed. On the other hand, if the system attempts to predict that the action of 'taking' is part of the actor's plan as would be necessary to account for the action sequence of table 1, then the actual satisfaction of the precondition will inform the observing system of the possible success or failure of the executed plan.

The psychological constraint on the propositions in the Goal and Outcome relations to the Plan are introduced in a similar manner. These constraints are:

```
((PLAN L) goal (PROPOSITION S)) =>
        ((L outcome S)
        ((L planner:wants S) before L)

((PLAN L) outcome (PROPOSITION S)) =>
        ((L planner:expects S) before L)
        ((L planner:knows S) after L)
```

The Goal proposition is also an Outcome proposition. However, the Goal outcome is the only Outcome which the actor can be said to want. A person who plans to buy a radio is said to want to own the radio, but it would not be appropriate to say that the person wants to get rid of the money needed to pay for the

radio. If this were the goal, then the plan would not be a 'buy.' This distinction between the Goal of a Plan and its Outcomes will play an important role when the composition of Plan units is considered.

The propositions involved in the Reason-For relation play a distinctive role in the plan recognition process. Recall that the observer does not know the actor's goal. Consequently, the hypothesis that the actor is pursuing a particular Plan and thus a particular Goal must be supported. This support arises from the observer's psychological theory about the kind of internal psychological states that motivate persons to choose to execute particular plans. The propositions in the Reason-For relation are the propositions about the internal states of the actor which support the assumption that the actor Wants the Goal of this particular plan. These propositions about the Planner's internal states arise through the application of psychological rules of interpretation which specify the feelings, sentiments, needs and so on which the observer believes motivate the pursuit of a particular goal. This constraint is represented as:

```
((PLAN L) reason-for (PROPOSITION S)) =>
        (L planner:motivated-by S)
```

With these classes of propositions defined, it is now possible to consider the various ways in which Plan units may be related and how these relations are used to structure the formation of an overall explanation of the person's actions. Four types of relations between Plan units will be defined. These four relations are referred to as Enable, Motivate, In-Order-To, and Means-Of. The use of these relations to construct an explanation of an observed action sequence is exemplified in figure 2. This figure provides a highly schematized graphical representation of an explanation of the action sequence of table 1. In this figure the small rectangular boxes which are numbered 1 through 14 contain a single verb which corresponds to the actions mentioned in the description in table 1. The numbering as well as the horizontal position of these boxes in the figure is used to represent the temporal ordering of the actions. Each of the boxes of figure 2 represent a Plan unit. The boxes which contain two or more other boxes are referred to as a

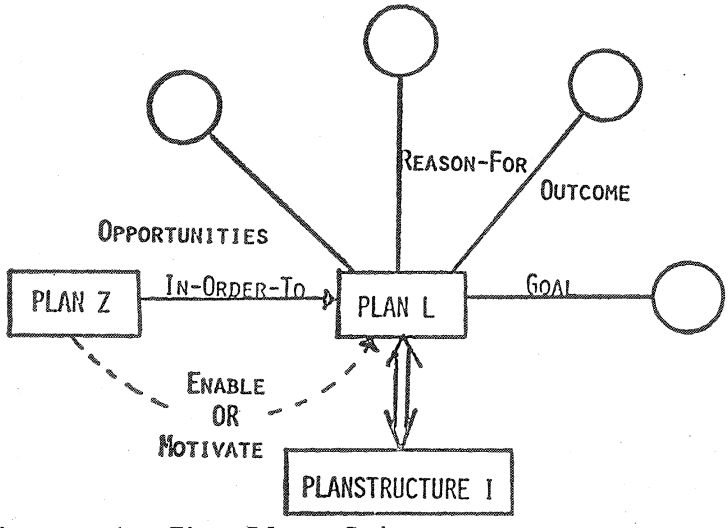


Figure 1. The Plan Schema.

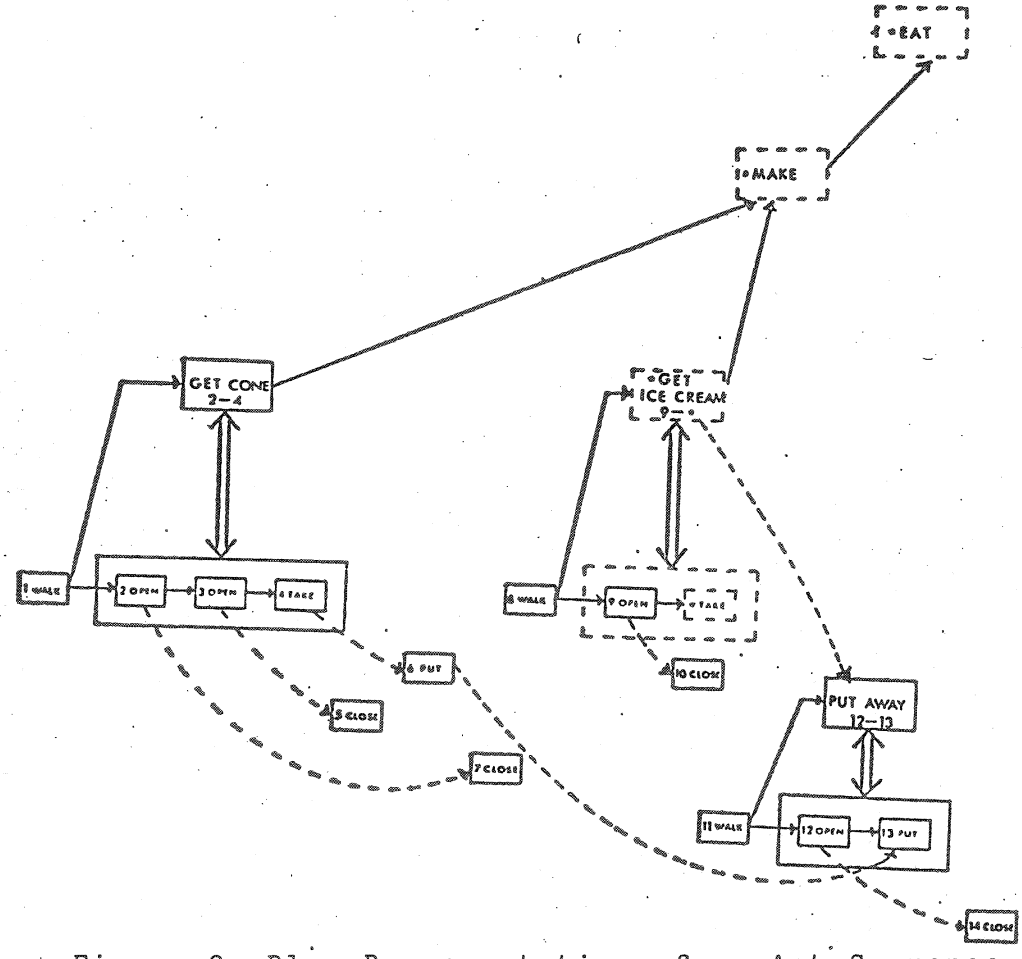


Figure 2. Plan Representation of an Act Sequence.

Planstructure. Some of the boxes of figure 2 are shown with dotted lines. This has been done to indicate that although these are part of the plan hypothesis, the actions that were predicted by these structures were not observed in the action sequence.

Consider first a Plan Z which precedes but does not overlap in time with another Plan L, and an Outcome S of the Plan Z. If this Outcome S is a precondition for the execution of the Plan L then Z is said to Enable L and S is one of the Enabling States of the Enable relation. A Motivate relation can be defined in an analogous manner. In this case the Outcome proposition S of a Plan Z is a member of the set of propositions that constitute the Reason-For the subsequent Plan L. Both of these relations capture a logical dependency between two Plan units. The dotted lines of figure 2 represent some of these Enable and Motivate relations. For example, the act of opening the cabinet (2) Enabled the closing of the cabinet (5). The dotted line between (2) and (7) represents both a Motivate and an Enable relation between these two actions. The Motivate relation holds since the outcome of opening the cabinet (2) "activated" a tidiness normative rule which states that one ought to reverse the side effects of one's actions when these side effects disrupt the prescribed states of affairs in a particular setting.

Although many of the Enable and Motivate relations are represented in figure 2, for reasons of clarity a good many of these relations have not been shown. If these ubiquitous relations were sufficient grounds for concluding that two Plan units were actually elements of the same overall plan, then the hypothesis space would become enormous. It is for this reason that the distinction made earlier between a Goal and an Outcome is so important. Clearly, Tom's goal in opening the cabinet was not to have the door open so that he could then close it. Rather his goal was to get a cone out of the cabinet.

This means that Plan units can not be composed into a larger plan based on the Enable or Motivate relation alone. A new relation is required that is defined over the Enable or Motivate relations and introduces the appropriate psychological constraints. This In-Order-To relation is rather torturously expressed as:

```
((PLAN Z) in-order-to (PLAN L)) =>
  [(PROPOSITION S)(Z goal S) =>
   [(Z enable L)(S enablestateof (Z enable L))
    v (Z motivate L)(S motivatestateof
                     (Z motivate L))]]
  [((Z planner) wants (L goal)) before Z)]
  [(PROPOSITION B)((B opportunity L)
   v (B reason-for L)) ~(Z outcome B) =>
        ((Z planner) believes B)]
```

The In-Order-To relation represents one relation whereby a particular Plan may be composed out of Plan units. To say that one Plan was done in order to be able to do another constitutes a claim that the two Plan units are really part of the same Planstructure. This relation defines a partial ordering capturing the non-linear nature of plans and allows different execution sequences of a single plan to be recognized.

The In-Order-To relation is represented in figure 2 by a solid arrow. This relation is shown as a pair-wise Plan unit relation. However, the definition of the psychological constraints on this relation creates the effect of giving the transitive closure of the relation. Consequently, actions (1) through (4) of figure 2 really represent the hypothesis that this sequence of actions constituted a single unit of activity for Tom.

The final relation is the Means-Of relation. This is a relation between a plan and a Planstructure. A Planstructure is simply an In-Order-To connected sequence of Plan units. The significance of the Means-Of relation is that it gives the plan hypothesis representation a hierarchical structure that reflects the goal-subgoal structure of the total plan hypothesis. This relation holds if the final goal of the Planstructure can be viewed as the goal of the higher-level Plan unit. The constraint on this Means-Of relation is expressed as:

```
((PLANSTRUCTURE I) meansof (PLAN L)) =>
  (P believes ((I goal) translatable (L goal))
```

The idea of Translatable is that if certain presuppositions of the Goal of L are supported by the context and satisfied by the Goal of I, then the Planstructure I may be viewed as an instance of the Plan L. At its simplest, the Goal of I and L may be

identical. However, the conditions that must hold between the Goals can be quite complex as is exemplified in speech acts such as 'request,' 'promise,' and 'demand' (Cf. Bruce and Schmidt, 1974; Bruce, 1975).

The Means-Of relation is represented in figure 2 as a double arrow. The first Means-Of relation shown in figure 2 was formed because the goal of take (4) was to have the cone. This matched the goal of the subplan Get Cone which resulted from the elaboration of the hypothesis that Tom was planning to make an ice cream cone. The Means-Of relation acts as a kind of equivalence relation that is conditional upon the point of view that results from the observer's hypothesis about the final goal of the Planner's overall plan. The first act of walking to the cabinet is not subsumed under this Means-Of relation since an In-Order-To relation still holds between walk and Get Cone. No such relation holds between Plan units (2) through (4) and Get Cone and therefore these Plan units are subsumed under the Get Cone Plan unit.

Again, this relation and its psychological restriction serve to guide the process of composing plan hypotheses. The Means-Of relation summarizes the individual Plan units that constitute the Planstructure. In doing so, it also dismisses these Plan units from participating in further compositions of Plans. Furthermore, in characterizing the type of Plan that L is, potentially new propositions in the Opportunities, Goal and Outcome relations are introduced. This is especially true for social actions. For example, if Mary hands $10 to Sue and this handing to is seen as a loan, then the propositions constituting the plan 'loan' represent a very different set of propositions than those that define the plan 'hand to.' The introduction of these new propositions makes possible the creation of In-Order-To relations between the higher level Plan L and other subsequent Plans which would otherwise be impossible.

The boundedness property of plans arises from the fact that a Plan cannot further be composed if the final Goal of the Plan has been identified. In order for a Plan unit to be identified as the terminating unit of a Planstructure, the Reason-For relation on the plan unit must be filled by virtue of the observer having a psychological rule for motive attribution that implies the actor's choosing

this as a final Goal (Cf. Schmidt, 1976; Schmidt and D'Addamio, 1973).

The identification of the starting point of a Plan hypothesis is aided by the constraint that the actor must foresee the preconditions of the Plan hypothesis. This foreseeability condition results from the various Believes relations that were defined on the In-Order-To relation as well as the Means-Of relation. These constraints have the effect of shifting the belief constraints back in time to the beginning of the hypothesized Plan. Another aid to identifying the start of a Plan is provided by the hierarchical structure that the Means-Of relation imposes on the Plan hypotheses. This hierarchical structure of subplans that are linked to the final Plan unit provide a partition of the observed actions. This partitioning aids the identification of the start of the Plan by limiting the number of possible starting points.

This set of relations provides the basis for recognizing the different plans that are contained in figure 2. The main plan is, of course, the attempt to make and eat an ice cream cone. However, the execution of the subplan to get a cone led to side effects which generated three additional plans independent of this main plan. These side effect generated plans were (5), (6), and (7) of figure 2. The fact that they are independent is attested to by the fact that the planner could have chosen not to perform them and still have been able to complete the subplan. Similarly plan (10) was also undertaken to reverse a side effect of pursuing the main plan. Finally, the plan to put the cone away and the plan linked to it as a side effect plan were motivated by the failure of the main plan. Thus, not only does the plan representation give information about the various plans that were involved in the observed actions, but it also gives information about the relations between the various independent plans.

THE SUMMARIZING OF ACTIONS

The structure shown in figure 2 arose out of the recognition process. The construction of each plan hypothesis was always triggered by some aspect of the observations. Consequently, this recognition procedure automatically insures that the plan representation is supported by the observations.

```
((PLAN Z) in-order-to (PLAN L)) =>
  [(PROPOSITION S)(Z goal S) =>
   [(Z enable L)(S enablestateof (Z enable L))
    v (Z motivate L)(S motivatestateof
                     (Z motivate L))]]
  [((Z planner) wants (L goal)) before Z)]
  [(PROPOSITION B)((B opportunity L)
   v (B reason-for L)) ~(Z outcome B) =>
        ((Z planner) believes B)]
```

The In-Order-To relation represents one relation whereby a particular Plan may be composed out of Plan units. To say that one Plan was done in order to be able to do another constitutes a claim that the two Plan units are really part of the same Planstructure. This relation defines a partial ordering capturing the non-linear nature of plans and allows different execution sequences of a single plan to be recognized.

The In-Order-To relation is represented in figure 2 by a solid arrow. This relation is shown as a pair-wise Plan unit relation. However, the definition of the psychological constraints on this relation creates the effect of giving the transitive closure of the relation. Consequently, actions (1) through (4) of figure 2 really represent the hypothesis that this sequence of actions constituted a single unit of activity for Tom.

The final relation is the Means-Of relation. This is a relation between a plan and a Planstructure. A Planstructure is simply an In-Order-To connected sequence of Plan units. The significance of the Means-Of relation is that it gives the plan hypothesis representation a hierarchical structure that reflects the goal-subgoal structure of the total plan hypothesis. This relation holds if the final goal of the Planstructure can be viewed as the goal of the higher-level Plan unit. The constraint on this Means-Of relation is expressed as:

```
((PLANSTRUCTURE I) meansof (PLAN L)) =>
  (P believes ((I goal) translatable (L goal))
```

The idea of Translatable is that if certain presuppositions of the Goal of L are supported by the context and satisfied by the Goal of I, then the Planstructure I may be viewed as an instance of the Plan L. At its simplest, the Goal of I and L may be

the summaries D through I violate Rule (1) with the possible exception of F and I. Summaries D through F were written to demonstrate violations of Rule (2) above. Summary G also violates Rule(2), but in this case the violation hinges on repeating a plan that has already been summarized at a higher level. Summaries H and I were written primarily to illustrate violations of Rule (3).

CONCLUSION

We have examined the concept of a summary of observations of human action. By viewing the understanding of human action as a plan recognition process, we have been able to show that the plan recognition process itself provides a basis for the development of an information processing account of how persons generate summaries of observed actions. A good deal of linguistic and pragmatic knowledge about human communication is still needed in order to fully specify a process of summarization. However, the plan representation discussed here provides a kind of competence theory of summarization and empirically testable statements about the structural characteristics of summaries can be derived from the plan hypothesis and tested. The development of these ideas provides an example of one kind of paradigm that can be used to bring into closer contact the concepts of representation in AI and empirical observations which reflect the way in which the human information processing system organizes information about the social world.

ACKNOWLEDGMENT

We wish to acknowledge financial support from the NIH Special Research Resource on Computers in Bio-Medicine at Rutgers. Prof. Saul Amarel who is the Principal Investigator of the Resource has always given us his expert support and criticism which has been valuable. We wish to thank Prof. C.V. Srinivasan for extended discussions which helped us understand the design concepts of his MDS. Prof. Andrea Sedlak and Frank Hawrusik have participated actively in the development of the concepts reported here.

REFERENCES

Bruce,B.C. (1975) Belief systems and language understanding. Computer Science Department, Rutgers University, NIH Report CBM-TR-41.

Bruce,B.C. & Schmidt,C.F. (1974) Episode understanding and belief guided parsing. Paper presented at the Association for Computational Linguistics Meeting at Amherst, Mass.

Schmidt,C.F. (1976) Understanding human action: Recognizing the plans and motives of other persons, in (eds J.Carroll & J.Payne), Cognition and Social Behavior. Potomac, Maryland: Lawrence Earlbaum Associates.

Schmidt,C.F. & D'Addamio,J. (1973) A model of the common-sense theory of intention and personal causation. Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, Stanford, Calif.

Srinivasan,C.V. (1973) The architecture of coherent information system: A general problem solving system. Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, Stanford, Calif. (A revised version appears in IEEE Transaction on Computers, April, 1976.)

Elliot M. Soloway
Edward M. Riseman

Computer and Information Science
University of Massachusetts
Amherst, MA 01002, U.S.A.

MECHANIZING THE COMMON-SENSE INFERENCE OF RULES WHICH DIRECT BEHAVIOR

We shall present a system that augments its a priori general understanding of human behavior in a spatio-temporal domain of physical activity, goals, competition, and gaming. Observing seemingly novel situations within that domain, it uses common-sense reasoning to construct an interpretation of the observed actions. This understanding is expressed in terms of the goals of the actors executing the actions. The system has the ability to focus its attention, as well as generation, generalize, and verify hypotheses. The desired result is a consistent structure of generalized hypotheses which represent both the regularities and an understanding of the observed domain.

INTRODUCTION

People seem able to face novel situations and within a short time understand them fairly well. For example, a child entering a new school often can quickly learn what the social structure is in his new classroom, e.g., if there is a bully or teacher's pet, who that individual is, etc. If that new student would observe a fellow classmate shoving a student out of the front of a line, and teasing another about his freckles, etc., he might reason that this fellow may be the class bully. What the child has done is use prior knowledge in order to structure his model of this new situation. By an inference process, which we would call *common-sense reasoning*, the child was able to piece together an understanding of seemingly isolated actions in terms of the common goals of the actor executing those actions. From this interpretation, the new student can predict future scenarios and develop a strategy to circumvent the power structure of the class bully. Similarly, in our daily lives, we are often faced with the behavior of humans in novel mini-situations. To explain such behavior we draw on our general understanding of people's goals in various situations in order to fit together a model of this new situation.

In this paper we will present some results of an application of this common-sense reasoning process to knowledge acquisition by the computer; it represents further development of some of the ideas previously outlined (Soloway and Riseman

1975). Our system operates in a domain of *actors* and *actions* varying over *space* and *time*. Specifically, we are developing a program that is initially given a high level description of action-oriented games (e.g., cricket, baseball, tennis, etc.). This description is expressed in terms of the *goals* and *intentions* involved in this situation, e.g., winning, scoring, etc. The description must also include the conditions that must be satisfied for actions to be counted as mediators of those goals. Driven by the observation of the activity in the game, which in our case is baseball, the system will use its general knowledge of action-oriented gaming in order to acquire an understanding of the particular goals of the people involved in this game.

As a second part of this reasoning process, the system must abstract regularities that it perceives in its world. These regularities serve as *rules* or *conventions* that govern the game and constrain the ways in which players can achieve their goals.

Not surprisingly, there is a high degree of similarity in the issues that our system must deal with and the issues that story understanding systems must face. Understanding human actions, whether read from a narrative or perceived directly, requires in addition to the understanding of the underlying goals and intentions of the actors, an understanding of the underlying causal relationships that link the actions of those actors together. Schank (1974) stresses the need for discovering via inferences the causal relationships between actions. Scripts (Schank and Abelson 1975) permit the system to 'fill-in' those causal links in stories that deal with stereotypic behavior scenarios. Since it is the task of our system to generate something like a script for baseball, our system is more akin to a constructive approach to behavior understanding (Schmidt 1976; Schmidt and Sridharan 1976; Schmidt and Goodson 1976). Schmidt argues that in order to deal with the complexity and infinite variations in human behavior, a system--be it human or machine--must ultimately be able to construct a plan(s) that serves to explain the observed behavior. This plan is expressed in such higher level terms as goals, motives, reasons, etc. Finally, in a later section we will discuss the relationship of our model of learning to those of Winston (1970), Sussman (1973), and Hayes-Roth (1976). Thus, our system uses its general knowledge about the kinds of goals and causal relationships important in action-oriented games to compose plans that explain the activity in baseball, and that capture regularities in that activity.

The following presentation will mirror the flow of information in our system, which is depicted graphically in figure 1. We will first discuss the representation of our

mini-world and the systems' knowledge of the lowest level descriptors in this representation. This will be followed by a description of the various roles of *Attention Mechanism*: (1) focusing subsequent processing on 'interesting' action sequences; (2) abstracting recurrent action sequences; (3) watching for specific action sequences that have been fed back from the higher centers of processing. Next, examples will be given which illustrate the reasoning process of the Hypothesis Generator. It is in this phase that conjectures are made as to the causal relationships between the players' activities, goals, and corresponding successes and failures. Of most importance is the formation of a coherent interpretation of the composite activity. Finally, the Hypothesis Verification and Generalization section briefly describes techniques used to gather evidence for confirmation or rejection of hypotheses, and the process of abstracting important recurrent events during the grouping of similar sequences of activity.

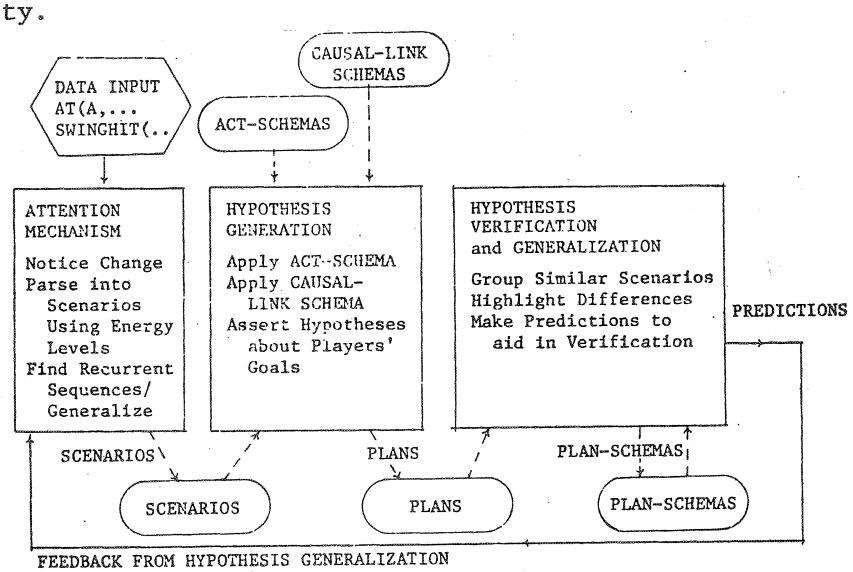


Figure 1: System Overview

Ovals indicate Data Bases and Squares indicate Procedures.

REPRESENTATION OF THE ACTIONS

In order to understand sequences of actions over time, the system must be able to understand the necessary and probabilistic changes and constancies brought about by those actions, i.e., one aspect of the Frame Problem (McCarthy-Hayes 1969; Sridharan 1976). This requires a somewhat surprising amount of detail. (figures 2 and 3). For each action the following information is grouped into an ACT-SCHEMA: (1) the direct and indirect preconditions for execution of the action; (2) the

direct and indirect consequences of the action; (3) the degree of skill and energy required to perform the action; (4) the (coarse) expected distribution of the probabilities of success/failure; (5) the general goals of the action.

```
CATCH(PLAYER,OBJECT,LOCATION)          AT(PLAYER,LOCATION)

THROW(PLAYER,OBJECT,LOCATION)          ON(PLAYER,LOCATION)

WALK(PLAYER,STARTING-LOCATION)         TEAMBOX(TEAM-NAME,OUTS,RUNS-THIS-INNING,TOTAL-RUNS)

RUN(PLAYER,STARTING-LOCATION)          BATBOX(PLAYER-NAME,STRIKES,BALLS)

FAST,SLOW,GROUND,AIR- used as modifiers on actions

SWINGHIT(PLAYER,OBJECT,LOCATION)       MOVING(OBJECT,STARTING-LOCATION,MODIFIERS)

SWINGMISS(PLAYER,OBJECT,LOCATION)      INNING(INNING-NUMBER)
```

Figure 2: Listing of Primitive Descriptor Units

The system does not understand the 'baseball' meaning of any of the primitives. In particular, it understands only that INNING, BATBOX, and TEAMBOX are scoreboards. It does not know which events correlate with the counts and does not understand the concept INNING.

```
(SWINGHIT
 (PRIMACT ((PROPEL-INANOBJ ?SNAPNUM1 ?PERSON1 ?LOCATION1)
           (GENERAL-GOALS (PROPEL-INANOBJ (XOR (TO $LOCATION5) (AWAY $LOCATION1))))
           (DIFFICULTY (HIGH SKILL) (MEDIUM ENERGY))))
 (DIRECT-PEC
  ((MUST-EXIST ((MOVING-INANOBJ (BEFORE $SNAPNUM1 ) NIL ?LOCATION2)
                ((COUNTS-AS PROPEL-INANOBJ) (BEFORE $SNAPNUM1) ?PERSON2 $LOCATION2)))
   )
  (DELTA-ENERGY-SKILL (DIFFICULTY-INCREASES-IF (FASTER MOVING-INANOBJ))))
 (INDIRECT-PEC ((MUST-EXIST ((COUNTS-AS LOCATE-ANOBJ) $SNAPNUM1 $PERSON1 $LOCATION1))))
 (DIRECT-CONSEQ
  ((MUST-EXIST ((MOVING-INANOBJ (AFTER $SNAPNUM1) NIL ?LOCATION4)
                ((COUNTS-AS LOCATE-INANOBJ) (AFTER $SNAPNUM1) ?PERSON5 ?LOCATION5)))
   (DELTA-ENERGY-SKILL (CAN-EFFECT (FASTER MOVING-INANOBJ)(FARTHER MOVING-INANOBJ)))))
 (INDIRECT-CONSEQ ((MUST-EXIST ((COUNTS-AS LOCATE-ANOBJ) $SNAPNUM1 $PERSON1 $LOCATION1)))))
```

Figure 3: ACT SCHEMA Representation of the act SWINGHIT

NOTE: In the actual implementation of the ACT SCHEMAS, the atom names preceded by ? or $ are actually function calls which serve to bind those atom names to the actual values in the action descriptor units of the scenario

REPRESENTATION OF THE SPATIO-TEMPORAL ENVIRONMENT

The game of baseball is fed to the system in a discrete form. Frozen *snapshots* of the real activity are taken at successive event times during the game. Each action descriptor unit of a snapshot captures four fundamental features of a spatio-temporal domain: action, actor, location and time. Figure 4a gives a sample of 3 snapshots in which player A1 throws a ball. Unfortunately, space does not permit a full discussion of the descriptors chosen to represent the game. Suffice it to say that, though we do not represent the color of the players eyes, or the clouds moving, the combinatorics of the many descriptors that we did chose still make the problem far from trivial. Moreover, in discussing the rest of the system, it

will be clear that many irrelevant features of this environment can be habituated out. Note that the machine does *not* initially understand in any operational sense what the meaning of the symbol INNING is. Semantic labels on locations, like home-plate, pitchers mound, etc., are equally mysterious; to the system they are (x,y) coordinates which happen to be locations of recurring activity.

ATTENTION MECHANISM: FOCUS

As in any animal's brain, our computer program must filter out most of the incoming sense impressions and pass on to the higher centers only the most interesting ones. In particular, there are 22 action descriptor units per snapshot and about 6500 snapshots per game. Therefore, the job of the Attention Mechanism is to focus attention on interesting sequence of actions and pass those on for further analysis. The system's definition of 'interesting' is biologically motivated and embodied in two ways in this front-end preprocessor: (1) it attends to sequences of snapshots where there is activity and change, and (2) it notes in particular those subsequences of actions that recur.

The first characteristic translates into an algorithm which filters out non-activity (AT, ON in our case) while highlighting activity chains. The amount of data reduction using this algorithm is quite significant. Figure 4b illustrates the application of this filtering algorithm to the snapshots of figure 4a.

COMPLETE SNAPSHOTS:

TIME:	14	15	16
	HOLDOBJ(A1,BALL,PM)	THROW(A1,BALL,PM)	AT(A1,PM)
	AT(A2,HP)	AT(A2,HP)	AT(A2,HP)
	AT(A3,FB)	AT(A3,FB)	AT(A3,FB)
	.	.	.
	.	.	.
	AT(A9,RF)	AT(A9,RF)	AT(A9,RF)
	AT(B1,HP)	AT(B1,HP)	AT(B1,HP)
	AT(B2,DUGOUTB)	AT(B2,DUGOUTB)	AT(B2,DUGOUTB)
	AT(B3,DUGOUTB)	AT(B3,DUGOUTB)	AT(B3,DUGOUTB)
	.	.	.
	.	.	.
	AT(A9,DUGOUTB)	AT(A9,DUGOUTB)	AT(A9,DUGOUTB)
	INNING(1)	INNING(1)	INNING(1)

Figure 4a: Partial raw, prefiltered snapshots

REDUCED SNAPSHOTS:

TIME:	14	15	16
	HOLDOBJ(A1,BALL,PM)	THROW(A1,BALL,PM)	AT(A1,PM)

Figure 4b: Remaining primitive descriptor units after snapshots are filtered by attention mechanism.

This filtered data must now be further structured. The continuous action stream must be parsed into relatively small chunks, much like words in a paragraph are chunked into phrases or sentences. The heuristic to perform this task is suggested by the following observations of an action-oriented environment (game): (1) a flurry of activity often indicates that some cohesive process is taking place (competition), while (2) relative calm often indicates the completion of that process (resolution of competition) and the lull prior to the next spate of possibly relevant activity (another round of competition). This crude heuristic does partition snapshots into meaningful chunks. A semantic routine during a later phase will sharpen the boundary points of the activity.

In order to notice repetition of relatively similar sequences of events, generalization over various parameters of the action descriptor must be performed. For example, implicit in finding repetitive events is a generalization over absolute time. By using the *generalization operators* (figure 5), the system can abstract repetitive subsequences of actions within scenarios. This permits the system to build up, in a hierarchical structure, complex sequences of events into more complete scenarios. Then, instead of seeing isolated actions, the system can eventually perceive these complex sequences as if they were single action units, e.g., perceiving a batter's hitting and running as simply a 'hit' or a fielder's catching the ball and throwing it as a 'fielding play.'

Original descriptor unit:	THROW(A1, FROM-PM, BALL)
Generalization of descriptor unit	
Person Operator:	THROW(ANY-PERSON, FROM-PM, BALL)
Place Operator:	THROW(A1, FROM-ANYPLACE, BALL)
Person and Place Operators:	THROW(ANY-PERSON, FROM-ANYPLACE, BALL)

Figure 5: Syntactic generalization operations

NOTE: There is implicit generalization over time.

HYPOTHESIS GENERATOR: PLANS

The major goals of the competitors in an action-oriented game can be expressed (roughly) as follows:

GOAL: Both teams are trying to win. A team can win only by 'scoring' more than the opposing team. This implies both offense and defense.

GOAL: The players on each team try to help members of their own team and try to hinder members of the opposing team from achieving their goals.

We characterize competition in the following way: (1) acts that are considered competitve often require a medium to high

degree of skill and/or energy; (2) causal relationships that link the actions of opposing teams will highlight subgoals. They are also the basis for determining the successes and failures of the teams with respect to those subgoals. Examples of such causal relationships, called *CAUSAL-LINK SCHEMAS* (sometimes denoted CLS) are given in figure 6.

CAUSAL-LINK SCHEMA & TRIGGERING CONDITIONS	HYPOTHESES MADE
PHYSICAL-CONFLICT (P-CONFLICT)	
a. Action ACT1 executed by P1 was the direct physical enabling condition for action ACT2 executed by P2 b. P1 and P2 are on opposing teams c. DIFFERENTIAL-ANALYSIS returns T by finding some way that P1 could have performed ACT1 so as to (decrease) the likelihood of P2 executing ACT2	a. P1 did not intend that P2 execute ACT2 b. P1 failed to prevent P2 execute ACT2 c. P2 intended to execute ACT2 d. P2 succeeded
PHYSICAL-COOPERATION (PHYS-COOP)	
a. same as a. above b. P1 and P2 are on the same team c. same as c. above, except substitute (increase)	a. P1 intended to help P2 b. P1 succeeded if P2 succeeded, but not necessarily conversely
LOGICAL-COOPERATION (LOG-COOP)	
a. NOT a. above, yet ACT1 must precede ACT2 b. P1 and P2 are on the same team, and may in fact be the same person	a. execution of ACT2 required the execution of ACT1 b. P1 intended to execute ACT1 c. P1 succeeded
RELATIVE-TIME (REL-TIME)	
a. P2 executed ACT2 after P1 executed ACT1, and ACT1 and ACT2 are not linked by physical enabling conditions b. P1 and P2 are on opposing teams c. there exists actions ACT1* and ACT2* that DIFFERENTIAL-ANALYSIS says could have permitted P1 and P2 to execute acts ACT1 and ACT2 sooner	a. P1 was allowed to execute ACT1 because P1 executed ACT1 before P2 executed ACT2 b. P2 intended to execute ACT2 before P1 executed ACT1 c. P1 succeeded d. P2 failed
LOGICAL-CONFLICT (L-CONFLICT)	
A. Change of Action a. P2 changed from executing ACT2 to ACT2' b. P1 executed ACT1 concurrently with ACT2 c. P1 and P2 are on opposing teams	A. Change of Action I. a. P1 succeeded by executing ACT1 for some goal which forced P2 to execute ACT2' b. P2 did not intend to execute ACT2' c. P1 succeeded d. P2 failed or, II. a. P2 succeeded by executing ACT2 for some goal, and therefore intended to execute ACT2' b. P1 failed to do something which could have prevented P2 from succeeding c. P2 succeeded d. P1 failed or, III. a. P2 succeeded by executing ACT2 for some goal, and therefore intended to execute ACT2' b. P1's ACT1 is NOT causally linked to the actions of P2, i.e., ACT1 and ACT2 are independent

Figure 6: List of CAUSAL-LINK SCHEMAS

All the hypotheses for a CAUSAL-LINK SCHEMA are asserted when all the triggering conditions are satisfied.

In the analysis of the scenario passed to it by the Attention Mechanism, Hypothesis Generation proceeds by first applying the appropriate ACT-SCHEMA to each action descriptor unit in the scenario. This process establishes the precondition

links for the action. It is followed by the application of the CAUSAL-LINK-SCHEMAS (e.g., PHYSICAL-CONFLICT, RELATIVE-TIME, PHYSICAL-COOPERATION, LOGICAL-COOPERATION, etc.). These demon-like routines (Charniak 1972) search for sequences of actions that satisfy conditions specified in each CLS. Meeting of the conditions implies that the causal relationship specified by the particular CLS *may* exist between those actions. Once triggered, they make hypotheses about the goals of the players and about the success or failure of the players with respect to those goals.

During a third phase, isolated actions and their hypothesized goals are grouped by player and team into *PLANS*. A plan is a very important aspect of human behavior. It permits a coherent *interpretation* of a sequence of actions. Intermediate actions serve as means for attaining subgoals, while subgoals are executed in order to achieve the final goal (often indicated by the last action).

Let us illustrate the application of the CAUSAL-LINK SCHEMAS. Consider the scenario in Figure 7, which depicts an infield single. In the analysis of action descriptor #6 (the SWINGHIT by player B1), the PHYSICAL-CONFLICT SCHEMA finds itself applicable. As a result of the ACT-SCHEMA, the system knows that the THROW by opponent A1 'direct-physically' enabled B1 to hit the ball, and that both actions require a high degree of skill. Before labeling success or failure, PHYSICAL-CONFLICT must be sure that the performance of the player who threw the ball definitely had some effect upon the performance of the action of the hitter. DIFFERENTIAL-ANALYSIS confirms this possibility by accessing data in the ACT-SCHEMA, and accessing general facts about actions in the data base. In this case DIFFERENTIAL-ANALYSIS infers that A1 could have thrown the ball faster by applying an increase of energy and skill. (Note that skill or energy is usually expended by a person whose motive is to achieve an action-oriented goal.) This would have had the effect of requiring a corresponding increment of skill for B1's action. The point is that A1 could have done something to decrease the likelihood of B1's hitting the ball. In this case PHYSICAL-CONFLICT makes the following hypotheses: (1) A1 did not intend that the effects of his actions should allow B1 to hit the ball, therefore A1 failed with respect to his goal; (2) B1 intended to execute the act SWINGHIT, therefore B1 succeeded with respect to this goal.

Consider act descriptor #20 (ON firstbase by B1). The CAUSAL-LINK SCHEMA of RELATIVE-TIME asserts the possibility that the reason B1 was allowed to execute that action was because he did it *before* action descriptor #22 (CATCH ball at firstbase by A3). Again DIFFERENTIAL-ANALYSIS is called to

confirm that the time of execution of both acts could have been influenced by a change in the skill or energy expended. Yes, A6 could have made the ball arrive at firstbase sooner if he could have thrown the ball faster; and B1 could have arrived at firstbase sooner if he had run faster. The hypotheses of this CLS are: (1) B1 did intend to execute act ON firstbase, therefore B1 succeeded with respect to his goal; (2) A1 did not intend for B1 to execute that act, therefore he failed with respect to his goal.

Team A	Team B	Ball
**(1 1 HOLDOBJ (A 1) PM BALL)		
**(2 2 THROW (A 1) PM BALL) ((PHYS-COOP SUCCEED)(PEC 1 2))		
**(3 3 AT (A 1) PM NIL) ((LOG-COOP SUCCEED)(LOGICAL 2 3))	**(4 3 AT (B 1) HP NIL)	**(5 3 MOVING NIL PM BALL (FAST AIR))
	**(6 4 SWINGHIT (B 1) HP BALL) ((P-CONFLICT SUCCEED)(SUCCEED 6)(FAIL 2)) ((LOG-CONF SUCCEED)(CHANGE-LINK 3 4))	
	**(7 5 RUN (B 1) HP (FAST)) ((LOG-COOP SUCCEED)(LOGICAL 6 7))	**(8 5 BOUNCE NIL HP BALL)
**(9 6 AT (A 6) TB NIL)	**(10 6 RUN (B 1) HP (FAST)) ((PHYS-COOP SUCCEED)(PEC 7 10))	**(11 6 MOVING NIL HP BALL (FAST GRND))
**(12 7 CATCH (A 6) TB BALL) ((P-CONFLICT SUCCEED)(SUCCEED 12)(FAIL 6 ((LOG-CONF SUCCEED)(CHANGE-LINK 10 9))	**(13 7 RUN (B 1) HP (FAST)) ((PHYS-COOP SUCCEED)(PEC 10 13)) ((LOG-CONF SUCCEED)(NOCHANGE-LINK 10 9))	
**(14 8 THROW (A 6) TB BALL) ((PHYS-COOP SUCCEED)(PEC 12 14)) ((LOG-CONF SUCCEED)(CHANGE-LINK 13 12))	**(15 8 RUN (B 1) HP (FAST)) ((PHYS-COOP SUCCEED)(PEC 13 15)) ((LOG-CONF SUCCEED)(NOCHANGE-LINK 13 12))	
**(16 9 AT (A 6) TB NIL) ((LOG-COOP SUCCEED)(LOGICAL 14 16)) ((LOG-CONF SUCCEED)(CHANGE-LINK 15 14))	**(17 9 RUN (B 1) HP (FAST)) ((PHYS-COOP SUCCEED)(PEC 15 17)) ((LOG-CONF SUCCEED)(NOCHANGE-LINK 15 14))	**(18 9 MOVING NIL TB BALL (FAST AIR))
**(19 10 AT (A 3) FB NIL)	**(20 10 ON (B 1) FB NIL) ((PHYS-COOP SUCCEED)(PEC 17 20)) ((REL-TIME SUCCEED)(SUCCEED 20 FAIL 22)) ((LOG-CONF SUCCEED)(CHANGE-LINK 16 17)) ((EOC SUCCEED)(END-OF-COMPETITION))	**(21 10 MOVING NIL TB BALL (FAST AIR)
**(22 11 CATCH (A 3) FB BALL) ((PHYS-COOP SUCCEED)(PEC 14 22)) ((LOG-CONF SUCCEED)(CHANGE-LINK 20 19))		
**(23 12 HOLOBJ (A 3) FB BALL) ((PHYS-COOP SUCCEED)(PEC 22 23)) ((EOC SUCCEED)(END-OF-COMPETITION))		
**(24 13 THROW (A 3) FB BALL) ((PHYS-COOP SUCCEED)(PEC 23 24))		
**(25 14 AT (A 3) FB BALL) ((LOG-COOP SUCCEED)(LOGICAL 24 25))		**(26 14 MOVING NIL FB BALL (SLOW AIR))
**(27 15 AT (A 1) PM BALL) ((LOG-CONF SUCCEED)(NOCHANGE-LINK 3 4))		**(28 15 MOVING NIL FB BALL (SLOW AIR))
**(29 16 CATCH (A 1) PM BALL) ((PHYS-COOP SUCCEED)(PEC 24 29))		
**(30 17 HOLDOBJ (A 1) PM BALL) ((PHYS-COOP SUCCEED)(PEC 29 30))		

Figure 7: Infield Single Scenario with Asserted Hypotheses

This listing reflects only successful CAUSAL-LINK SCHEMAS; not shown are the reasons returned when they fail. The syntax of the action descriptor unit is:

(descriptor-number snapshot-number action actor location modifiers)

The columns represent all actions of a team or object, while rows reflect concurrent actions (equal snapshot numbers).

Note that the two CLS's above 'perceive' competition on two different levels. PHYSICAL-CONFLICT actually observes the physical interaction between the actions of the players, while RELATIVE-TIME must posit the existence of a relationship between the actions of the players. Of course, both require that their respective relationships exist between members of opposing teams. PHYSICAL-CONFLICT deals with a specific

feature of the physical environment, e.g., skill and energy, while RELATIVE-TIME deals with a specific feature of the logical environment, e.g., time precedence as a relevant relationship.

There may be additional features in either (or both) levels that the system cannot directly perceive, but which are nonetheless important to the specific game under observation. For example, since the system does not perceive the specific placement of the ball as it passes over the homeplate, it will not be able to perceptually distinguish between a 'called strike' and a 'ball'. (To be a 'called strike', the ball must pass over homeplate at a height somewhere between the batter's shoulders and his knees.) The general CAUSAL-LINK SCHEMA, LOGICAL-CONFLICT, deals with this type of situation. This general CLS looks at the changes (and non-changes) in players' actions, and tries to explain those changes (or non-changes) in terms of the existence of some causal relationship, even though it cannot directly perceive one. This decision is based on an understanding of the kinds of causal links that might be necessary to explain a player's actions. With this ability, the system has a flexible and powerful technique for dealing with novel situations.

Generation of plans will ultimately require the binding of many of the local hypotheses distributed across the action scenarios. The system must find interlocked and globally consistent subsets of inferences which might explain the observed situation. For example, during the PLAN building phase, another hypothesis generator, EOC, attempts to find the end of a competitive epoch so it can highlight the final goals of the two teams. In figure 7, EOC hypothesizes that RUN-ON and CATCH-HOLDOBJ are the last competitive acts of the two teams. A process called PLAN-BUILD then backs up the final goals and attempts to relate them--the SWINGHIT of B1 was executed in order to enable B1 to execute the act ON first-base. Similarly, the A team's goal now has become: prevent B1 from executing ON firstbase. We are presently investigating other more dynamic techniques to assist in this analysis and transformation of local hypotheses into globally consistent plans.

This is only the first stage of hypothesis generation. We have not used all the concepts from action-oriented gaming. For example, we have not as yet made hypotheses about what counts as scoring, or what counts as failed opportunities to score. Nor have we started to keep track and tally up these kinds of actions, usually an important facet of scoring. Nor have we introduced the high information cue of spectator cheering. This latter stage of hypothesis generation will build on the hypotheses made so far, but will have to wait until after the next phase, hypothesis verification and

generalization, where evidence will be gathered to support or reject those earlier hypotheses. Note, however, that our model of knowledge acquisition, using a general description of action-oriented gaming, has already moved from the perception of actions to the possible goals intended by those actions.

HYPOTHESIS VERIFICATION AND GENERALIZATION: PLAN-SCHEMAS

Exemplar learning models (e.g., Winston 1970, Sussman 1973, Hayes-Roth 1976), usually have the following characteristics. First, they may require a partially ordered training sequence with presentations of positive and negative instances of the class in order for the desired concepts to be properly learned. Next, such models require that the system be told to which class an instance belongs. This is usually done either explicitly by associating the class name with the presentation, or implicitly by requiring that the trainer present examples of only one class at a time. Third, the set of relationships used in generalization is basically the same as used in the examples. Finally, a local similarity measure (e.g., frequency of occurrence) relating examples of a class is used to define a generalized class description.

However, abstracting regularities of human behavior by simply observing that activity in a natural setting requires a more sophisticated model of unsupervised learning. The complexity of our problem domain requires an extrapolation of the above issues in the following ways. First, natural experience is often a fickle teacher. A model that learns from experience must be flexible enough to accept an unordered training sequence and impose its own order. Second, in a new experiential domain, the system cannot expect to know or be told to which class an example belongs; it must be able to *infer* the classes, using both a priori knowledge of what could count as a class type and the observations of specific examples. Third, given the multiplicity and non-specificity of features in any given real-world situation, a priori semantic knowledge is required in order to hypothesize the existence of higher level relationships that serve to highlight relationships that are important to a specific interpretation. For example, in our domain, the CLS posit the existence of relationships that are important to interpreting that activity in the context of action-oriented games. Other relationships would need to be hypothesized if the system were trying to interpret that activity as a religious ceremony. The above properties characterize the 'experiential model of learning' employed in the present system. We believe that it examines issues underlying human developmental learning that previous systems have not addressed.

In order to first generate classes and then generalize

within those classes, two types of similarity measures are required. A more global one that can partition examples into classes, and another more local one that can abstract the important characteristics within a class. The global criteria under which we have chosen to group scenarios stems from a simple but powerful observation: events that begin the same but end differently, events that begin differently and end the same, and events that begin and end the same but have different middles--are cues to the structure of the general scenarios which govern those situations.

Using these principles the system will be able to group together a set of scenarios that will eventually be labeled as infield singles, and a set of scenarios that will eventually be labeled as infield groundouts. Comparing these two groups using the 'begin the same, end differently' heuristic, we note that they do only differ at the end. Looking at the hypothesized interpretations in each group, we saw that the team which was labeled as having succeeded in one group is the same team that was labeled as having failed in the other group (figure 8). This result not only lends support to a correct partitioning of the scenarios, but also supports the correctness of the hypotheses.

A powerful input during this phase of analysis comes from the CAUSAL-LINK SCHEMAS themselves. If their hypotheses about the observed actions are correct, then they should expect to see action sequences in which the team that was previously hypothesized as being successful should now fail, and vice versa. In fact, predictions are made of precisely these complementary outcome conditions, and prime the Attention Mechanism via feedback. From our example (figure 7), PHYSICAL-CONFLICT will predict that A1 will throw the ball, and some B team member will not hit it. RELATIVE-TIME will predict that A3 will catch the ball before some B team member reaches first-base, and that some B team member will not be permitted to execute the act 'ON firstbase'.

The convergence of both techniques to similar conclusions is strong evidence for the correctness of the hypotheses and the partitioning. Finally, just as the Attention Mechanism is building up generalized, repetitive subsequences of actions, this stage of the system will generalize PLANS into *PLAN-SCHEMAS*, based on its grouping of scenarios. These latter structures will be the final output from the system. They will represent both an understanding of the goals and intentions involved in the scenario, and also the rules or regularities observed in the scenarios.

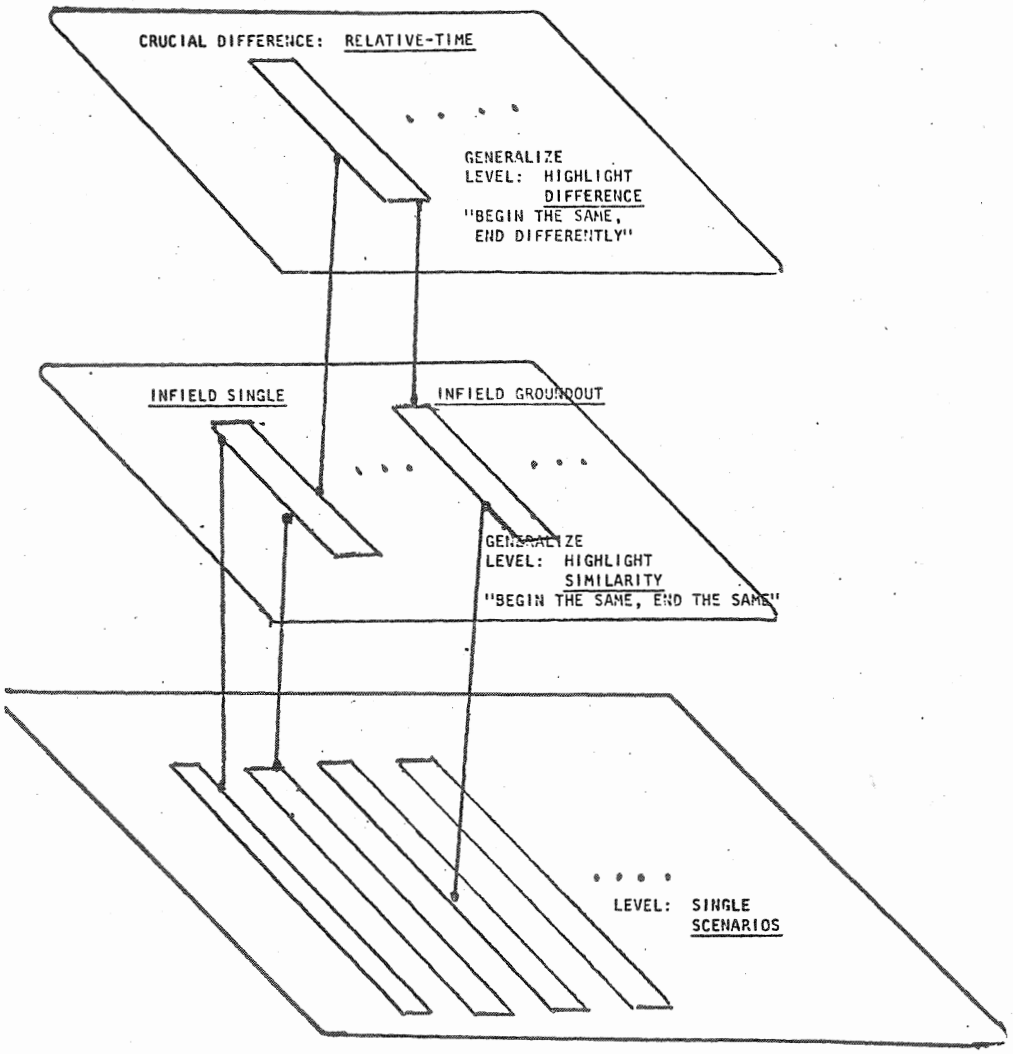


Figure 8: Hierarchical Generalization of Scenarios

SUMMARY

The implementation of the Attention Mechanism was done in SNOBOL4. The output from this subsystem is used by the Hypothesis Generation phase which is written in LISP. In the analysis of a typical scenario, this program uses approximately 27K of the CDC6600 and requires about 30 seconds of processor time. A preliminary version of the Hypothesis Generalization and Verification system is presently being built and tested. Results from the Hypothesis Generation part of the system confirm its inferential power, while early results from the hypothesis generalization part of the system are also

encouraging.

What we have explored is a way in which a system can add to its present knowledge base an understanding of some new situations. Though initially driven by the perception of knowledge about actions, the system develops a consistent structure of hypotheses about the goals of the actors. This conceptual representation permits the system to abstract the commonalities from a multiplicity of somewhat varying situations. This generalization process provides the general rules governing behavior in the observed situations of this environment.

ACKNOWLEDGEMENTS

This work has been aided by the stimulating dialogue within the Cybernetics group at the University of Massachusetts. The many discussions with Chuck Schmidt of Rutgers University has served to significantly sharpen our understanding of the issues. We also wish to thank W. Kilmer, M. Arbib, J. Lowrance, and D. Lawton for helpful comments during the preparation of this paper.

REFERENCES

Arbib,M.A. (1975) Two papers on schemas and frames. Univ. of Mass. COINS Technical Report 75C-9, Amherst.

Charniak,E. (1972) Toward a model of children's story comprehension. MIT AI Lab Technical Report AI-TR266, Cambridge.

Hayes-Roth,F. (1976) Uniform representation of structured patterns and an algorithm for the induction of contingency-response rules. *Information and Control*, in press.

McCarthy,J. & P.Hayes (1969) Some philosophical problems from the standpoint of artificial intelligence, in *Machine Intelligence 4* (ed D.Michie). Edinburgh: Edinburgh Univeristy Press.

Rieger,C. (1973) The common-sense algorithm as a basis for computer models of human memory, inference, belief and contextual language comprehension. Theoretical Issues in Natural Language Processing, June, Cambridge.

Rumelhart,D.E. (1975) Notes on a schema for stories, in *Representation and Understanding* (eds D.Bobrow & A.Collins). New York: Academic Press.

Schank,R.C.(1974) Understanding paragraphs. Instituto per gli Studi Semantici e Cognitivi, Technical Report 5, Castognola, Switzerland.

Schank,R.C. & R.P.Abelson (1975) Scripts, plans and knowledge. Advanced Papers *IJCAI 4*, USSR.

Schmidt,C.F. (1976) Understanding human actions: recognizing the plans and motives of other persons, in *Cognition and Social Behavior* (eds J.Carroll & J.Payne). Erlbaum Press, in press.

Schmidt,C.F. & J.L.Goodson (1976) Summarizing observed actions: a structural hypothesis about the nature of summaries. Dept. of Computer Science Technical Report CBM-TR62, Rutgers Univ., New Brunswick.

Schmidt,C.F. & N.S.Sridharan (1976) The representation of plans. Dept. of Computer Science Technical Report CBM-TR61, Rutgers Univ., New Brunswick.

Soloway,E.M. & E.M.Riseman (1975) Common-sense theory formation: towards understanding baseball. Univ. of Mass COINS Technical Report 75C-5, Amherst.

Sridharan,N.S. (1976) The architecture of BELIEVER, PART II: the frame problem. Dept. of Computer Science Technical Report CBM-TR47, Rutgers Univ., New Brunswick.

Sussman,G.J. (1973) A computational model of skill acquisition. MIT AI Lab Technical Report TR-297, Cambridge.

Winston,P.H. (1970) Learning structural descriptions from examples. MIT Project MAC Technical Report 76, Cambridge.

N. S. Sridharan

THE FRAME AND FOCUS PROBLEMS IN AI: DISCUSSION IN RELATION TO THE *BELIEVER* SYSTEM.

ABSTRACT

There are two types of reasoning about actions - *Planning* and *Interpretation*. We are involved in the Interpretation of observed sequences of actions using a psychological theory of act interpretation. The Interpreter uses a model space to maintain a current description of the observational world. There are two kinds of problems in maintaining a model - *Consistency* with respect to what we know about the domain and *Correctness* with respect to an external observable world. Past discussions of the problem have dealt mainly with the issue of consistency. The issue of ensuring correctness is a difficult one and this paper proposes the *Heuristic* of using the detected inconsistencies of the model state as a guide to selectively removing disparities between the model state and the observable world. A Knowledge Representation Framework is described which allows all aspects of the Frame Problem to be tackled coherently. An example is given to illustrate our approach and an indication of the rule learning capability of the system is given. The Focus problem for the Observer is introduced and is shown to be complementary to the Frame Problem for the modelling system.

A provocative and central issue in AI for the past six years has been the Frame Problem. (McCarthy and Hayes 1969; Raphael 1971; Sandewall 1973; Fikes, Hart and Nilsson 1973; Hayes 1973). The *FRAME PROBLEM* is concerned with the maintenance of *consistency and correctness* of the model state when updating it or when constructing a new model state in response to an initial description of the observed world. It specifically excludes the more difficult problem of finding and correcting errors in the domain knowledge itself. Although much of the literature concerns the representation of actions in a robot's domain, the problem is common to any model-based reasoning system (Amarel 1974). Some examples are

key phrases
the Frame Problem; the Focus Problem; Modelling; Model Maintenance; Belief Systems Modelling; Plan Recognition; Plan Generation; Problem Solving; Knowledge Representation; Knowledge Acquisition; Knowledge Representation Language.

Natural Language Question Answering (Sandewall 1971), Chemical Synthesis Planning (Sridharan 1971), Generative CAI (Carbonell and Collins 1973), and our work on Belief Systems Modelling (Schmidt 1975, Sridharan 1975).

Any system or person that has to reason about actions faces the Frame Problem in that he has to represent knowledge and to reason about both the *primary effects* of certain categories of action and the *contingent changes* that take place when that action is performed in a given state of the world. There are two types of reasoning about actions. The first is the problem of planning a sequence of actions [PLANNER] to achieve a desired goal state. The second is the problem of observing a sequence of actions and being able to interpret them meaningfully [INTERPRETER]. The first problem has been worked on by researchers who work in Robotics. We on the other hand, are interested in the second problem, for the task of BELIEVER Systems (Sridharan 1975) is to interpret sequences of observed actions.

A *Planner* is set the task of using a predefined collection of actions (Transformations) to *plan* a course of action (often only a sequence of actions) that will achieve a stated desired *goal state* from a stated *present state*. The actions are represented to the system in terms of their *primary predictable effects* on the state to which they are applied. However, there is a considerable amount of knowledge about the *contingent effects* of the action that is not easily represented and used. When the Planner uses a model to plan the course of actions it might use only the primary effects of actions to sketch out a plan (*). But the later elaboration of this plan must carefully consider the contingent effects of the actions as well.

An *Interpreter* receives *action descriptions* from an *Observer* who takes in input from an observational world [Figure 1]. The Interpreter is given the task of *recognizing the plan* structure that underlies the action sequence and of *predicting* possible future actions in terms of predefined action types (See Schmidt and Sridharan 1976; this volume for a discussion of our representation of Plans). The Interpreter uses a model space to maintain a current description of the observational world. Our concept of a *Model Space* involves three components: the conceptual framework and the *Domain Knowledge* that determines consistency of the model state; a

*See (Sacerdoti 1975) for some interesting ways of accounting for the contingent effects during planning.

given *Model State* which provides a description of the observable world; and a collection of *Rules* that prescribe ways of updating the model state. The model state is a collection of *relational triples* describing the state of the observational world e.g., (MARY loc KITCHEN) (MARY holds CUP) etc.. It also uses a *Hypothesis space* to maintain and reason with hypothesized plans and intentions of actors. The present treatment concerns the maintenance of consistent and correct model states. The need to entertain inconsistent hypotheses and resolve them is handled in the Hypothesis space and will not be treated here.

AN APPROACH TO THE FRAME PROBLEM THROUGH THE DESIGN OF A KNOWLEDGE REPRESENTATION LANGUAGE

The modelling system attempts to maintain a collection of entity descriptions and propositions that captures relevant information either about an observational world (for the Interpreter) or about a projected reality (for the Planner). The problem in maintaining a model concerns ensuring *consistency and correctness*. Past discussions have not distinguished clearly between the problem of consistency/inconsistency of a model state and the correctness/disparity between an external world and the model. Discussions of the Planner and its frame problem are necessarily confined to issues of maintaining consistency and have no scope for introducing the correctness issue except when dealing with planning in an environment of execution. The Interpreter, however, must squarely face the problem of disparity and our treatment will clearly entertain this distinction.

Let us start by assuming that at some given instant the model state is both consistent and free of disparity with the observational world.

Introduction Using an Example

Consider the action of a person MARY who GOES from location KITCHEN to a target location LIVING ROOM. How does one derive a "Natural" description of the act of going in the very simple sense of merely transferring location? The PRECONDITION for the act requires that Mary be in the Kitchen. The *primary effect* of GOing is that Mary is in the Living Room and this is stated as the GOAL of the act.

If however, you know she is holding a cup while in the kitchen, you immediately recognize the *contingent effect* of GOing, and a reasonable statement of this contingent effect is (for this instance)

"If Mary is holding a cup in the Kitchen while she goes to the living room, then she is still holding the cup, with

the cup now being in the livingroom".

Or more abstractly,

"If a person P is holding an object J and goes to a location L then object J will also be transferred to that location L and person P will remain holding the object J".

This statement can be conveniently written in two parts:

[Recognition Rule] When a Person P changes location from L1 to L2, collect propositions about whatever Object J that P is holding.

[Resolution Rule] In the above situation, assert that the Object J changes location to L2, and P still holds J.

This way of dividing Frame rules is a minor extension of the concept of Production rules, where the Recognition Rules test a condition and specify a chunk of the model state on which Resolution rules may be applied, the Resolution rules contain the action specification that computes the Frame effects.

We give to the Model Space knowledge about the terms used in the domain in a form that permits the system to *extract automatically* the dependency between the location and holding in the form of a Recogntion Rule.

The following expressions define the classes of entities and names of relations among them,

```
[GO
(agent  PERSON  agentof)
(to-loc LOCATION  to-locof)
(from-loc  LOCATION  from-locof)
(goal  [PROPOSITION ((X agent)  loc  (X to-loc))] goal of)]

[PERSON
(loc   LOCATION   locof)
(holds OBJECT     heldby)]

[OBJECT
(loc     LOCATION   locof)
(heldby  PERSON     holds)]
```

defined by using the format:

[< Classname> (< Relationname> < Classname> <Inverserelation>)...]

The class names and relation names constitute the VOCABULARY of the domain out of which the Domain Language (the mini-language for this example)is constructed.

GO is declared to be an entity with four relations associated with it, viz., agent who is a PERSON, from-loc and to-loc

which are LOCATIONs and an outcome which is a PROPOSTION. The outcome proposition is defined to be one that asserts that the agent of the GO is in the to-loc location.

Furthermore the semantic of a PERSON holding an OBJECT is defined separately as

((PERSON X) holds (OBJECT J)) =>
        ((SOME LOCATION Z) (X loc Z) (J loc Z))

The system can (structurally) analyze the definition of a relation "holds" given above and extract the Recognition Rules. For example, the dependency between (PERSON loc LOCATION) for a Person X and a Location L (as specified in the DIMENSION of the rule), and (PERSON holds OBJECT) is the expression (OBJECT Z) (X holds Z),

[Recognition Rule]
(DIMENSION (OBJECT O) (LOCATION L))
(ASSERTING X loc L)) (INSPECT ((PERSON P) (P holds O)))

i.e. when an object changes location any assertion concerning some person holding O is affected.

The language (notation) for writing the Recognition rules is the same as that in which the Domain knowledge is written by the user. This simplifies the task of composing the Recognition rules by re-representing the Domain knowledge.

The following commands generate the simple model state in which we can investigate the action GO.

(IT (PERSON MARY)) [IT declares an instance of an entity class]
(IT (OBJECT CUP))
(IT (LOCATION KITCHEN))
(IT (LOCATION LIVINGROOM))
(IR (MARY loc KITCHEN)) [IR instantiates a relational triple]
(IR (CUP loc KITCHEN)
(IR (MARY holds CUP))

The last assertion made to the model space causes its associated defintion to be evaluated and the residue (cf. Srinivasan 1976) of the evaluation which points to (*) (MARY loc KITCHEN) and (CUP loc KITCHEN). The residue is to be interpreted as saying that a change in one of the propositions in the residue [in this case (MARY loc KITCHEN) or (CUP loc KITCHEN)] will cause the system to examine the validity of assertion to which

*The fact that CUP is an OBJECT and that MARY is a PERSON are necessary to assert (MARY holds CUP), but do not appear in the residue for they are properties that are fixed with respect to the domain, i.e. they cannot change and are not relevant to the updating process.

it is attached [in this case the (MARY holds CUP) assertion].

Consider the action that Mary does as described by the Observer by means of the description:

(INSTANTIATE (GO (agent MARY) (from-loc KITCHEN) (to-loc LIVING-ROOM))). The goal relation defined for GO is used to cause the primary change to the model state, which updates Mary's location to be Living-room. The recognition rule is consulted which in this case collects the single proposition (Mary holds Cup) as the only triple that needs to be inspected. The location of the cup has not yet been updated and thus evaluating the definition of the 'holds' relation shows that there is an inconsistency in this partially updated state of the model.

The resolution rules are invoked for that subset of the propositions that lead to inconsistency in an attempt to remove such inconsistency. The form of the Resolution rule as given to the system is given below.

Resolution Rule for the dependency of 'holds' on 'loc'.

```
(DIMENSION   (PERSON P)(OBJECTJ)(LOCATION L1))
(ASSERTING   (P loc L1))
(INSPECTING (P holds J))
(RESOLVE-USING (IR(J loc L1)(P holds J)))
```

The applicability of the Resolution rule is checked with the first three subexpressions of the rule, and when checked out, the assertions specified by the rule would restore the consistency of the model state. In this instance, the application of this rule and the subsequent update make the resulting state consistent by updating the Cup's location and maintaining that Mary still holds the Cup.

Though the rule appears like a categorical imperative, one should appreciate the essential fragility of the rule because the rule gets applied to a model space and not a passive data base. That is, if the Resolution rule submits (J loc L1) as an update, the model space may refuse to accept it if it makes the model state inconsistent. In that case the residue from that refusal will guide the Model Space in its further action.

If an inconsistency is discovered for which there is no resolution rule the observer is queried for factual information thus entering into a mode of comparing the model state with the observational world but guided by domain knowledge and the recognition rules. In a separate report (Sridharan 1976) we explore the possibility of the system summarizing its experience when in this mode and learning a generalized Resolution rule that would be applicable and useful in the future. Here again,because the language in which the rules

are summarized and generalized is the same as the one in which the domain knowledge was given initially there is no difficulty in using learned rules during the modelling and interpretation process. This condition is essential for a meaningful exploration of learning methods to be carried out.

Recognizing and Resolving Inconsistencies

Our approach to ensuring consistency of the model state is to utilize a system architecture that *separates the problem solving structure from aspects of model maintenance.* The model space is given explicit charge of maintaining consistency when updating the model state.

When asserting a new proposition to the model state, the model space attempts to ensure that the resulting state is consistent, by accessing the collection of propositions that are directly inconsistent with the change made. This is accomplished using *RECOGNITION RULES* that map a given proposition to a collection of other propositions in the model state. These propositions collected by the Recognition Rules are the only propositions that need to be checked for ensuring the consistency of the model state.

If after making the primary changes the model state is found to contain inconsistencies, then there are alternative ways of restoring consistency, and futher computational rules need to be invoked to select a method of resolving the inconsistency in the model state. Rules which accomplish this are the *RESOLUTION RULES.* By the generic term *FRAME RULES* we mean both the Recognition and Resolution rules.

It is a characteristic of our system that the Recognition Rules can be computed by the system from the domain knowledge as stated initially by the user, but the Resolution Rules must be supplied by the user.

It is obvious that the actual content of the Frame Rules is dependent on the domain that is being modelled. It is not equally obvious but true that the *method* of representation, of recognition, of resolution or of acquisition can be made free of domain semantics.

Recognizing and Resolving Disparities

It is important to realize that there is no foolproof shortcut to recognizing disparities between the model and the world - the only failsafe route is via a thorough examination of the world. We suggest a *heuristic* that couples together the problem of internal consistency and disparity. When there is no inconsistency detectable in the model state the system can pretend that there is also no disparity. However,

when there is inconsistency detected *the domain structure can be used to guide selective comparison of the model state with the external world.*

Thus when updating the model state, the recognition rules are accessed to discover the inconsistencies. When inconsistencies are discovered that have no applicable Resolution rules then an appeal to the observational world is made for factual information. The recognition rules play a second important role in directing the search for disparities between the model and the world by serving to focus on relevant propositions. When all such inconsistent propositions have been queried to the Observer and the model state consistency is restored, the state can be considered to be a correct description of the world. Since the domain language used by the model state and the Observer are uniform, the Resolution Rules can be *acquired* by the model space by storing experience and generalizing on them (with user interaction and guidance).

FOCUS PROBLEM: AN ASPECT OF COMMUNICATION

The interpreter gets its input from the Observer process. The data that the Observer uses are state differences computed in terms of snapshots of the world. The Observer does not (and should not) transmit complete snapshots or state differences but attempts 'to describe' by relational triples the changes that occur. For Believer, the observer process is simulated by hand-coding descriptions of actions that have taken place in the episode. The extent and the nature of the summarization that can be done in this encoding depends on the assumptions one is willing to make (or on what one knows) about the Frame rules possessed by the Model space. When the collection of Frame rules in question is meager there is a need to submit more complete collections of the differences observed in the snapshots. The problem for the Observer process, namely, of describing the relevant changes and invariances and also at times, grouping together in a single description a collection of changes, we call the *FOCUS PROBLEM*. This manifests itself as the Segmentation Problem in speech understanding systems where the system observer process has to determine the 'unit' groups of utterances. Soloway (Soloway and Riseman 1975) uses focusing rules in the observer for baseball actions to parse the sequence of state changes to determine the start and end of actions.

There is a complementary relationship between the Frame Problem and the Focus Problem. Whereas the Focus problem concerns the attribution of a single action with a single primary outcome to summarize a large collection of relations

in the state of the world, the Frame Problem concerns the computation of all contingent changes to the model state that must be effected when only the primary change is given as input. One can conceive of the mappings between propositions effected by the recognition rules and focusing rules being inverses of each other [Figure 2]. The implication of this complementary relation for the design of a system that includes an Observer process is the following: The presence of each Frame rule could be rephrased into a Focus rule for the Observer, and to the extent the Frame and Focus rules are derived from the same domain knowledge the communication between the Observer and Model space can be smooth. The design of the knowledge representation language we have permits us to experiment with this aspect of communication among processes.

CONCLUSION

There is an implementation under development in an AI language called FUZZY (LeFaivre 1974) wherein the Recognition and Resolution rules are currently operational. The practicality of these ideas will receive their test in the application area of BELIEVER.

ACKNOWLEDGEMENT

Discussions with Professors Saul Amarel, Casimir Kulikowski, Charles Schmidt, Elliot Soloway and Naftaly Minsky have helped me attain the proper emphasis in this paper. My students John Goodson and Frank Hawrusik participated in a Seminar on Representation of Knowledge and they have helped me considerably. All significant ideas in the discussion of modelling exhibited in the paper are due to Professor C. V. Srinivasan. The specific analysis presented above is an outgrowth of attempting to apply his modelling ideas to the Believer problem of interpreting actions. I thank Marti DeWispelaere for help in typing this paper. This research was supported by the NIH Rutgers Research Resource on Computers in Biomedicine.

REFERENCES

Amarel, S. (1974) Computer-based modelling and interpretation in medicine, psychology; the Rutgers Research Resource. *Proceedings of a Conference on the Computer as a Research Tool in the Life Sciences*, Aspen, Colorado, June 1974. Available as Report RUCBM-TR-29, Rutgers University, June 1974.

Carbonell, J.R. & A.M. Collins (1973) Natural semantics in artificial intelligence. *Proceedings of the Third International Joint Conference on Artificial Intelligence*

(3IJCAI), Stanford.

Fikes, R.E., P.E. Hart & N. Nilsson (1973) Some new directions in robot problem solving, in *Machine Intelligence 7* (eds. B.Meltzer & D.Michie), Edinburgh: Edinburgh University Press.

Hayes, P.F. (1973) The frame problem and related problems in artificial intelligence, in *Artificial and Human Thinking* (eds.A. Elithorn & D. Jones) New York Elsevier Scientific.

Hewitt, C. (1971) Procedural embedding of knowledge in planner. *Proceedings of 2nd IJCAI*, London.

LeFaivre, R.A. (1974a,b) Fuzzy problem-solving, Ph.D. dissertation, Computer Science Department, University of Wisconsin, Madison. The representation of fuzzy knowledge, DCS-TR-33, Rutgers University.

McCarthy, J. & P.J. Hayes (1969) Some philosophical problems from the standpoint of artificial intelligence, in *Machine Intelligence 4*, (eds. B.Meltzer & D. Michie) Edinburgh: Edinburgh University Press.

Raphael, B. (1971) The frame problem in problem-solving systems, in *Artificial Intelligence and Heuristic Programming* (eds. N.V. Findler & B. Meltzer), New York: American Elsevier.

Sacerdoti, E.A. (1975) Non-linear nature of plans, in *Advance Papers of the 4th International Joint Conference on Artificial Intelligence* (Tblisi, USSR), A.I. Lab Publications, M.I.T.

Sandewall, E. (1971) Formal methods in the design of question-answering system, in *Journal of Artificial Intelligence 2.*

Sandewall, E. (1973) An approch to the frame problem and its implementation, in *Machine Intelligence 7* (eds. B.Meltzer & D. Michie) Edinburgh: Edinburgh University Press.

Schmidt, C.F. (1975) Understanding human action, in *Cognition & Social Behavior* (eds. J. Carrol & J. Payne) Lawrence Earlbaum, (In Press 1976).

Schmidt, C.F. and N.S. Sridharan (1976) The representation of plans: Rules of consistency for plan recognition, This volume.

Soloway, E. and E. Riseman (1975) Common-sense theory formation: Towards understanding baseball, COINS-75C-5, Computer and Information Science, University of Massachusetts, Amherst.

Sridharan, N.S. (1971) An application of artificial intelligence to discover syntheses for complex organic molecules, Ph.D. dissertation, SUNY at Stonybrook, (Available from University Microfilms, Ann Arbor, Michigan).

Sridharan, N.S. (1975) The architecture of BELIEVER- Part I, RUCBM-TR-46, Computer Science Department, Rutgers University.

Sridharan, N.S. (1976) The architecture of BELIEVER - Part II. The frame problem in AI. RUCBM-TR-47, Rutgers University.

Srinivasan, C.V. (1973) The architecture of a coherent information system: A general problem solving system, in 3IJCAI, Stanford. A revised version of this appears in *IEEE Transactions on Computers, Special Issue on Artificial Intelligence, April 1976.*

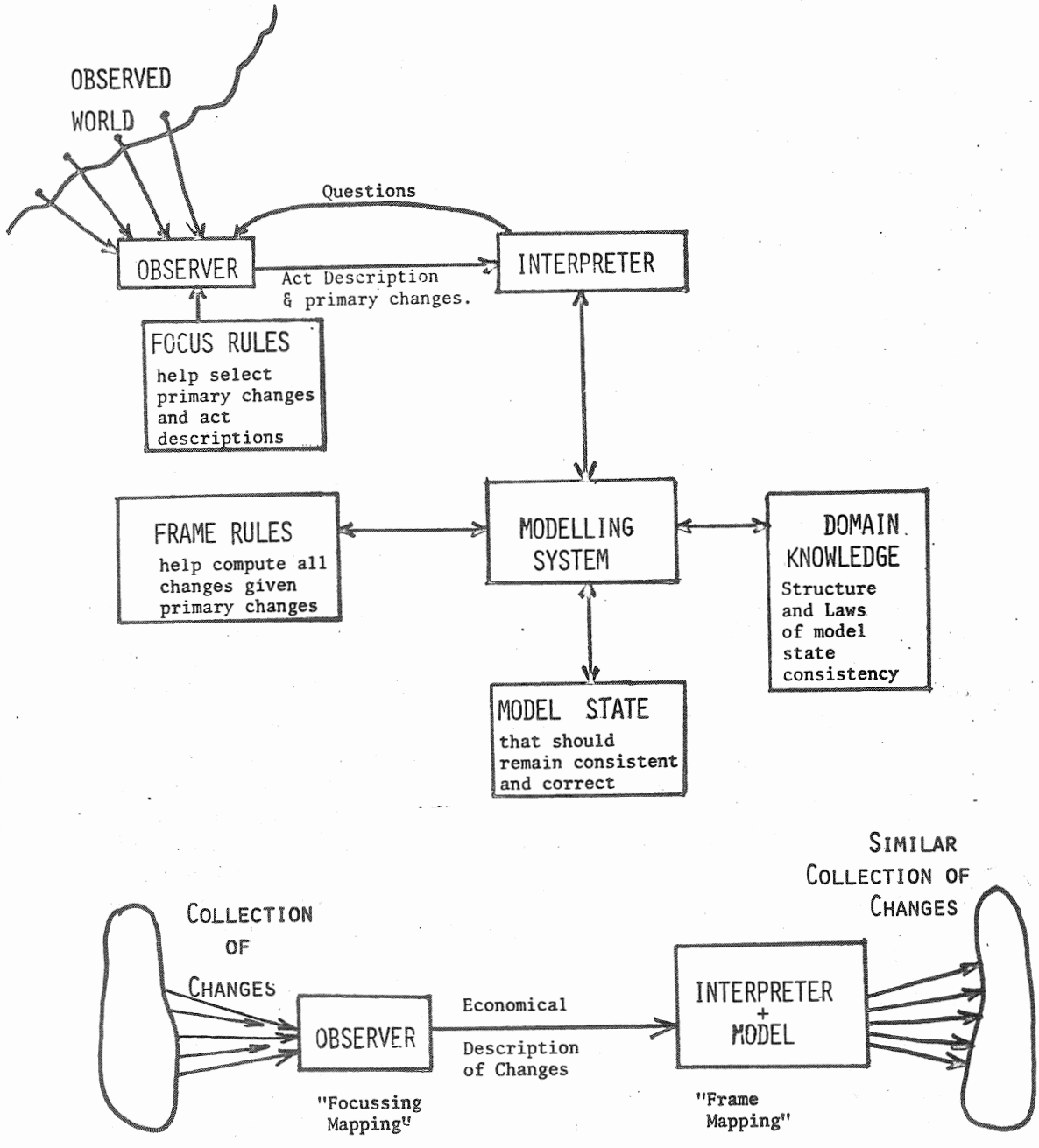


FIGURE 2. FOCUS-FRAME RELATIONSHIP

M. Trigoboff

PROPAGATION OF INFORMATION IN A SEMANTIC NET

ABSTRACT

In creating a computer system which can function as an ophthalmological consultant, particular attention must be paid to the choice of a representation for the system's medical knowledge. This choice affects both the system's efficiency, and its ability to explain itself by reference to this knowledge. We describe a representation consisting of a semantic net in which a set of rules can be associated with any link. The system uses this representation to perform in a bottom-up, event-driven manner. Information propagates through the semantic net under control of the rules. We show how both efficiency and explanation are served by this representation.

1. INTRODUCTION

Our goal is the creation of a computer system which can function as an expert ophthalmological consultant. Such a system must necessarily contain a large quantity of medical knowledge. The choice of a representation for this knowledge has effects on many important aspects of the system's operation.

To be useful, the system must operate efficiently. The knowledge representation we choose must therefore be directly usable by the system's decision processes. The system must also be able to explain its decisions to the user. It can do this only by reference to the medical knowledge on which the decisions were based. This knowledge must therefore be stored in a format which will be comprehensible to the user when presented as part of an explanation.

2. KNOWLEDGE REPRESENTATION

There are two types of knowledge that must be represented in a medical artificial intelligence system. The first is general knowledge of diseases, disease mechanisms, syndromes, pathophysiological states, treatments, and their interrelationships. From these primary elements we can develop a schematic description or model of the disease process. It will be static in the sense that it does not incorporate knowledge of any particular diagnostic or therapeutic situation. The second kind of knowledge is specific to the current context of a given patient. It will change continually as new facts about

a patient are obtained and actions (treatments) performed.

In our system, the first type of knowledge is represented as a semantic net. This is a directed graph in which the *nodes* represent medical states (i.e. diseases, syndromes, symptoms, etc.). The *links* (or *relations*) which connect these nodes represent relationships between them. Links are associated in pairs. Each member of a pair is its partner's relational inverse. Whenever two nodes are connected by a member of such a pair, they are connected in the reverse direction by the pair's other member. People can easily read and understand such a representation, and computers can readily manipulate and process graph structures. We thus have a representation that satisfies both of the above criteria.

The second type of knowledge is represented as ISPECs. ISPEC is an acronym derived from Information SPECification. Each node in the semantic net can have a list of ISPECs associated with it (called its ISPECS property). This list is the system's only information about the status of the node with respect to the patient. Each ISPEC contains among other things a measure of belief, a time notation, a side notation (right, left, or NIL), modifiers which can serve to further specify the concept represented by the node, and complete information detailing how and why the ISPEC was generated. At any point, the system's total knowledge of the patient is contained in the ISPECs.

In the following section, we will describe how the system creates and uses ISPECs. While the syntax of this process is uniform throughout the semantic net, the semantic interpretation of an ISPEC varies depending on its node and on the task currently being performed. For example, the creation of an ISPEC for a node representing a disease can be thought of as hypothesis selection, while the creation of an ISPEC with a time attribute of "future" for a symptom node can be seen as prognosis.

3. PROPAGATION OF INFORMATION THROUGH A SEMANTIC NETWORK

So far, what we have described amounts to a computerized checklist. It merely records the facts we have about our patient. Often, however, when we make a finding we can draw some other conclusions based on that finding. In our system, this capability is implemented as a process called propagation. When an ISPEC is generated for a node, the fact of that ISPEC's creation is propagated along all the links emanating from its node. As a result, ISPECs may be generated for the nodes propagated to, at which point the propagation process continues recursively. Each new finding is like a droplet which, when added to our pool of knowledge, causes a ripple

which can propagate any distance through the pool.

Information contained in the semantic net is used to guide the propagation of suspicion and belief through the node space. Each relation in the net will cause a distinctive pattern of propagation, both because different relation types have different patterns, and because a particular relation type can lead to different patterns depending on its surrounding context.

In order to implement a propagation mechanism, we need a means of associating various types of information with individual links. We have implemented our semantic net using BBN's SEMNET package (Brown, et al., 1974). *Augment nodes*, which are part of this package, provide a way of doing this. An augment node can be associated with any individual link, and can carry whatever information we desire.

Propagation across a link is controlled by the set of rules associated with the link via its augment node. Each rule consists of conditions (RCONDS) which must be satisfied for it to be considered true, and specifications for the creation of ISPECs (RISPECS) in case it is true. RCONDS are requirements that a particular node be true of the patient in the way specified (e.g. PAST, SEVERE, etc). The satisfaction of an RCOND is determined by examining its node's ISPECs. When propagation takes place, the rules associated with the link in question are evaluated. The RISPECS of those that are true specify creation of the ISPECs which are the result of propagation across the link.

Within the set of rules associated with a link, there can be both enabling and superseding relationships. A particular rule may not be directly associated with an augment, but rather may have an enabling link with one that is directly associated with the augment. When propagation across a link is attempted, all the rules directly associated with the augment are evaluated. A rule that is not directly associated with the augment will only be evaluated if a rule that enables it is found to be true.

This facility permits us to simplify the rules considerably. If one particular condition is necessary to a whole class of decisions, it can be embodied in a rule which enables the rules that actually make the decisions. Without enabling, each rule would have had to contain the condition, and the condition would have been evaluated again for each rule. With the enabling capability, the condition need only be specified once, and will only be evaluated once when propagation is attempted.

A rule may also have a "supersedes" relationship to another

rule. When a rule is true, none of the rules it supersedes will produce ISPECs or enable rules even if the superseded rule is itself true. One use of this is to permit enabled rules to turn off the rule that enabled them.

We will illustrate these mechanisms with the example in Figure 1.

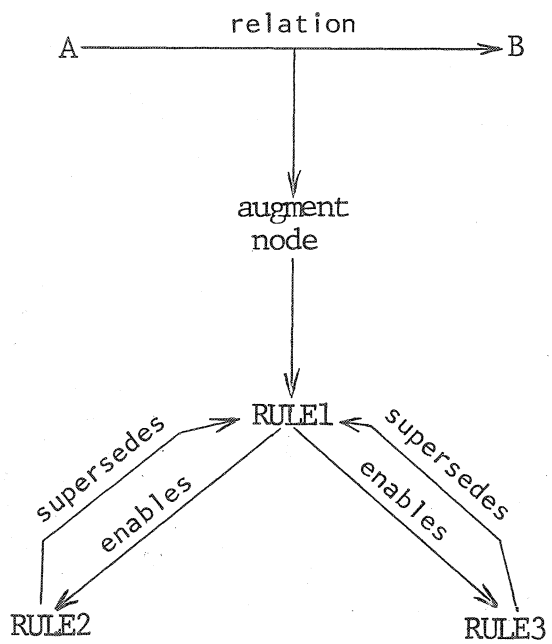


Figure 1

RULE1 requires merely that node A be true of the patient. RULE2 requires in addition that A be true with modifier M2. RULE3 requires A to be true with modifier M3. If A is true but without modifiers, B will get an ISPEC specified by RULE1. If A is true with modifier M2, B will get an ISPEC specified by RULE2 (RULE1 is still true, but has been superseded by RULE2). If A is true with modifier M3, B will get an ISPEC specified by RULE3. Finally, if A is true with modifiers M2 and M3, B will get ISPECs from both RULE2 and RULE3. We mention in passing that any level of enabling is permitted. RULE2 could enable other rules, they in turn could do the same,

and so on.

Thus, to summarize the propagation process: whenever a new ISPEC is created, propagation is attempted across all links with augments that emanate from the new ISPEC's node. For each such link, the rules associated with its augment are evaluated, and the ISPECs of the true, not-superseded ones are added to the ISPECs list for the node at the far end of the link. The process then repeats recursively, potentially producing a wave of propagation across the semantic net.

The propagation process as described above could easily result in a combinatorial explosion. If, on the average, creation of an ISPEC resulted in the propagation of more than one ISPEC, an uncontrollable chain reaction of ISPEC propagation would result. We avoid this problem in two ways. To begin with, once a node has been propagated from, no propagation to it will be allowed from nodes which were reached deeper in the recursion. This keeps the propagation wave from doubling back on itself, just as the refractory period does for neural impulses. Secondly, as was mentioned above, each ISPEC contains a measure of belief. This is a number which can range from -1 to 1 with 1 representing total belief, and -1 representing total disbelief. Most RCONDS require a measure of belief of at least .7 in order to be satisfied. When propagation takes place across a link, quite commonly the measure of belief for the new ISPEC is computed as the measure of belief of the old one times a constant associated with the link. These constants range from 0 to 1. Therefore propagation from a particular node will generally die out within two or three links when the propagated measure of belief falls below the threshold for RCOND satisfaction.

When propagation across a link takes place, two lists are created. One, already mentioned, is the list of ISPECs produced by true, not-superseded rules. The other is a list consisting of all rules associated with the link (either directly or indirectly) which were either false, superseded, or not enabled. This list, and the notation on each ISPEC of the rule which produced it, make it possible for propagation to be undoable. Suppose, for whatever reason, we decide to delete an ISPEC. The ISPEC is removed from its node's ISPECS list, and we then propagate from the changed node. As propagation takes place across any link out of the changed node, rules which were formerly true may now be false. Each ISPEC on the node being propagated to is checked to see if the rule which produced it is now on the second list. If so, the ISPEC is deleted from its node, and the process repeates recursively. This permits the deletion (or alteration) of an ISPEC to propagate through the semantic net in just the same way as the

creation of a new ISPEC does.

Any arbitrary boolean combination of conditions about any number of nodes can be expressed in a rule's RCONDS. This becomes very important when defining propagation to nodes which represent diseases. In general, a disease node will have many incoming links from nodes whose presence or absence characterize it.

We call such a configuration a propagation *cone*. It is usually visualized with the disease node at the apex, and the other nodes occupying the cone's base. Since the same set of rules will be used no matter which node at the base of the cone we are propagating from, we define just one augment node for all base-to-apex propagation in the cone, and allow it to serve as the augment node of all the base-to-apex links. This can also be done for apex-to-base propagation.

A simple semantic net link connects two nodes. Propagation in each direction (semantic net links are always relational-inverse pairs of directed links) is defined by rules associated with the augment node for that directed link. A cone is a generalization of this basic structure. With cones, there can be a set of nodes rather than a single node on one side of the structure. There are still two directions propagation can follow, and propagation in each direction is still defined by a single augment node.

Consider the semantic net link A REL B (REL could be any of the relations defined). When we add an ISPEC to A, one or more ISPECs may be produced for B. We cannot just add them to B, and consider B altered. A may have had ISPECs before the current one was added, and may have already propagated to B. If the new ISPEC does not alter the truth status of any of the rules on the augment for A REL B, a set of ISPECs for B will be produced which is identical to the set produced the last time propagation from A to B took place.

Each ISPEC contains a side notation, the identifier of the rule which produced it, the identifier of the augment node from which the rule was accessed, and a list of the nodes which satisfied the rule. Whenever a set of ISPECs is produced for a node by the propagation process, we check each one to see if it is 1) identical to one already on the node, 2) a replacement for one already on the node, or 3) a totally new one that needs to be added to the ISPECs already there.

In the first case we do nothing. The node is not marked as changed, and propagation will not take place from it. The second case is identified when, for an ISPEC just produced by the propagation process, there is one already on the node

which has the same side notation, and was produced by the same rule from the same augment. In this case we delete the old ISPEC, and the new one, mark the node as changed, and propagate from it. If neither of the above is true, then we add the new ISPEC, mark the node as changed, and propagate from it.

As was mentioned earlier, two lists are produced when propagating across a link. The first is the list of ISPECs produced for the node being propagated to. The second is a list of all the rules which were false, not enabled,or superseded when propagation was being done. This list is used to delete ISPECs from the target node which depend on rules that are no longer true. When a node is being propagated to, each of its ISPECS is checked to determine if it should be deleted from the node. The tests are: its side notation should agree with the side we are currently propagating with respect to, its augment identifier should be identical to the augment we are currently using for propagation, and its rule identifier should appear on the second list.

These mechanisms make the propagation process completely and flexibly undoable. We can add items and then delete selected ones in any order we desire, totally independent of the order in which they were entered. The system can draw conclusions as it goes along, and change its mind as new information comes in. This all gets taken care of automatically: the system's user need only be concerned with coding decision criteria into rules.

One of the criteria on which we base the measure of belief assigned to a disease node is how well it explains, or covers the observed data. Our system deals with this issue by propagating downwards from disease nodes that have a significant measure of belief. The ISPEC that is propagated downwards contains the information that this symptom is explained by the disease node being propagated from. In this way, the propagation process can mark all the symptom nodes that a particular disease node is currently capable of explaining. When a global decision is being made as to which disease node will in fact be chosen as our diagnosis, this information is used. Note that this is implemented as a set of rules on the downwards augment node of the disease node's propagation cone. It is handled by the propagation process in just the same way as any other augment node with associated rules.

4. RELATED WORK

Of all the work in medical artificial intelligence, our system most closely resembles Shortliffe's MYCIN (Shortliffe 1974). MYCIN is a rule-based system whose expertise is in the area of infectious diseases. The rules are of the form:

"If C1 and C2 and ... and Cn then we can conclude D with certainty E." When MYCIN needs to find out whether a particular medical state is true of the patient, it uses those rules which can confirm or deny the state. These rules are accessed from the state via the state's UPDATED-BY property. This is a list of all the rules in the system which can affect the truth-status of the state (in other words, all the rules in which the state in question appears on the right side). This process repeats recursively, terminating at states which are either observable (the system can ask the user if they are true of the patient), already assigned a measure of belief-disbelief, or which have no UPDATED-BY property. A measure of belief for the state the process started from then percolates back up, based on the set of rules that were found to be true.

In our system, the rules are connected to specific links in the semantic net, and this determines when they are used. We might say that they are suspended in a matrix consisting of the semantic net. MYCIN's rules, on the other hand, are suspended in a structure composed of UPDATED-BY pointers. The recursive return part of MYCIN's processing, as described above, is quite similar to the propagation process in our system.

Semantic nets have been widely used in research in artificial intelligence and related areas. Quillian (Quillian 1968), and Rumelhart (Rumelhart et al., 1972) have used semantic nets to model the processes by which humans organize their semantic information. Since we are concerned only with the performance of the given task (medical reasoning), and not at all with modeling a human process, we tend to use our semantic net in very different ways. Brown, Bell, and Burton (Brown et al., 1974) use a semantic net in their system, SOPHIE. SOPHIE's semantic net represents the structure and state-of repair of an electronic device. This is much closer to the way we use our semantic net.

Finally, our work is largely based on, and has grown out of the work of Kulikowski (Kulikowski et al., 1972) and Weiss (Weiss, 1974). It was through experience gained in the use of their system that the need for a general semantic net (and many other features as well) in the current system was first discerned.

5. CONCLUSION

Our system reasons in a predominantly event-driven, bottom-up manner. We have chosen this mode of action because of its wide range or applicability. Any type of reasoning in which some entity's presence or absence can be determined from the

presence or absence of other entities can be modeled using the facilities provided by the propagation process.

The data structure used by our system is a semantic net in which a set of rules can be associated with any link. Such a structure can be readily used by processes whose task it is to explain the system's decisions to the user. If asked "How did you conclude X?," the system can scan X's ISPECs to see which nodes they propagated from, and then say something like: "Because we know U, V, and W, and U CAUSES X, and V and W are ASSOCIATED-WITH X." As we have already shown, the propagation process permits us to use this data structure as the basis for competent performance of medical tasks such as diagnosis. Our system thus fulfills both of the criteria mentioned at the beginning of this paper. Its medical knowledge is stored in a format which can be described in terms familiar to the physician, but nevertheless can be used by the system in performing the tasks of diagnosis and therapy selection.

Currently, the propagation process has been implemented in INTERLISP. Our semantic net consists of approximately 100 nodes, with propagation governed by approximately 200 rules. This covers most of the knowledge needed to diagnose glaucoma. We are moving ahead in the design and implementation of higher level components which will make use of the structure of the semantic net and the results of the propagation process.

REFERENCES

Brown,J.S., A.G.Bell & R.Burton (1974) *Sophisticated Instructional Environment for Teaching Electronic Troubleshooting.* Lowry AFB, Colorado: Technical Training Division Report AFHRL-TR-74-77.

Kulikowski,C.A. & S.M.Weiss (1972) Strategies of data base utilization in sequential pattern recognition, in *Proc. IEEE Conference on Decision and Control and 11th Symposium on Adaptive Processes.* 103-105. New York: IEEE.

Quillian,M.R. (1968) Semantic memory, in *Semantic Information Processing* (ed M.Minsky), 227-270. Cambridge, Mass: The MIT Press.

Rumelhart,D.E., P.H.Lindsay & D.A.Norman (1972) A process model for long-term memory, in *Organization of Memory* (eds E.Tulving & W.Donaldson), 197-246. New York: Academic Press.

Shortliffe,E.H. (1974) *MYCIN - A Rule-Based Computer Program for Advising Physicians on Antimicrobial Therapy Selection.* Stanford: Artificial Intelligence Laboratory Memo AIM-251.

Weiss,S.M., (1974) *A System for Model-Based Computer Aided Diagnosis and Therapy*. New Brunswick, N.J.: Rutgers University Department of Computer Science Report CBM-TR-27.

Woods,W.A., (1975) "What's in a link, in *Representation and Understanding* (eds D.Bobrow & A.Collins), 35-82. New York: Academic Press.

David H. D. Warren

GENERATING CONDITIONAL PLANS AND PROGRAMS

This paper describes Warplan-C, a general purpose system for synthesising plans and programs from precise problem descriptions. The system combines an adequate solution to the "side effect" problem (typified by the task of "interchanging the contents of two machine registers") with the case analysis necessary for generating programs containing conditionals.

Predicate logic is used both as the problem specification language and, in the form of Prolog, as the implementation language. The entire Warplan-C program comprises 66 clauses and compiles into quite an efficient program by current standards.

Keywords:

Planning, automatic programming, side effects, conditionals, predicate logic, Prolog.

1. INTRODUCTION

This paper describes Warplan-C, a general purpose system which can automatically synthesise small conditional plans or programs to solve precisely specified problems.

The notions of "planning" and "programming" are generally used rather loosely. In this paper we propose to use the terms in a more exact and limited sense. The characteristic feature of "planning" is the problem of "side effects". We say that a side effect is associated with an event E if there is some statement P which is true before E but is not necessarily true after E. For example, if P is the statement "Ian has a pound note in his pocket" and E is the event of "Ian buying an ice cream".

Side effects are an important consideration in a programming language such as Fortran, where the execution of a statement typically means that the value of some variable is not what it was. In contrast, pure Lisp is a language without side effects - a statement that some Lisp function applied to certain arguments yields a certain result is true universally if it is true at all.

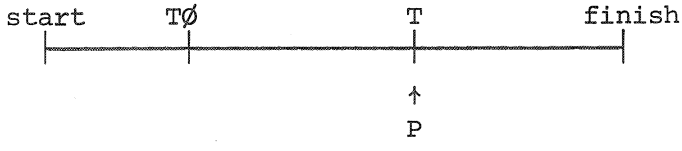
"Planning" is therefore the design of a sequence of actions (or events) which results in a certain statement being true, where it is a fundamental property of the actions that they may cause side effects. Thus we would not regard the problem of

constructions in geometry as "planning" since, in the ideal mathematical world at least, there are no side effects involved in effecting a geometrical construction. In so far as we distinguish "programming" from "planning", we shall use "programming" to mean the production of plans containing conditional and/or iterative steps.

Warplan-C is implemented in Prolog (Battani and Meloni [1973], Roussel [1975]), a practical embodiment of the principle of programming in predicate logic (Kowalski [1975], van Emden [1975]). The system is an extension to handle conditionals of an earlier planning program, Warplan (Warren [1974]). The simplicity of the extension is a good example of the modularity advantages to be gained from programming in logic.

2. BASIC METHOD

The approach to the side effect problem is as described by Warren [1974], and is very similar to that of Manna and Waldinger [1974] (see also Waldinger [1975]). Briefly, the plan is constructed by progressively modifying a partial solution until there are no unsatisfied goals. For example consider the partial plan diagrammed as:-

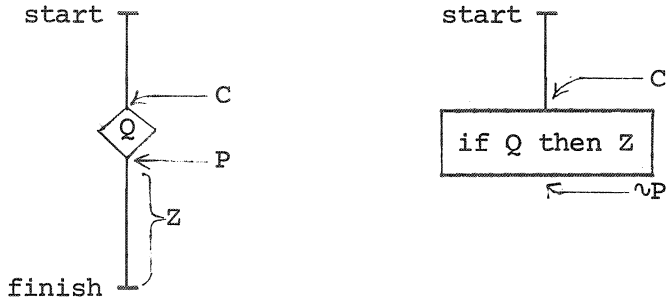


where the goal P needs to be true at time T. Nothing need be done if P is already true at T, i.e. P has been made true by some earlier action and is not affected by any intervening action. Alternatively a new action which achieves goal P must be inserted in the plan somewhere before T, say at TØ. The system makes sure that P is not affected by any action between TØ and T, and also that the new action doesn't itself affect any of the already achieved goals. In general the new action will require that further goals be established at TØ. The system effectively attaches "comments" to its partial solutions so that it can "protect" already achieved goals (cf. Sussman [1973]).

The main difference between the above and the method of Manna and Waldinger is that here it is permissible to insert an action to achieve P even if P is already true in the partial plan as it stands. This allows goals to be considered in only one order, which may be chosen arbitrarily. Because Manna and Waldinger do not allow this, they have to introduce a rule to permute the order of considering goals (as STRIPS, Fikes and Nilsson [1971]). In the absence of a satisfactory way of coping with the enormous redundancy this introduces, we consider it is better to accept the somewhat unintuitive feature

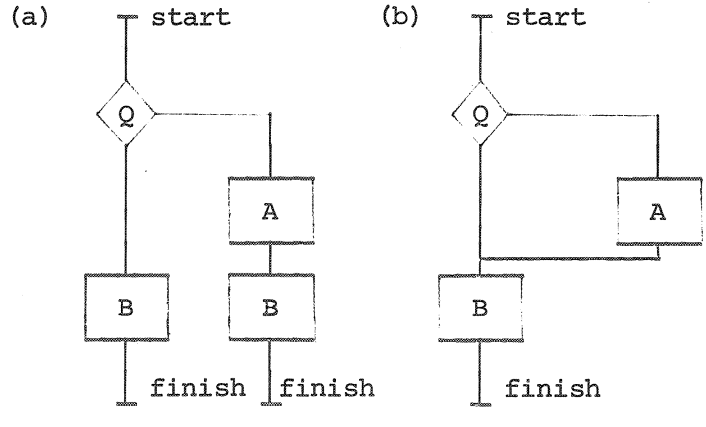
mentioned.

We have found that conditional planning can be integrated into our scheme for handling side effects in a simple but effective manner. Certain actions are specified to the system as conditional, with two possible outcomes labelled P and ∿P, where P is some statement. Initially, such actions are treated just like unconditional actions which achieve the goal P. When a "solution" has ostensibly been found, the system traces back through the plan to see if there are any other cases it must consider. If a conditional action is encountered, the plan is rewritten as indicated:-



The test "if Q then Z" is now treated like an unconditional action which achieves ∿P and which has C as preconditions, where C are the goals protected immediately before Q in the original plan. Planning starts afresh with the new plan as "initial state", trying to achieve the original overall goal. Note that extra actions may get inserted before the test if needed for the "else" branch now being generated.

This process is iterated until a complete plan has been generated with no unresolved cases. Note that different branches never "rejoin" in the plans thus constructed. i.e. The plans are like (a) rather than (b)below:-



It is difficult to generate plans of the second type without redundancy. We feel they are best arrived at as a subsequent

optimisation or, better, by breaking the main task into smaller subtasks in a structured programming approach.

To facilitate the specification of problems to the system, particularly those requiring conditional solutions, the system permits definitions of relations in the domain in terms of other relations in the domain.

A number of devices are incorporated into the system to restrict its search and to direct it more quickly to a solution:-

(1) Consistency

The system can be given information that certain statements can never be true simultaneously, and use it to reject inconsistent goals. This feature plays an especially important role in conditional planning, as it guides the system to generate different solutions to the overall goal as the "then" and "else" branches of a conditional.

(2) Minimality

The system ensures that a condition is never tested twice on the same path through a program.

(3) "Loop" Checks

These optional checks are ad hoc, in that they destroy the completeness of the system. i.e. They may well prevent a bona fide solution from being found. However, in general they have a strikingly beneficial effect. The "weak" "loop" check rejects a goal if it is a subgoal of an identical goal at the same point in the plan. The "strong" "loop" check rejects the subgoal if it is merely unifiable with the supergoal.

3. SAMPLE DOMAIN AND PROBLEM

The use and performance of the system is illustrated below on the problem of synthesising a simple machine code program. As a domain, machine code programming is of interest for its prototypical side effects and the wealth of practical examples.

Problem domains and individual problems are described to the system in statements of predicate logic. We would advocate this irrespective of the choice of implementation language for the planning system.

The Prolog syntax used below is that accepted by the author's compiler. Variables are distinguished by an initial capital letter. '_' is used for an "unnamed" variable (following Pavel Brazdil). Clauses are written in the form 'P:-Q,R,S.' meaning "P if (Q and R and S).". Prolog's "slash" ('/') determinator is written '!'. Infix notation is used freely for predicates as well as functors.

The machine described has an accumulator 'acc' and a number

of registers 'reg R' where R is a machine address. Space only permits us to show the descriptions of the machine instructions actually needed for the problem. These are:-

LOAD R	: to load a value from register R into the accumulator;
STORE R	: to store a value from the accumulator into register R;
LOADI R	: to load indirectly a value from the register whose address is in register R;
ADDC C	: to add a constant C to the value in the accumulator;
JUMPLT R	: to test whether the contents of the accumulator is less than the contents of register R.

The description uses terms such as 'loadi(R,Rl,X)' to name the event of executing the instruction "LOADI R" where register R contains Rl and register Rl contains X. The "extra" arguments can be thought of as "comments" which enable the system to remember how the plan works and to "protect" solved goals. 'L:X' means "location L contains value X". The prefix '#' is used for program constants (addresses known at load-time).

.1 Principal Effects of the Instructions

```
acc:(X+C) after addc(C,X) :- const(C).
acc:X after load(#R,X).
acc:X after loadi(#R,Rl,X).
reg#R : X after store(#R,X).

X<Y aftertest jumplt(#R,X,Y).
```

'P after U' means "statement P is true after the event U". 'P aftertest U' means "statement P is true if the test U is successful".

.2 Instruction Preconditions

```
addc(C,X) needs acc:X.
load(R,X) needs reg R:X.
loadi(R,Rl,X) needs reg Rl:X & reg R:Rl.
store(R,X) needs acc:X.
jumplt(R,X,Y) needs acc:X & reg R:Y.
```

'U needs C' means "event U can only take place if condition C holds".

.3 Side Effects etc.

```
U affects R:X :- R:Xl after U.
                                    always X=/=X :- !,fail.
never ~X & X.                       always X=/=Y.
never R:X & R:Y & X=/=Y.
never X<Y & Y<X.                    const(Ø).   const(1).
```

'U affects P' means "it should not be assumed that statement P holds after event U merely because it held before". [nb. There is no presumption that P is false after U nor that P is true before U. In this domain, the single 'affects' statement above suffices to describe all possible side effects.]

'never C' means "statement(s) C can never be true simultaneously.

'always P' means "statement P is universally true".

.4 A Problem

The example problem is that of loading into the accumulator the smallest value from a vector of registers. In fact, the system is just given the major subtask of generating the body of a "loop" to iterate through the vector and replace the value in the accumulator by a value from the vector if the latter is smaller. The present system is not able to design iterative programs unaided.

The addresses of the start and end of the vector are given in registers #a1 and #a2. We make use of the definitions of two relations:- 'min(X,L,Y)' meaning "Y is the minimum of X and the elements of the list L"; 'lesser(X,Y,Z)' meaning "Z is the lesser of X and Y". These are defined just as one might write them for an ordinary Prolog program except that (a) the predicate 'if' is used to make the rules explicit and (b) the functors used are the list destructors 'hd' and 'tl' instead of the usual constructor.

```
min(X,L,Y) if lesser(X,hd(L),X1) & min(X1,tl(L),Y).
min(X,L,X) if null(L).

lesser(X,Y,X) if X<Y.
lesser(X,Y,Y) if ~X<Y.

reg R1:hd(L) if R1 to R2:L & R1<R2.
R1+1 to R2:tl(L) if R1 to R2:L & R1<R2.
null(L) if R1 to R2:L & ~R1<R2.

initially acc:n.
initially start to end:list.
initially reg#a1:start.
initially reg#a2:end.

want min(n,list,X) & acc:X.
```

'R1 to R2:L' means "L is the list of values stored between addresses R1 and R2".

'initially P' means "statement P is true in the initial state 'start'".

'want C' means "statement(s) C are the goal(s) which must be true in any final state".

.5 Induction Hypothesis

In order for the system to generate a "loop", the user has to supply an induction hypothesis in the form of a description of the routine being synthesised ('minrtn') for recursive use by the planner. In this case, the information is a simple re-write of the 'initially' and 'want' statements above, plus a "safe" assumption about the routine's side effects.

```
acc:min(X,L) after minrtn(X,L,R1,R2).
minrtn(X,L,R1,R2) needs
     acc:X & R1 to R2:L & reg#a1:R1 & reg#a2:R2.
minrtn(X,L,R1,R2) affects reg#R:X1.
always min(X,L,min(X,L)) :- L=tl(list).
```

The last statement is surreptitiously restricted to prevent 'minrtn' being used to solve the overall goal. These statements are placed first for Prolog control purposes.

4. SOLUTION OF THE PROBLEM

The first solution generated by Warplan-C, given the control information implicit in the clause and goal orderings above, and applying the "strong" loop check, is shown in Figure 1. Alongside it is the corresponding assembly language program in a more conventional syntax. The code to be executed following a successful jump is shown indented rather than separated elsewhere. The code generated by Warplan-C makes recursive calls to 'minrtn'; in this case there is an obvious direct iterative equivalent which is shown in the assembly language version.

The numbers to the left of the instructions indicate the order in which Warplan-C has generated them. Note:-

(a) An earlier failure results in the generation of the apparently "unnecessary" STORE and re-LOAD of value 'n' (steps 3 and 2). This would not be possible with the Manna-Waldinger method. Access to 'hd(list)' is now possible (steps 4 to 6) without "losing" 'n'.

(b) Instruction (8) on a "then" branch causes '#a1' to be incremented before the test (1) to avoid destroying 'n' in the accumulator.

(c) The "else" branch (12) takes advantage of the steps (9, 10, 11, 4, 5) already introduced for the "then" branch (8).

This example was chosen to illustrate the full range of the system's capabilities. It is not usual for such a near-optimal solution to be generated at a first attempt. Other

Figure 1

```
(3)  store(#T1,n)
(7)  load(#a1,start)
(6)  jumplt(#a2,start,end)
(5)     loadi(#a1,start,hd(list))
(4)     store(#T2,hd(list))
(11)    load(#a1,start)
(10)    addc(1,start)
(9)     store(#a1,start+1)
(2)     load(#T1,n)
(1)     jumplt(#T2,n,hd(list))
(8)        minrtn(n,tl(list),
                  start+1,end)
           exit
(13)    load(#T2,hd(list))
(12)    minrtn(hd(list),tl(list),
                start+1,end)
        exit
(14) load(#T1,n)
     exit
```

```
MINRTN:
   STORE TEMP1
   LOAD A1
   JUMPLT A2,LAB1
LAB1:  LOADI A1
       STORE TEMP2
       LOAD A1
       ADDC 1
       STORE A1
       LOAD TEMP1
       JUMPLT TEMP2,LAB2
LAB2:     GOTO MINRTN


       LOAD TEMP2
       GOTO MINRTN


   LOAD TEMP1
   EXIT
```

Time: 11.6 sec (compiled Prolog on a DECsystem 1Ø).

Store: 2ØK words (total for program + data areas,
including Prolog run-time system).

examples tested successfully include:-

(1) a machine code routine to compute factorial:

(2) a version of the "sort 3 variables" problem described by Waldinger [1975];

(3) implementation in machine code of the "sort2" routine used in (2) above.

5. EXTRACT FROM THE IMPLEMENTATION

The entire Warplan-C program (excluding user interface) comprises 66 clauses. A verbatim extract is reproduced below showing the main rules used in constructing a plan. 'plans' is the user's handle to start the system running. 'planfrom(T,A)' means "T can be constructively modified to a plan for the overall goal, compatible with assumptions A". 'plan(X,PØ,TØ,P,T,A,G)' means "T is a sequence of actions which achieves goals P under assumptions A and which contains a subplan TØ achieving goals PØ where P is equivalent to X&PØ". [The extra argument G is a goal list used for "loop checking" and is not semantically significant.] As executed by Prolog, X,PØ,TØ,A,G will be input arguments and P,T output.

```
plans:-planfrom(start,_).

planfrom(TØ,A):-
     want C, consistent(C,A),
     plan(C,void,TØ,P,T,A,[]),
     othercases(T,P,A,exit).

plan(X1&X2,PØ,TØ,P,T,A,G):-!,
     plan(X1,PØ,TØ,P1,T1,A,G),
     plan(X2,P1,T1,P, T, A,G).
plan(X,PØ,TØ,PØ,TØ,_,_):-always X.
plan(X,PØ,TØ,P, TØ,_,_):-holds(X,TØ),addtoset(X,PØ,P).
plan(X,_,_,_,_,A,_):-minimalityviolation(X,A),!,fail.
plan(X,_,TØ,_,_,_,G):-loopcheck(Type),
     (Type=strong;Type=weak,mkground(X)),
     memberoflist(X:TØ,G),!,fail.
plan(X,PØ,TØ,X&PØ,T,A,G):-
     X after U, U needs C, consistent(C,A),
     extraassumptions(U,A),
     achieve(X,U,PØ,TØ,T,A,[X:TØ,..G]).
plan(X,PØ,TØ,P,T,A,G):-
     X if C, consistent(C,PØ&A),
     plan(C,PØ,TØ,P,T,A,[X:TØ,..G]).
```

'[H,..T]' is our notation for a list whose head is H and tail is T.

6. DISCUSSION

One of the main advantages of this system is that it is conjectured to be complete. i.e. If a problem can be expressed to the system, the system can in principle find a solution if there is one. This is certainly not the case with "STRIPS-like" systems such as HACKER and (apparently) Luckham and Buchanan's, which cannot, for instance, solve the problem of interchanging the contents of two registers.

The major limitation of Warplan-C is the redundancy inherent in the existence of plans which are identical but for the precise ordering of the actions. This problem is addressed by Sacerdoti [1975], whose system constructs plans which are only partially ordered with the aim of avoiding unnecessary orderings. We are not altogether convinced that this approach is practicable. It is certainly a major problem to find an explicit representation of the set of all valid reorderings of a plan (cf. the difficulties of applying the Vienna Definition Language tothe formal definition of PL/1). Human programmers appear to proceed by relaxing previous "unnecessary" ordering assumptions rather than by only introducing an ordering if it is absolutely necessary. It is a much more difficult task to design a program for a parallel processing machine than for a sequential one.

Most other authors of planning systems have used languages which provide built-in facilities for maintaining one or more models of a "world state" (e.g. Sussman: Conniver, Tate [1975]: HBASE, Luckham and Buchanan: Microplanner). We would argue that such facilities, even the more ambitious "contexts" of Conniver et al., have proved too limited and inflexible for planning applications. In most cases they have prevented an adequate solution to the side effect problem. Even in Tate's system, where the problem has been overcome, the decision to manipulate a sequence of goals seems to have been forced by the inability to manipulate the plan itself. Manna and Waldinger do use the contexts of QLISP, but only in a weak way - all contexts are immediate descendants of a single "global" context. The "skeleton" models that they advocate are effectively implemented by Warplan-C in the form of protected goals attached to each point in a plan. Their "regression" rules are mirrored by the use Warplan-C makes of the 'after', 'needs' and 'affects' statements. We do not feel Prolog needs a "context" facility, but a built-in "set" datatype would be useful, if implemented efficiently.

7. ACKNOWLEDGEMENTS

This work was supported by an SRC research studentship, under the supervision of Prof. Bernard Meltzer, Dr. Robert Kowalski and, formerly, Prof. Donald Michie. It would not have been

possible but for interaction with members of the AI Group, Marseille. The ideas on conditional planning were stimulated by discussions with Gottfried Eder.

8. REFERENCES

Battani, G. and Meloni, H. [1973]. "Interpreteur du Langage de Programmation Prolog", Groupe d'Intelligence Artificielle, Marseille-Luminy, 1973.

van Emden, M.H. [1975]. "Programming with Resolution Logic", Univ. of Waterloo Research Report CS-75-3Ø, Nov. 1975.

Fikes, R.E. and Nilsson, N.J. [1971]. "STRIPS: A New Approach to the Application of Theorem Proving in Problem Solving", Artificial Intelligence, Vol. 2, No. 3/4, Winter 1971.

Kowalski, R.A. [1974]. "Logic for Problem Solving", Dept. of Computational Logic Memo 75, Edinburgh, March 1974.

Luckham, D.C. and Buchanan, J.R. [1974]. "Automatic Generation of Programs containing Conditional Statements", AISB 1974 Conference, March 1974.

Manna, Z. and Waldinger, R.J. [1974]. "Knowledge and Reasoning in Program Synthesis", AI Center Technical Note 98, SRI, Nov. 1974.

Roussel, P. [1975]. "Prolog: Manuel de Reference et d'Utilisation", Groupe d'Intelligence Artificielle, Marseille-Luminy, Sept. 1975.

Sacerdoti, E.D. [1975]. "The Non-linear Nature of Plans", AI Center Technical Note 1Ø1, SRI, Jan. 1975.

Sussman, G.J. [1973]. "A Computational Model of Skill Acquisition", Ph.D. Thesis, MIT, August 1973.

Tate, B.A. [1975]. "INTERPLAN: A Plan Generation System Which Can Deal With Interactions Between Goals", MIRU Memo MIP-R-1Ø9, Edinburgh, December 1974.

Waldinger, R.J. [1975]. "Achieving Several Goals Simultaneously", AI Center Technical Note 1Ø7, SRI, July 1975.

Warren, D.H.D. [1974]. "Warplan: A System for Generating Plans", Dept. of Computational Logic Memo 76, Edinburgh, June 1974.

Y. Wilks

DE MINIMIS, OR THE ARCHAEOLOGY OF FRAMES

There is a principle of common law that you cannot be convicted of, say, possessing a forbidden substance unless you possess a certain minimum amount of it. If you possess only a *very* small amount you can cite the principle of De Minimis. In this note I introduce the notion of a *dynamic thesaurus row*. Whether or not that constitutes possession of 'frames' in one's system, in the sense of Minsky (1974), is for the courts of opinion to decide but, if it is found to be one, it can only be an absolutely *minimum frame*.

In a recent paper (Wilks 1975b, sec VII) I argued that it had not yet been made clear what claims were being made in the applications of a *frames hypothesis* to the understanding of natural language. I suggested two things they *might* be claiming, and argued that these had not been shown to be true.

Here I want to take a rather different point of view and to accept that, quite apart of the truth or falsehood of general hypotheses connected with frames, there is an important aspect of word-sense representation concerned with *sequences of events*, and that this aspect has not been taken account of adequately in the primitive-based system of representation I have described elsewhere (for example Wilks 1975a). In this note I want to suggest a way in which this dynamic aspect could be built into a semantic representation system, and in a way closely associated with a traditional form of linguistic information, the *thesaurus*, which is normally thought of as composed of *rows* of semi-synomnyms such as:

hammer, chisel, file, screwdriver, ... wrench

all of which are tools, and the row is unordered. I shall suggest a notion of a *dynamic thesaurus row* which is ordered.

THE DEFECT OF A MERELY STATIC WORD-SENSE REPRESENTATION

A clear demonstration of the need for a non-static element in sense representation comes from very simple considerations. Suppose we wish to express the meaning of an action like 'smoke'. It soon becomes clear that any formalization of 'drawing of a gas from an object into the lungs of a human'

is somehow inadequate. That throws no light at all on the meaning of

Smoking makes your fingernails yellow (1)

Smoking shortens your life span (2)

Smoking is a social disease (3)

Though one might argue that it could help with

Smoking makes your breath smell (4)

One trouble with this inadequacy argument is that it is by no means clear just what help is needed with those sentences, and one might argue that 2. could be dealt with simply by putting in the representation of 'smoke' some formalization of 'causes people to die'. Other examples like

His mother's smoking made him feel unhappy (5)

might be 'understood' so long as we had inference rules that would link representations by some formalization of

Activity X causes people to die → X cause other people to be unhappy (6)

and so on. Nonetheless, even examples as naive as 1 suggest the need for something in the sense representation that could express the *sequence of activities that constitute smoking*, rather than particular one-step sequence inferences like 6. Notice that this need of dynamicism is present in all word-sense representations, whether expressed in semantic primitives or not.

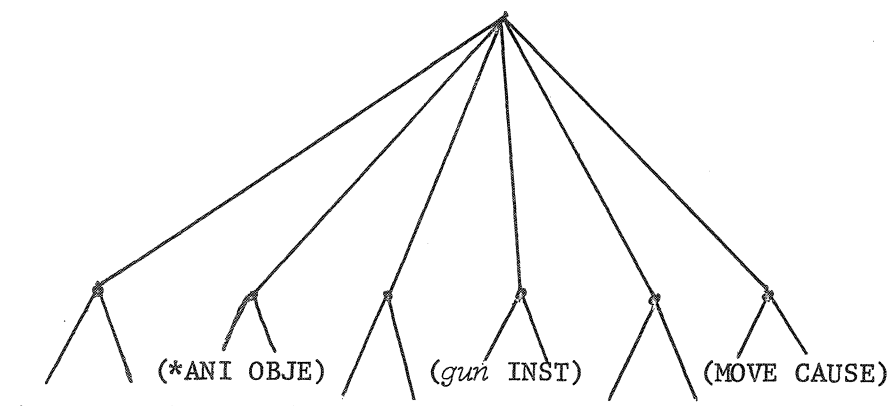
THE DEFECT OF A MERELY PRIMITIVE REPRESENTATION

In Wilks (1975b, sec II) I argued, as many have, that expressing sense representations with the aid of only very general concept names (MAN, THING, CAUSE, MOVE and so on) led to enormous redundancy of notation so that, for example, the inference rules in (Wilks 1975a) are actually expressed in a short-hand like 6 above, because to write in full the functions ranging over primitives that express the same content would take many times the space. After a while one begins to suspect that there may be some reality to the short hand, even if the procedures, based only on the primitives, work well enough in practice.

ONE NAIVE SOLUTION TO THE DEFECTS OF PRIMITIVES

In Wilks (1975a, sec II) I suggested, not wholly seriously, one possible solution, namely that *word names* could also function in representations so long as they themselves had

sense representations elsewhere, i.e. any words so employed were also in the dictionary themselves. This might lead to inserting in the dictionary a sense formula for 'shoot', using primitives plus the word names 'gun' and 'bullet' as follows



(7)

Where the whole thing is to be read as: (syntax details in Wilks 1975a or 1975b). Shooting is a CAUSing to MOVE of an OBJect (a bullet), done by a HUMan SUBJect, with an INSTrument that is a gun and with the GOAL of STRIKing an ANImate object. Here 'gun' and 'bullet' replace a more general THING.

This suggestion was also proof against the charge that the sense formula now contained surface words at random (as Schank's (1973) graphs certainly seem do to), because these inserted words would be merely shorthand for other formulas. So, 'gun' in 7 is simply shorthand for the formula for 'gun' in the dictionary.

The trouble with this was that it made too many concessions to the anti-primitivists: 'gun', as it functions in 7, is already too specific: what we want is something more general, even if *less* general than THING, as I shall now show.

THE DEFECT OF A MERELY SURFACE WORD REPRESENTATION

Any reader who had been puzzled, because it has never occurred to him to represent meaning other than by an unrestricted use of ordinary English words, should be aware of the limitations to that view. A recent suggestion for a frame system (Charniak 1975) writes the first three lines of the frame (for shopping in a supermarket) as follows

a) Goal: SHOPPER owns PURCHASE - ITEMS

b) SHOPPER decide if to use BASKET, if so set up cart-carry FI

c) SHOPPER obtain BASKET. (8)

The question is, *what are* those objects in upper case, words or what? Is BASKET the English word 'basket'? The question is not an empty one because, if that frame is ultimately to be applied to English stories, as Charniak intends, than how BASKET is tied in the process of analysis to whatever word is used in the story, say, one of:

basket, box, carrier-bag, push-cart, plastic-sack..hands (9)

will be crucial. The assumption (not made by Charniak himself) that one will actually be able to *match frame objects like BASKET directly onto the input* because the story teller will always be considerate enough to use 'basket' is too silly to be worth discussing. But then how is a surface-only system like 8 to be applied, even in principle? I am not saying that such a system must have *intensional* sense representations like 7, because *extensional* thesaurus rows of words like 9 might do equally well. But it must have something more than a naked 8.*

A COMPROMISE POSITION LEADING TO AN EXTENSIONAL-INTENSIONAL DUALITY

One way of getting a greater generality than the substitution of words into sense representations, as in 7, would be to use more general concepts than words, though not as general as primitives. So, instead of 'gun' in 7, something at the level of 'shooting instrument' might be better. This sort of entity I shall call a thesaurus head (TH) and write as '+shootinginstrument', to contrast with 'word' and 'PRIMITIVE'. Though, of course, all three forms are words in some sense, and the difference concerns how we *use* the three kinds of entity in a system.

---

*I do not mean to suggest that Charniak has not considered how the frame variable BASKET is bound, for he has: (Charniak 1975) discusses, for example, how it should *not* be bound to 'pocket'. However, it is not clear that consideration of the binding of frame variables is the same thing as considering the linguistic forms of information that will allow the frame to be attached to text. The former activity leaves us inside the frame system, the latter forces us to connect it with something external.

A thesaurus (i.e. Roget) is a traditional classification of words via general concepts; as it were a reverse dictionary. In Roget there are about 1000 TH's* like +goodness and +agriculture. Under each TH are blocks of words, usually corresponding to a particular part of speech, that at the lowest level consist of rows like 9, which might be a row under a TH +container. Presumably, +shootinginstrument would be a low-level subhead, perhaps just the name of a row.

The suggestion of 7, i.e. putting word names in formulas, would have given rise to a recursive form of representation, because a word name in a formula might well have, as its formula, one that itself contained words, etc. One might in fact circle, and never actually reach formulas containing only primitives, but that need not matter so long as from every word you *could always go on* to another formula.

On the present suggestion, the TH's in formulas (i.e. with +shootinginstrument not 'gun') would point off in two ways: (i) to a formula (the *intension* of +shootinginstrument) and (ii) to whatever rows of words were listed under +shooting-instrument in the thesaurus (the *extension*, lexically speaking, of the term). The thesaurus need not even exist explicitly, for it could be constructed when needed - in this case the row consisting of all the words whose formula trees contained the sub-formula that was in the dictionary as the intension of +shootinginstrument. That is to say, the formulas for 'rifle', 'colt45', 'mortar' etc. would all have a common sub-formula which would be the formula for +shoot-inginstrument.

These formulas composed of primitives and TH's would also be recursive like 7. It should also be noted that, unlike TH's, primitives (like MAN) would *not* be the names of parts of a thesaurus: they would be too general - though they might be considered the names of classes of TH's.

We would thus have a dual representation, or rather two forms of organization of the word-sense representations, and this duality would extend to the inference rules, because the form of rules like 6 would now be not only a shorthand form of the full primitive version of the rule but a dual of it, and should now be written

[+activity X +cause-die *HUM]→[*HUM +dislike X](10)

---

*this use of 'head' here has *no connexion* with its use as the leftmost item of a LISP list, or the rightmost item in a formula like 7.

where the primitive *HUM ('*' denoting a class of primitives containing MAN) remains in the rule because, as just noted, such concepts are too general to be abbreviated by TH's. Thus the dual of 10, its primitive-only 'full form', would have *HUM in the same slots as 10.

FRAMES AND THESAURI: REDISCOVERING HISTORY

I have not just plucked the notion of a thesaurus out of the air. It was a popular tool in the computational linguistics of the Fifties (see, for example, Masterman 1956) and, moreover, one designed to do a job very like the one frames are now being suggested for. To put it crudely, the thesaurus using system was designed to tell you what the context was, or, as it is now put, what 'frame you are in'. The basic thesaurus algorithm of twenty years ago was as follows: for every word, list the TH numbers in Roget (between 1 and 1000) under which the word falls. Then set intersect the TH number lists of all the words in a sentence. The set intersection should yield the numbers of the TH's that define the 'context'. Thus, 'buck' in one sense would be under +gameanimal and (as an object) under, say, +hunting, while in the other sense it would be under +money. Then, in (cf Riesbeck 1974)

The hunter went shooting and got 4 bucks (11)

we would get the right sense of 'buck' in this context because 'shoot' would also be under, say, the TH +hunting and so the right sense of 'buck' would have its head in the set intersection of TH's, while the money sense would not. This was an inadequate method, as can be seen from

The hunter went shooting with 4 bucks in his pocket (12)

But the moral is this: those frame implementations which tackle 11 by saying 'shooting' will set up the *Hunting Frame* and so when we encounter 'buck' etc. etc. etc. are rediscover ing the thesaurus and will fall down on 12 in just the same way.

There already exist 'frame papers' that rediscover the algorithm above, though referring not to lists of TH's but to the intersection of one 'frame index' with another.

What is needed to overcome the defects of both the unaided thesaurus and the 'naive frame' so as to tackle examples like 12 is, of course, quite other structural mechanisms, necessarily required for the actual application of knowledge structures to texts. My point here, however, is not simply to show that the traditional role of the thesaurus in computational linguistics had similarities to that now being

claimed for the frame, but to add another twist to the traditional thesaurus so as to make it even more like a frame. I shall then suggest that this augmented thesaurus may actually be useful, provided it is seen only as a part of an understanding system and not as its star turn.

A DYNAMIC THESAURUS

Thesaurus rows like 9 are unordered; except in so far as we might designate say, the first item to be a default value, as in Minsky (1974). Suppose we also had *ordered rows*, and moreover ones that consisted not of words, as 9 does, but of TH's. Suppose we wrote

+extract +hold +light +suck +blow +stub (13)

and we said that the name of that row was the TH +smoke, just as +container was the name of 9, except that it will not be the case in 13, as it was in 9, that the formula for the row name is part of the formula for each and every member of the row.

In some sense, this ordered row captures the 'sequence of events' aspect of the meaning of the action 'smoke' that was missing from the earlier static representation, so and might allow us to 'understand' sentence 1 better.

This additional information 13 may seem a bit thin, but it is recursive (in the same sense as used in 'recursive transition network') in that the TH's composing it can point not only to thesaurus rows, of semi-synonyms, but also to other *dynamic rows*. For example, +light might itself be the name of a dynamic row. Again, it would be a small matter to adjust the notation so as to admit exceptional conditions and side conditions in the way that real dictionaries and thesauri do. So, a more realistic version of 13 might be

+extract(∿OBJE=pipe)+hold +light +suck +blow
(OBJE+diminish)+stub(∿OBJE=pipe) (14)

Here we see that the smoked object is not extracted or stubbed if it is a pipe and that in the sequence the main object diminishes. An interesting point, that might divide those who do and do not believe that such descriptions should *simulate* the activity concerned, would be whether or not the row should contain some expression of 'do sucking and blowing again *while* object is greater than 1 cm'. I do not believe this is an essential part of the linguistic description, but it would require no great feat of gymnastics to incorporate it into this notation.

There is no trouble about the binding of variables such

as 'main object' ('main' can be understood to designate the object of the row-naming action +smoke since it must be remembered that the TH's in the row point not only to word rows (and possibly other 'dynamic' TH rows recursively) but also to their own formulas like 7. It is the associated formula that gives the intensional structure that the system expects to be associated with an action: thus in 7 we see that shooting is expected to have, or prefers to have, a human agent and some sort of animate object. Similarly, for +extract in 14 above: its formula would show its anticipated agent, object, instrument etc. Hence there is no assumption in 14 that +extract and +suck have the same object, for their associated formulas would make clear that they do not.

Thus, it should be clear that 14 is also a form of shorthand*, and could, if and when analysis of text demands it, be expanded out to a fuller structure (a 'maximal frame', if you will) since each TH will point to a formula, which is, in its turn, a blue-print for a small text structure.**

---

*'Compacted form of information, or knowledge' sounds better. Some readers may feel that I have over emphasised the point about 'shorthand' throughout, and that the notion of a word name standing for its own formal definition is well established in the field and needs no emphasis. True, but as I have argued in (Wilks 1975b, section III), it is a convention fraught with danger, in that it is often used when there is nothing in a system that the word *is in fact shorthand for*. That is why I emphasise that the word names as used here are indeed *shorthand* for real objects like 7.

**This may sound confusing without describing the system of (Wilks 1975a). Roughly, the 'small text structures' are meaning representations called *templates*. They consist of formulas (usually agent, action and object formulas, at least). At the same time formulas, like 7, specify the sorts of template in which they participate - i.e. 7 prefers to fit in a template which also has an agent formula that is itself for a human being. Rules like 6 connect templates together. Confusion arises only if formula parts, like (*HUM SUBJ) in 7, are thought of as *slots* to be filled. On the contrary, *whole* formulas like 7 go into slots in higher order structures, the templates.

An additional consideration is that thesauri are a traditional form of linguistic information, one which human informants are able and willing to provide. This paper is in complete agreement with Charniak's view (1975) that frame-like objects are best seen as *data*, rather than procedures, but is even more reactionary in arguing that they should if possible be *traditional forms of data*.

It seems to me an important principle, in this field as in science, to assimilate the unknown to the known if one can. The unknown notion of frames would be in no way diminished by being assimilated to the well-known one of thesauri, on the contrary.

But the present suggestion would have the effect of making the frame notion less central to a language understanding system than it has seemed to be in recent work, and not only because these are admittedly *minimal* frames, but because the centre of the picture is still occupied, as in my view it must be, by the mechanisms that acturally apply structures to the surface language. In the present note the dynamic rows become just another ancilliary mechanism for doing that. In my view, frame-like entities have only seemed central because there has been a tacit agreement on all sides to ignore questions of application, or what to *do* with them in practice.

Not that I have said very much about such procedures here, but then the space is limited. I assume that the overall structure of procedures would be like that of (Wilks 1975a) but that we would now have the ability, should we wish to use it, of expecting, say, extracting and lighting etc. to be discussed after smoking had been introduced into a discourse. These matters are separate, and rest on decisions about such matters as whether one sees a knowledge structure as essentially:

a) paralleling an incoming text, so that we tick off the expected sub-topics as they are mentioned, or

b) expectationally interpreting the text, resolving ambiguities etc. by 'knowing in advance what is going to be said', or

c) an additional mechanism for getting a surface text into a semantic representation, by resolving ambiguities and references that cannot be settled in other ways.

Decision between these, if they are indeed different, is a separate matter that brings up the question of what

hypotheses about language are being made by frame users. This topic is discussed in (Wilks 1956, sec VII) and avoided here, where I have been concerned only with a lacuna in representation: the dynamic aspect of word-sense meaning.

I have also introduced the notion of a dual semantic representation — both extensional and intensional, as it were, though in a non-standard sense of 'extensional' — to overcome the defects of both excessively primitive and excessively superficial representation. This point is quite separate from that of the 'minimal frame'.

SUMMARY AND CONCLUSION

In the proposed structure each word sense is tied by the dictionary to a sense *formula:* each formula is structured from primitives and TH's. Each of the latter points in three ways. (i) each to *its own* formula (i.e. +smoke is itself a word in the dictionary) (ii) to the rows of (semi-synonymous) words under it in the thesaurus, and possibly (iii) to a dynamic, ordered, row of TH's. (i), (ii) and (iii) can all be recursive. Similarly, inference rules which range from template to template (see earlier footnote) are composed of functions that can be expressed dually in terms of primitives and TH's (pointing as in (i)-(iii) above).

It should be noted that in this proposed structure (unlike Charniak 1975 and Rieger 1975) frame-like objects are not 'demon-like' objects pre-stacked, as would be the case if the explication of 'frame' here was a pre-set stacking of inference rules, like 6, that *are* essentially demon-like. The 'dynamic thesaurus' row is not composed of demon-like objects, but is more compact and linguistic in nature, while still preserving the essential property of expressing event sequences. However, as was pointed out, a fuller (maximal) frame could be constructed from a dynamic row by replacing the TH's with their corresponding formulas and then filling those out *as if the preferences had been satisfied.*

I have emphasised the historical similarity between the role of the thesaurus and one role being touted for the frame, then added a twist to make the similarity even stronger, and finally opted for the older form, because it is a *linguistic* form. I also argued that those who deal in frames and natural language must be sure to do *at least as well* as the old static thesaurus, and would recall to mind the adage about those who forget history being doomed to relive it.

REFERENCES

Charniak, E. (1975) Organisation and inference in a frame-like system of common-sense knowledge, in *Proc. Conf. on Theoretical Issues in Natural Language Processing.* Cambridge, Mass.

Masterman, M. (1956) The thesaurus in syntax and semantics. *Mechanical Translation.*

Minsky, M. (1974) Frame systems. *MSS,* Cambridge, Mass. and (abridged, in *The psychology of computer vision* (ed Winston) 1975.

Rieger, C. (1975) The common sense algorithm, in *Proc. Conf. on Theoretical Issues in Natural Language Processing.* Cambridge, Mass.

Riesbeck, C. (1974) Computational understanding, *Memo 4,* Inst. for Semantic and Cognitive Studies, Castagnola, Switzerland.

Schank, R. (1973) Identification of conceptualizations underlying natural language, in *Computer Models of thought and language,* (eds Schank & Colby), San Francisco.

Wilks, Y. (1975a) An intelligent analyzer and understander of natural language. *Comm. A.C.M.*

Wilks, Y. (1975b) Seven theses on Artificial Intelligence and Natural Language, *Memo 17,* Inst. for Semantic and Cognitive Studies, Castagnola, Switzerland.

Ian H. Witten

GENERATING NATURAL SPEECH FROM TEXT

This paper outlines progress towards speech synthesis systems which are capable of generating natural speech from ordinary text. Its theme is that although the engineering and phonetic tools for this task are nearing completion, many difficult problems of high-level linguistic analysis exist which have hardly been considered - for example, the generation of appropriate intonation information from text.

These problems are highlighted by describing an existing synthesis system, paying particular attention to the way the input is specified and how it differs from ordinary text. It is clear that AI techniques of syntactic and semantic analysis are necessary to achieve the goal of generating natural speech from text.

1. INTRODUCTION

The automatic synthesis of speech is usually considered to be the province of engineers and phoneticians - and indeed much of it is. However, in this paper I will argue that the engineering and phonetic problems, although not yet solved, are at least well understood, and steady progress towards their solution is being made; while fundamental difficulties of high-level linguistic analysis exist which have hardly been tackled. Surely here is scope for artificial intelligence techniques!

Although synthetic speech has its roots in antiquity (witness the oracles of Greece and Rome, and the age-old popularity of the ventriloquist's dummy), the first potentially useful synthesizer was constructed in 1951 (Lawrence, 1953). Since then, rapid technological development has kept synthesizer design on the move, and in the last few years several novel ideas have appeared as hardware. Holmes (1973) demonstrates the extent of the progress by exhibiting synthetic utterances which are perceptually identical to the originals they were copied from, even under high-fidelity listening conditions.

From a practical point of view, the usefulness of artificial speech is limited by the volume of storage required for it, the ease with which new utterances can be added to an existing corpus, and the flexibility with which elements of

the corpus can be concatenated to produce new combinations. After all, tape recorders can produce excellent speech! Holmes' high-quality speech does not require inordinate amounts of store (it needs of the order of 1000 bits/second, or 1/50'th of that required for direct digital storage of speech), but new utterances must be analysed painstakingly for several months before they can be synthesized with such success. And although it is possible to concatenate existing utterances to produce new ones, unnatural rhythm and intonation will result.

The development of electronic speech synthesizers provided a tremendous stimulus to the science of acoustic phonetics. In an important series of experiments in the mid fifties, researchers at the Haskins Laboratories in New York used synthetic speech to isolate the acoustic cues for perception of consonants (Liberman, 1957) - the relatively easy case of vowels having been studied much earlier - and this work forms the basis of most efforts to generate speech by rule from a phonetic input. These began in 1964 with a computer program that accepted phoneme strings with additional 'modifier' elements (Holmes et al, 1964), and have continued to the present day with input formats growing ever more sophisticated (Witten, 1975). In fact, a hardware synthesizer is marketed which accepts a sort of sub-phonemic input string (Votrax). Although the relentless endeavours of acoustic phoneticians have provided a wealth of information which can be incorporated into these programs (Umeda & Coker, 1975), their quality of articulation remains rather poor at present; nevertheless, I believe that they are essentially capable of acceptable speech if written with sufficient care and attention to detail.

Speech, however, is much more than mere articulation. Consider, to borrow John Laver's delightful example, the number of ways a girl can say 'yes'. Breathy voice, slow tempo, low pitch - these are all features that can be synthesized. The question is, how to identify when they are appropriate from the input of a text-to-speech system? Of course, in practice we do not aspire to produce such subtle contrasts as these by machine. However, even segmenting text into units for intonation purposes, identifying the points of major stress, and selecting pitch contours, present severe problems that have not really been tackled.

Finally, at a more pedestrian level, there is the task of converting words to their phonetic representations, and determining their stressed syllables. Here we need to make both syntactic ('wind' and 'wind') and semantic ('red lead' and 'dog lead') distinctions, as well as coping with the formidable information-processing problems of English's notorious irregularities.

This paper is posing a problem, not solving one. My thesis is that the 'speech-handling' part of a reading-aloud system is at least partially resolved, while the 'text-handling' part has hardly been touched. We have programs to generate speech from a phonetic format with additional markers (indicating rhythm, points of major stress, intonation contours, pauses, and so on), and this format is substantially the same as those used by linguists when recording natural utterances. What we do not have are techniques for generating this information from text.

The theme will be developed as follows. The next section reviews briefly the so-called suprasegmental features of speech which need to be controlled. Then follows a description of how these features can be specified - the input to the speech-handling part of an existing system. Section 4 discusses the problem of specifying the most recalcitrant feature, intonation, and finally progress is outlined on the more routine word-to-phoneme conversion process.

2. SEGMENTAL AND SUPRASEGMENTAL ASPECTS OF SPEECH

Phoneticians divide features of natural speech into segmental ones (giving segments of syllables) and suprasegmental ones (relating to properties other than pure articulation), and this distinction is useful in synthesis work as well. Of course, segmental synthesis must come first, otherwise there is no vehicle to carry the prosodic features. However, in the past little consideration has been given to systems for suprasegmental synthesis which provide enough freedom to allow production of natural-sounding, varied speech.

To be sure, most synthesis schemes have incorporated some system for prosodic control. Mattingly (1966) concentrated on prominence - for which he used acoustic features of durational increases and pitch changes - pausal features, and intonation. Prosodic markers in the phonemic input string indicated stressed syllables, final and non-final pauses, and type of intonation contour. He confined the intonation repertoire to a falling tone, a fall-rise, and a rising tone. Durations were treated as segmental features, with additional suprasegmental lengthening of stressed vowels. In a similar system, Ainsworth (1974) incorporated a tendency towards equal time intervals between stressed syllables, by lengthening the vowels of these syllables appropriately. Additionally, he used an archetypal pitch contour between stressed syllables which increased their perceptual prominence, coupled with an overall fall or rise in pitch depending on whether the utterance was a statement or a question.

Abercrombie (1967) splits suprasegmental features into two basic categories: features of voice quality and features of

voice dynamics. Variations in voice quality - which are accounted for by anatomical differences and long-term muscular idiosyncracies - have little part to play in the kind of applications I have in mind, so I consider here only elements of the second category. Abercrombie lists these as

tempo (speed of speaking)
continuity (pauses)
rhythm
range (compass between highest and lowest pitch levels)
tessitura (mean pitch level)
pitch fluctuation (intonation contour)
loudness.

Some of these features - for example, pitch fluctuation and rhythm - are intimately bound up with the way the utterance unfolds in time, and so are best passed from the text handling part of the system to the speech handling part as markers in the phoneme string; while others - tempo and tessitura, for instance - have a more global character and can be represented by parameters which are altered only occasionally. Incidentally, the richness of contrasts in speech even when reading from a book should not be underestimated. Read aloud to an audience and notice the contrasts in voice dynamics deliberately introduced for variety's sake. If stories are to be read there is even a case for controlling voice **quality** to cope with quotations and affective imitations.

3. AUTOMATIC SYNTHESIS OF SPEECH FROM A PHONETIC REPRESENTATION

Here are examples of the way utterances are specified to the speech synthesis system under development at Essex.

```
3 ^ AW T UH/M AA T I K /S I N TH UH S I S UH V
/*S P EE T SH, 1 ^ F R UH M UH F UH/*N E T I K
/R E P R UH Z E N/T E I SH UH N.
```

The standard International Phonetic Association symbols are coded into one or two letters to cope with the limitations of computer input devices. Slashes mark the **foot boundaries** (which correspond to stressed syllables) and an asterisk identifies the point of major stress of an intonation pattern. Utterances must be divided by punctuation marks into intonation units called **tone groups**, and the shape of the intonation contour is specified by a numeral at the start of each tone group. Crude control over pauses is achieved by punctuation marks: full stop, for example, signals a pause while comma does not. The ^ character stands for a 'silent stress' or breath point.

The program that deals with the prosodic features of rhythm and intonation works roughly as follows (full details can be

found in Witten, 1975). Each foot is split into its constituent syllables, and from their types a rhythm for the foot is determined. According to the theory of isochronicity, each foot has approximately the same duration, and this is divided amongst the syllables according to their rhythm. The question of where each syllable begins and ends is a difficult one: research indicates the existance of a rhythmic 'tapping point' in a well-defined place in each syllable (usually just before the vowel). Then the phonemes of a syllable are classified into those whose duration is intrinsically determined (like the initial 's' and 't', the 'i', and the final 's', of 'straits' - try speaking the word in context, slowly but with a natural rhythm), and those where it is extrinsically determined by the rhythm (the 'a' and the second 't' of 'straits'); and the syllable time is divided amongst them accordingly (Lawrence, 1974). Intonation is handled using the tone group number to access a table of standard intonation contours (see Section 4).

The above seven attributes of voice dynamics are controlled as follows.

tempo - a special directive can alter the target foot duration

continuity - crude control achieved by punctuation

rhythm - specified by the slashes which mark foot boundaries

range, tessitura, pitch fluctuation - all controlled together by the intonation contour specifications, although the individual attributes could easily be factored out

loudness - not controlled at a suprasegmental level at all (contrary to intuition, loudness is a very **weak** cue for stress, and is outshadowed by rhythm and intonation).

4. INTONATION CONTOURS

The pitch of a voice is an essentially continuous, time-varying quantity, and it is not clear how it can be specified in a computer system. Linguists, however, are expert in the difficult area of discretizing essentially continuous (in both the spatial and the temporal sense) information, and have attacked the problem of classifying intonation with gusto.

Theories of British English intonation invariably segment utterances into units, which I call 'tone groups', each with a single salient or 'tonic' syllable (and a number of other syllables). The most appealing classification of tone groups, from the point of view of computer implementation, is Halliday's (1970), which identifies five different primary

intonation contours, each hinging on the tonic syllable. Several secondary contours, which are variations on the primary ones, are defined as well.

Our experience with Halliday's classification is that it does not permit a rich enough variety of pitch patterns to be generated for natural-sounding synthetic speech - its repertoire is too restricted. For example, an inflection on the pre-tonic part of a tone group is essential for naturalness, and the magnitude of the inflection should be altered slightly to add interest: but a considerable increase in it produces a semantic change by making the utterance more emphatic. These fine distinctions are not represented in Halliday's system.

We have developed a method of intonation specification which is intended to permit close control over those aspects of an intonation contour that affect the way it is perceived. Although this system is tentative and still under development, I will outline the form it takes at present. Ten pitch parameters are defined, each with a simple meaning. The overall pitch movement is controlled by specifying the pitch at three places: the beginning of the tone group, the beginning of the 'tonic' syllable (the one which bears major stress), and the end of the tone group. Provision is made for an abrupt pitch break at the start of the tonic syllable. The pitch is interpolated linearly over the first part of the tone group (up to the tonic syllable) and over the last part (from there to the end), except that it is possible to specify a non-linearity on the tonic syllable, for emphasis. On this basic shape are superimposed two finer pitch patterns, one to allow 'continuation' between tone groups, and the other to give a lilt, synchronized with the foot structure, to the initial part of the contour. This last is necessary to prevent the deadening effect of steadily increasing or decreasing frequency. Figure 1 shows details of the ten quantities.

The intention of this parametric method of specifying intonation contours is that the parameters should be easily derivable from semantic variables like emphasis, novelty of idea, surprise, uncertainty, incompleteness. Roughly speaking, parameters 4 and 7 control emphasis, 7 by itself controls novelty and surprise, and 8 and the relative sizes of 5 and 6 control uncertainty and incompleteness. Certain parameters (notably 9 and 10) are defined because although they do not appear to correspond to semantic distinctions, we do not yet know how to generate them automatically.

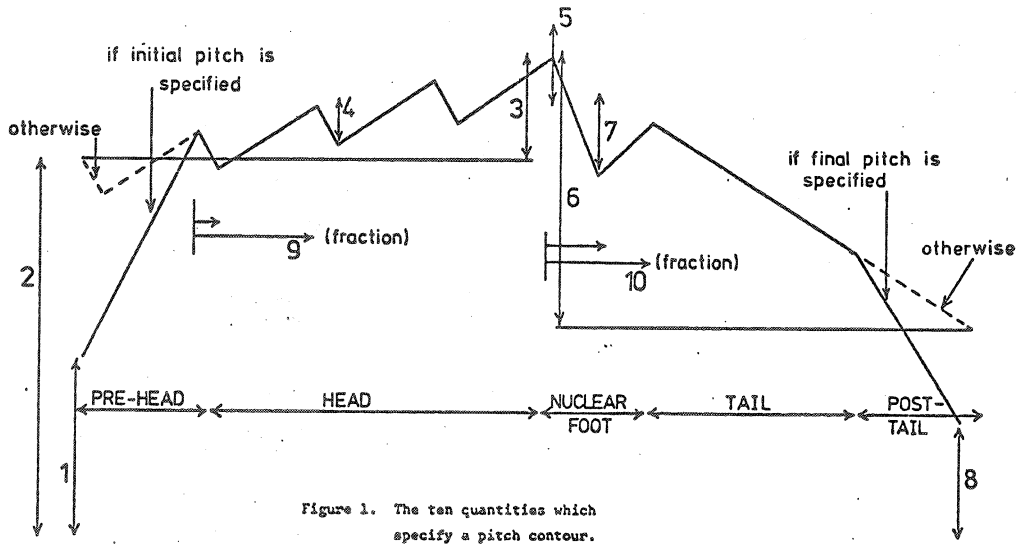


Figure 1. The ten quantities which specify a pitch contour.

5. GENERATING THE PHONETIC REPRESENTATION FROM TEXT

The most striking difference between the input to our existing speech synthesis system and the target input format of ordinary text is the **phonetic** specification of utterances. And yet our experience indicates that the 'grapheme-to-phoneme' conversion problem is not nearly so tough as eliciting suprasegmental information from text.

For example, we have implemented a straightforward string-matching and replacing procedure which takes little account of the morphology of words (described by McIlroy, 1974), and found it to have a reading age of 12 years, according to a standard Schonell test (Schonell, 1951). We can enter

> 3 ^ auto/matic /synthesis of /*speech, 1 ^ from a gra/*phemic /represent/ation.

and it produces

> 3 ^ AW T O /M AA T I K /S I N TH E Z I Z UH V /*S P EE T SH, 1 ^ F R UH M E I G R AA /*F E M I K /R E P R E Z E /N T E I SH UH N.

Quality could be improved still further by a simple rule for 'reducing' or centralizing the vowels of unstressed syllables.

Unfortunately, this grapheme-to-phoneme procedure, being ignorant of morphology, cannot detect the stressed syllables of words and place foot boundaries accordingly. We are currently implementing a morph dictionary so that prefixes and suffices can be treated properly (Allen, 1976).

6. CONCLUSION

The aim of this paper is to stimulate interest in the high-level problems of speech synthesis from text. The engineering and phonetic tools necessary to generate intelligible speech from a phonetic input with suprasegmental markers exist already, and the quality of artificial speech can be expected to improve slowly but steadily in the future. Most of the preceding sections have been intended to help make this point: we saw what information - especially suprasegmental information - is necessary for synthesis, how it can be specified, and how intonation can be handled. The question of grapheme-to-phoneme conversion has been discussed and shown to be tractable.

What problems are left? Let me isolate some.

Intonation group boundaries. If text is liberally punctuated, then the punctuation marks identify many of the boundaries of intonation groups. Unfortunately, current literary fashion favours sparse punctuation. Moreover, since most theories of British intonation insist that a tone group has a unique tonic syllable (not to do so would exacerbate the problem of classification), it is sometimes necessary to place a tone group boundary where a punctuation mark could never be (see example below).

Tonic syllables of tone groups. The tonic syllable occurs at the semantic focus of an intonation group; which can be caricatured as the most unusual new word. A simple example of Halliday's illustrates the problems:

> /this of course de/pends on the /country where they
> /*live

can be read as

> /this of course de/*pends on the //country where they
> /*live

or, in a different context, as

> /this of course de/pends on the /*country where they
> //*live

(// indicates a tone group boundary).

Tessitura and range. These both increase with dramatic tension (and are reduced for material in parentheses).

Intonation contours. The type of an intonation contour is sometimes apparent from the punctuation (statement/question, for example). Additionally, the introduction of a new concept - often at the beginning of a paragraph - carries a distinctive 'novelty' intonation. Section 4 has shown how semantic variables can be translated into appropriate pitch

contours.

Pauses. Here again, punctuation is invaluable. The old rule 'count one for a comma, two for a full stop' goes a long way.

Foot boundaries. Even though morphemic analysis of text helps to detect stressed syllables, it does not completely solve the problem. The 'of course' phrase in the example above does not have a foot boundary, although '/this of /course de/pends ...' would be an acceptable rendering.

Miscellaneous difficulties of reading. Dates, numerals (especially Roman!), abbreviations, titles, etc., all introduce peculiar practical problems.

Surprise, novelty of words, novelty of ideas - these all affect speech. Clearly, a semantic analysis like that of Wilks (1975) is necessary - but certainly not sufficient - to resolve these difficulties.

ACKNOWLEDGEMENT

The help and support of the speech research group at Essex is gratefully acknowledged, especially the constructive criticism of Pete Madams, Roger Moore, and Richard Pope. The work forms part of a joint research project with the Department of Computer Science, University of Calgary, and is partially funded by the Science Research Council.

REFERENCES

Abercrombie, D. (1967) **Elememts of gemeral phonetics.** Edinburgh Univ. Press.

Allen, J. (1976) Synthesis of speech from unrestricted text. **Proc. IEEE.**

Halliday, M.A.K. (1970) **A course in spoken English: Intonation.** London: Oxford Univ. Press.

Holmes, J.N., I.G. Mattingly, & J.N. Shearme (1964) Speech synthesis by rule. **Language and Speech 7**(3):127-143.

Lawrence, W. (1953) The synthesis of speech from signals which have a low information rate. In **Communication Theory** (ed. W.Jackson), pp.460-469. London: Butterworth.

Lawrence, W. (1974) The phoneme, the syllable, and the parameter track. **Proc. Speech Communication Seminar,** Stockholm, August 1-3.

Liberman, A.M. (1959) Some results of research on speech perception. **J. Acoust. Soc. Am. 29**(1):117-123.

Mattingly, I.G. (1966) Synthesis by rule of prosodic features. **Language and Speech 9**(1):1-13.

McIlroy, M.D. (1974) Synthetic English speech by rule. **Computing Science Tech. Rept. 14,** Bell Labs., Murray Hill, N.J.

Schonell, F.J. (1951) **The psychology and teaching of**

reading. Oliver & Boyd.
Umeda, N. & C.H. Coker (1975) Subphonemic details in American English. In **Auditory analysis and perception of speech** (eds. G.Fant & M.A.A. Tatham), pp.539-564. London: Academic Press.
Votrax. Vocal Interface Division, Federal Screw Works, U.S.A.
Wilks, Y. (1975) An intelligent analyzer and understander of English. **Comm. A.C.M. 18**(5):264-274.
Witten, I.H. (1975) A flexible scheme for assigning timing and pitch to synthetic speech. **Tech. Rept. EES-MIS-SYN1-75**, University of Essex, U.K.

Richard M. Young

DESIGN CHOICES FOR A WORLD-MODELLING SYSTEM

Abstract. The problem of adequately simulating an environment containing simultaneous actions and continuous processes is discussed. Two different approaches are outlined and compared on six design issues. It is argued that the six choices are not independent, and that when their mutual constraints are taken into account there are only four main families of world-modelling systems, differing primarily on the issues of *Units of Simulation* and *Activation of Processes*. The strengths and weaknesses of the different designs are explored.

Keywords: World modelling, simulation, robot planning, cognitive development.

A few years ago Hendrix (1973) published a paper describing the simulation of a robot's world containing "simultaneous actions and continuous processes". Hendrix's work was done in recognition of the fact that although many Artificial Intelligence programs involve the simulation of a simple world - e.g. "robot planning" programs, or Winograd's (1972) well-known "blocks world" - most of these worlds are totally passive and do nothing but respond to the operators the robot applies to them. They lack two crucial properties of the real world,

(a) simultaneity: the fact that several things may be going on at once;
(b) autonomy: the fact that the world does not necessarily remain quiescent at times when the robot is not acting upon it, and that things can happen without the robot having caused or intended them.

Hendrix proposed, therefore, a system which included "... the ability to model growing flowers, running streams and a sun which wanders gradually across the heavens". Independently, I had been proceeding with the implementation of a program to represent a toy "blocks world" to serve as an environment for studying cognitive development in young children (Young 1974, 1976). Interestingly enough, on some issues these two systems adopt positions as different as can reasonably be imagined, yet it is far from obvious that one system is uniformly better than the other. The contrast between them has proved invaluable for clarifying the possible options in the design of a "world

model", and it is to this topic that the present paper is addressed.

We begin, then, with a brief description of each system. Although failing to do complete justice to either, this will hopefully highlight the features of the systems most relevant to our present purpose. After that we will discuss a number of crucial "design issues" - or dimensions of choice - for world modelling programs, illustrating their significance by reference to the two systems. Neither of these systems has a name, so they will be referred to here simply as HS and YS, for Hendrix's system and Young's system respectively. It should be stressed that their comparison is not to be taken in a partisan spirit. Rather than regarding the two systems as rivals, it should be realised that it is their very differences which makes them useful for understanding the design of world-modelling programs.

THE TWO SYSTEMS

HS is derived from ideas in the Stanford robot planning program STRIPS (Fikes and Nilsson 1971), and many of its features share the same spirit. Most importantly, all information about the current state of the world is represented by *assertions*, or facts, and the modelled world is altered by the application of operators (here called SCENARIOS) which delete old information and add new. The units in which the simulation is structured correspond to *events* or activities in the simulated world - such as an alarm clock being set, the clock ringing, a bucket being filled with water, or a robot moving across the floor.

For example, the scenario for setting an alarm clock to ring at a given time, Cstime, is called SETALARM(Cstime). Initiating the process SETALARM(7.30 am) results in deleting the fact (ALARM OFF) and adding the new fact (ALARM SET 7.30 am). The actual (simulated!) ringing is the responsibility of a different scenario, SOUNDALARM, which is activated when the simulated clock time reaches the time the alarm was set for, 7.30 am. When run, it deletes the fact (ALARM SET ...) and adds (ALARM RINGING). The latter fact continues to hold until yet a third scenario, OFFALARM, is used to delete it and to restore (ALARM OFF).

Information that changes continuously with time is represented by an assertion containing time-dependent variables. The scenario for FILLBUCKET, for instance, adds an assertion (CONTENT BUCKET Ycontent) and contains a simple equation for calculating Ycontent from the current time and the rate of filling. Interactions between scenarios - and hence most of the interesting behaviour in the system - are mediated by a monitor which detects when the conditions for activating or terminating a scenario are satisfied. The job of the monitor is repeatedly to:-

(a) check the continuance conditions of all activated processes and the initiation conditions of all potential ones, in order to find the earliest time when some change is due to take place;
(b) advance the system clock to that earliest time; and
(c) make the appropriate change.

YS is aimed at simulating a toy world where the blocks obey a simplified physics - for example, they always remain vertical so one block can be supported by another even if they overlap only fractionally. In contrast to HS, YS draws its inspiration from the ideas in the programming language SIMULA (Birtwistle, Dahl, Myhrhaug and Nygaard 1973). The simulation is structured as a collection of interacting co-routines, each corresponding to an *object* in the simulated world: a block, the child's hand, and so on. Information about the current state of the world is held by the routine corresponding to the object for which it is most relevant.

In the case of the alarm clock YS would use just a single routine, called ALARM, to store information about the status of the alarm (OFF, SET or RINGING) and the time it was set for. The alarm is set by a *request* to the routine, ALARM (SET 7.30 am), and would eventually be turned off by a request of the form ALARM (OFF). The routine itself has the responsibility for seeing that the alarm "rings" at "7.30".

As already indicated, activity in YS is triggered by messages passed from one routine to another. If the bell of the alarm were to be modelled by a separate routine, for example, it would be told to ring - BELL (START) - by ALARM at the appropriate time. Time-varying quantities present no special problems, since the routine can do whatever is needed to compute the value whenever it is asked for. In a simple case like ALARM, the current state is presumably held in a variable local to the routine. In a more complicated case, such as the position of a block falling under gravity, an arbitrary computation can be invoked to determine the answer. YS, unlike HS, is willing to use *incremental simulation* to model dynamic aspects of the world. In other words, a certain interval Δt of simulated time is chosen for the "grain" of the system, and the simulation is advanced in discrete increments of Δt. The role of the monitor is then very simple. As in SIMULA, a queue is maintained of active routines ordered by the times at which each is to be resumed. All the monitor has to do is to remove the first routine from the queue, advance system time to its associated resumption time, and then re-activate the routine.

COMPARISON OF THE TWO SYSTEMS

Figure 1 summarises six differences between the two systems. Most of these points have been mentioned in the descriptions just given, but they are not wholly independent and our intention now is to explore their relationships and mutual constraints.

Motivation may seem like a strange starting point, but in fact it is central to the design of HS and is what makes HS hang together as a coherent system. Since HS is intended for use in a robot planning program, matters are simplified if the world-simulation is expressed in the same terms as those used by the planning procedures. The actions chosen by the planner are then guaranteed to lead to their intended effect in the simulated world. It pays to maximally confound the simulation of the world with the planning for it, and indeed most robot planning programs do not even draw the distinction. Given this, and the fact that HS is intended for a STRIPS-like planner using a state-space representation (Nilsson 1971), it follows that the simulation itself will also have to be in terms of states and operators, and the design choices of HS are thereby determined. The decision about *Units of Simulation* is made, since operators correspond to changes in the state of the world, i.e. to activities and events. The *Activation of Processes* is determined, since the operators must have explicitly stated pre-conditions and effects to be useful for planning. The *Type of Processes* is determined as a (somewhat extended kind of) operator. The *State of the World* has to be a globally accessible set of facts, in order to provide the basis for the planning. Even the treatment of the *Simulation of Time* is suggested (though not actually fixed) by the intended use, since it becomes natural for simulated time to be advanced from the moment of application of one operator in the planning space to the next.

Figure 1: Differences between the two systems

	HS	YS
Motivation:	Robot planning	Cognitive development
Simulation of Time:	Predict next "significant" change	Incremental
Activation of Processes (etc.):	Initiating conditions (etc.) checked by monitor	Messages between routines
Type of Processes:	SCENARIOS (extended operators)	ROUTINES (interacting co-routines)
Units of Simulation:	Events, activities	Objects
State of the World:	Assertional database	Information local to routines

In the case of YS, these constraints are all missing. The purpose of YS is to aid in studying the developmental processes that enable a child to acquire new production rules (Young 1974, 1976). The child himself has access to the outside world only by means of his senses - particularly touch and vision - and if the psychological model is to be right, it too must respond only to the *appearance* of the simulated world. There is no need for the psychological model to have access to the mechanism of the world model, so there is no reason (on these grounds) not to write "arbitrary code" to represent the workings of the environment. As a result, YS has more freedom to choose among the available design options. However there are still some constraints, and their consequences are explored further below.

The decision as to how the *Simulation of Time* is handled is important in its own right. To be able to treat time by advancing it to the occurrence of the next significant change, it has to be possible to analyse the world model so as to be able to predict *when* this change will occur. From this need

again follow the design decisions that characterise HS. The *Activation of Processes* has to be done through explicitly stated conditions, so that the monitor can tell where the next change will occur. The *State of the World* has to be in an accessible database to enable the monitor to work out when it will occur. And so on. In addition to all this, the need to predict changes means that dynamic quantities are restricted to being *simple* functions of time. HS is intended to work mainly with linear relations, though Hendrix does include one example of a quadratic relation resulting from an uniform acceleration. Arbitrary functions of time computed by arbitrary code are out of the question. Dynamic quantities have to be represented by easily-solved equations.

One systematic way of discovering all the dependencies between the design issues of Figure 1 is to consider them one at a time and ask whether they constrain each of the other issues. For example, suppose one were to implement *Activitation of Processes* by means of initiating conditions checked by a monitor. Does this affect the *Motivation*? Clearly not since the motivation is prior to questions of implementation. Does it affect the *Units of Simulation*? Yes, because it is only events (and not objects) that have "initiating conditions". Does it affect the *Type of Process*? No, that is an independent issue. The *State of the World*? Yes, because the monitor must have access to the current situation. The *Simulation of Time*? Not necessarily, but the use of initiating conditions (and the necessary presence of a monitor) makes it natural to advance time to when the next condition is to hold.

Proceeding in this way, it turns out that when all the interdependencies are explored there remain only eleven different ways of making a consistent set of choices. Even of these, some cases are barely worth distinguishing. Consider the question of *State of the World*, for example. Even when one has a free choice, whether one uses a public database or private information hardly seems an important decision in its own right. Disregarding these minor variations there are just four main "families" of world modelling systems:

(a) If the *Motivation* is for planning, or if the *Simulation of Time* is to be handled predictively, the one is led to a system similar to HS.
(b) Otherwise one has a choice of *Units of Simulation*. Choosing an object-centered representation leads to a system like YS.
(c) If an activity-based system is chosen, there is still a choice about the *Activation of Processes:* either by explicit, monitored conditions, as in HS,
(d) or directly by inter-process communication, as in YS.

What are the strengths and weaknesses of each of these designs?

PROS AND CONS

The discussion will be clarified if we first distinguish between two different aspects of the problem of simulation, which we can call the "geometrical" aspect and the "causal" aspect. Consider the problem of simulating the behaviour of a number of billiard balls all moving around bumping into each other and the walls of the billiards table. From a logical or *causal* point of view there is no difficulty in seeing what is going to happen. Each ball continues moving uniformly until it collides either with another ball or with the table, after which it moves off again with a new velocity towards its next collision .... The difficulty with this problem lies purely in the *geometrical* aspect of actually calculating which balls are going to collide, where, and when. Consider next a problem in which a robot is standing near a block A, which is touching another block B, which in turn is touching an immoveable wall. The robot tries to move A one inch in the direction of the wall. In this case there is no problem with the *geometry* of the situation: if A were free to move it would simply shift one inch towards the wall. The difficulty in modelling the situation correctly lies in the problem of representing the *causal chain* indicating that if A is to move it will have to push B, and if B is to move it will have to budge the wall, but the wall can't move, therefore neither can B, therefore neither can A ....

One of the difficulties with using a system like HS is that if the monitor is to be kept reasonably simple, then all information has to be represented and handled in the system in a uniform way. This means in particular that the geometrical and the causal aspects of the problem have to be dealt with by the same techniques, and this can be very difficult. Within the framework of YS, on the other hand, there is no problem in mixing different techniques. With regard to *Activation of Processes*, for example, although processes in YS are normally activated directly by other processes, there is always the option of using the scheduler instead. The call ALARM (SET 7.30 am) discussed earlier would simply result in the ALARM routine scheduling itself to be resumed at 7.30. Similarly, with regard to *Units of Simulation*, routines are usually chosen to represent objects rather than activities. But consider the problem of simulating the noise made by two blocks sliding against each other. There are several ways to handle this in YS, but the best is for the blocks, when they first make contact, to set up a "noise process" which thereafter continually checks that the blocks are still sliding. If they are, it produces its "noise"; and when they cease to be, it goes out of existence. Once the need for uniformity has been removed, it becomes possible to introduce some flexibility into the system. One way to take advantage of this is by the use

of "experts" - pieces of program which do not themselves correspond to any entity in the simulated world, but which are good at solving particular subproblems, such as the geometrical aspects of the simulation. In fact it is an open question whether one can write a practical model *without* the use of experts of various kinds. In my simulation of a blocks world, for example, there is a "spatial expert" which knows about the entire spatial layout: where each block is, its size, and so on. It is efficient at tasks like deciding whether or not a particular trajectory of a given block will collide with any other blocks, and the expert is frequently consulted on such matters by the individual block routines.

A consideration related to the *Units of Simulation* issue is that a decision to base the simulation on the different kinds of activities that occur in the simulated world involves an open-ended commitment to a large number of different process types. The problem is made worse in HS itself by the fact that a variable in HS may depend on at most one process. This means that as various situations involving different conjunctions of activities are considered, the number of different scenarios required increases combinatorially. Consider the situation where a bucket is being filled from a tap and simultaneously being drained by a siphon, say. There is no way in HS to model this by having two existing scenarios, FILLBUCKET and DRAINBUCKET, both active and somehow co-operating to keep track of the correct content. Instead, a new FILLANDDRAINBUCKET scenario is needed. YS on the other hand has no difficulty here, since during each time interval the bucket routine will be told both "Such-and-such amount of water has been poured into you" and "So much water has been removed from you", and the resulting change in content will simply reflect the difference between the two.

This is certainly not meant to suggest that an object-centered approach encounters no problems of analysis. On the contrary, the fact that physical objects can be decomposed in different ways and at different levels is well known, and poses considerable logical and philosophical puzzles. But the approach does not suffer from the same combinatorial problem. For a given purpose, a satisfactory division into "objects" is usually easy to find, and leads to a small set of different routines. Thereafter, the objects can be modelled interacting with each other in novel combinations simply by running their respective routines.

In order to complete the discussion, we need to introduce one further notion, that of *responsibility*. The idea is best explained with respect to YS, where a routine R modelling a particular entity may call for help on other routines and various experts, and so on, but R itself retains the *responsibility* for seeing that the simulation of that entity is done

correctly. This idea already came up in an earlier example: even if the BELL of an alarm clock is modelled by a separate routine from the ALARM itself, it is clearly still the responsibility of ALARM to tell BELL when to ring.

In a system like HS the question of responsibility can never arise, since it is impossible for one process to activate another directly, and *all* responsibility lies with the monitor. This raises as many problems as it solves, however. Consider the situation that arises when an object is moving because it is being held by a moving robot. There is no way in HS for the process representing the movement of the robot to communicate directly with the process representing the movement of the object. Instead one has to add enough assertions about what is moving and about what is holding what that the monitor, by running the two *independent* processes, can manage to keep the object and the robot in step. Responsibility has been diffused instead of being concentrated on, say, the robot. In YS the situation is handled quite differently. Whenever an object is grasped, the object's routine is told that it has been grasped, and by whom. Thereafter, any questions about the object's current velocity, or whatever, are simply passed on to the robot holding it.

The question of responsibility is intimately bound up with the problems that arise in trying to cope with two or more different agents, such as robots. With YS, having two agents leads to the need to decide responsibility whenever there is a conflict between them. But once that is settled, the system should handle the situation just as normal. The approach taken by HS, however, suffers a fundamental difficulty. As it is presently structured HS simply cannot cope with two agents, because the inferences that the scenarios rely on are no longer valid. The problem here is one known in Artificial Intelligence as the "qualification problem" (e.g. McCarthy and Hayes 1969). The usual example is that a car can be used as transport to the airport, *unless* the car has a puncture, or a flat battery, or there is a traffic jam, etc. ... , the point being that it is impossible (and wrong-headed to try) to think beforehand of everything that could possibly go wrong and then use these to hedge the original assumption with qualifications. In the case of HS, the corresponding need would be to qualify all inferences by a clause to the effect of "... provided that no other agent has decided differently", but this is no solution.

Finally, let us return for a moment to the FILLBUCKET situation. Suppose the bucket is under the tap and filling nicely when someone comes along and interposes a deflector between the tap and the bucket, so that the water falls on the ground and runs to waste. HS is unable to model this situation correctly (other than by an *ad hoc* ATTEMPT-FILLBUCKET-WITH-

INTERPOSED-DEFLECTOR scenario) basically because it is no-one's *responsibility* to tell the FILLBUCKET process that interposing the deflector means that the bucket will stop filling. The only way HS could cope with this would be for the conditions of the FILLBUCKET scenario to include the qualification "... and provided that no deflector is interposed ...", which we have just seen is unreasonable. The situation would obviously be hard for YS to handle, also. But the difficulty there is mainly the geometrical one: working out that the deflector will interrupt the stream of water and cause it to miss the bucket. The chain of responsibility on the other hand - from deflector routine to water-jet routine to bucket routine - is quite clear.

CONCLUSION

Where does this discussion leave us? Evidently the problems of writing a good world-modelling program are far from understood, and both the systems described here have severe limitations. One point that emerges with clarity however is that there is a fundamental choice between *flexibility* on the one hand and *aptness for planning* on the other. If the world model is intended to serve as the basis for rational planning in a state-space, it will have to share many of the features of Hendrix's system. But otherwise the system can probably benefit by taking advantage of the flexibility offered by the alternative design options.

Acknowledgements. I should like to thank: the Social Science Research Council, for financial support; Jim Howe, for encouragement and the excellent facilities of the Bionics Research Laboratory; Tim O'Shea, for drawing my attention to Hendrix's article; and Gary Hendrix, for one *very* long conversation.

REFERENCES

Birtwistle,G.M., O.J.Dahl, B.Myhrhaug & K.Nygaard (1973) *SIMULA begin.* Auerbach Publishing.

Fikes,R.E. & N.J.Nilsson (1971) STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2,* 189-208.

Hendrix,G.G. (1973) Modelling simultaneous actions and continuous processes. *Artificial Intelligence, 4,* 145-180.

McCarthy,J. & P.J.Hayes (1969) Some philosophical questions from the standpoint of artificial intelligence. In *Machine Intelligence 4* (eds B.Meltzer & D.Michie) Edinburgh: Edinburgh University Press.

Nilsson,N.J. (1971) *Problems Solving Methods in Artificial Intelligence.* McGraw-Hill.

Winograd,T. (1972) *Understanding Natural Language.* Academic Press.

Young,R.M. (1974) Production systems as models of cognitive development. *Proceedings of First AISB Conference*, 284-295. Also as *Bionics Research Report No. 22*, Department of Artificial Intelligence, University of Edinburgh.

Young,R.M. (1976) *Seriation by Children: A Production-System Approach*. Basel: Birkhauser Verlag.