Tony Leslie
Essex.

PROCEEDINGS OF THE AISB/GI CONFERENCE ON

Artificial Intelligence

HAMBURG

18–20 July, 1978

© SSAISB and GI (1978)

Enquiries about the purchase of copies of this volume should be addressed to:

Derek Sleeman
Department of Computer Studies
The University
Leeds, LS2 9JT
U.K.

AISB/GI 1978

Proceedings of
AISB/GI CONFERENCE

held at
UNIVERSITY OF HAMBURG

18th to 20th July, 1978

Program Committee

ALAN BUNDY (Edinburgh)
STEVE HARDY (Sussex)
H. -H. NAGEL (Hamburg)
JACQUES PITRAT (Paris)
DEREK SLEEMAN (Leeds)
YORICK WILKS (Essex)

Program Chairman: DEREK SLEEMAN
General Chairman: H. -H. NAGEL

Co-sponsored by: The Society for the Study of Artificial Intelligence and Simulation of Behaviour *and*

Gesellschaft für Informatik

List of persons who refereed or who offered to referee papers

AIELLO, L.
AMBLER, P.
ANDERSON, B.
BARSTOW, D.R.
BIBEL, W.
BIGGS, C.
BORNAT, R.
BRADY, J.M.
BROOKS, M.J.
BROWN, F.M.
BRUDERER, H.E.
BUNT, H.
CAMPBELL, J.A.
CERCONE, N.J.
CHANDLER, I.V.
CHISHOLM, I.H.
CLARKE, M.R.B.
COLE, A.J.
DARLINGTON, J.
DEUSSEN, P.
DORAN, J.
EISENSTADT, M.
FOITH, J.P.
GAINES, B.
GLANC, A.
GOEBEL, R.
GREUSSAY, P.
HANNA, F.K.
HARTLEY, J.R.
HARWOOD, W.T.
HAVENS, W.S.
HAYES, P.J.
HAYES-ROTH, R.
HEWITT, C.E.
HOWE, J.A.M.
IZARD, S.
KONRAD, E.
LARCOMBE, M.
LAUBSCH, J.H.
LEE, M.H.
LOZOVSKY, V.S.
LUGER, G.F.
MACKWORTH, A.K.
MCCOLL, W.
MCQUEEN, D
MELTZER, B
MICHALSKI, R.
MICHIE, D.
O'SHEA, T.
PALMER, M.
PASK, G.
POPPLESTONE, R.J.
PRINI, G.
RAULEFS, P.
RAUSCHNER, H.D.
REYNOLDS, C.F.
RICHTER, M.M.
ROSNER, M.
ROWBURY, C.R.
SANDEWALL, E.
SAYA, H.
SCHA, R.
SCHNEIDER, H.J.
SCHUBERT, L.K.
SCHWARZ, J.
SIEKMANN, H.-J.
SLOMAN, A.
SMITH, N.J.
SMYTH, M.B.
SOUL, M.
SRIDHARAN, N.S.
STEEDMAN, M.
TATE, A
UNVERICHT, E.
VALIANT, L.G.
VEENKER, G.
WAHLSTER, W.
WALDINGER, R.
WARREN, D
WERTZ, H.
WITTIG, T.
WOLFF, J.G.
WRIGHTSON, G.
YOUNG, R.M.

FOREWORD

There were eighty-four papers submitted to the Conference and the Program Committee was faced with the difficult task of fitting a sub-set of these into a three-day program. Various possibilities were suggested including parallel sessions, 'short' papers, and the use of display-boards. In the end, it was decided to use *all* of these 'devices' to some extent. These Proceedings contain the three invited review papers, fifteen long papers (less than fourteen pages) and thirty-two 'short' papers (less than seven pages); at the Conference these various categories have been allocated sixty, thirty and fifteen minutes respectively. I hope that you will evaluate each of these 'devices', and that you will complete a simple questionnaire at the end of the Conference which will help guide the planning of future events.

The Conference has been based around three main themes, which are the subjects of the review lectures that begin each of the days. These topics are *three* of the current major activities in the field, namely: AI's approach to designing and implementing sizable systems, what has been called "Representation of Knowledge", but which McDermott would have us describe as "Notational Engineering", and an aspect of "Computer Perception". The reviews in this volume by Bert Horn, Drew McDermott and Bob Wielinga deserve to be read very carefully and will, I am sure, suggest many *issues* which will be debated at length both during the Conference and subsequently. These themes are again taken up in the several discussions/debates; in particular the debate on the second day is essentially about "Representation and Control in Vision Systems" and hence should draw together these several (contrasting) themes. At the final session on Thursday afternoon, Leonard Uhr and Donald Michie will present overviews of the Conference from their own particular perspectives.

I hope you will agree that the Conference Program† and the Proceedings look very interesting, and I very much hope you will read them before the Conference so that you will be able to actively take part in the various discussions. After all, a considerable amount of *extra* effort has been expended to enable these to be circulated before the Conference.

Many people have contributed to the effort of producing these Proceedings. The reviewers and particularly the Program Committee have had a major influence upon the structure and content of the Conference. Indeed, I feel that the reviewing task would have been an impossible one without the active participation of the Program Committee. It is a pleasure also to acknowledge both the advice and efficient service which we have received from the University Printing Service, who have printed the Conference Poster and program as well as these Proceedings.

†*Published as a separate booklet and available from the Program- and the General-Chairman.*

Contd.

These Proceedings would never have seen the light of day, nor would the twelve-hundred pieces of Conference mail have been despatched without the valiant efforts of Pat Greenwood and Barbara Lewis.

Finally, it is my pleasure to thank the Gesellschaft für Informatik for taking the initiative in suggesting this joint-venture, and to thank Professor Nagel, in particular, for all the work that he and his colleagues in Hamburg have put in.

Once again, may I wish you happy reading!

Derek Sleeman
Leeds, May 1978

CONTENTS

How one learns strategies: processes and representation of strategy acquisition.
ANZAI, Y. (CMU)
p1

Experience with a refinement paradigm in a Knowledge-based automatic programming system.
BARSTOW, D.R. (Yale)
p15

On strategies for the synthesis of Algorithms.
BIBEL, W. (Munchen)
p22

A (self-adapting) network for recognition of visual structures.
BIRCH, F. (Sussex)
p28

Dialogues in ACTOR nets.
BOHM, H.P. (Bonn) FISCHER, H.L. (Karlsruhe) RAULEFS, P.
p33

The FIT approach to AI languages.
BOLEY, H. (Hamburg)
p35

Dangerous Behaviour.
BRADY, J.M. (Essex)
p42

Experimental Learning Model.
BRAZDIL, P. (Edinburgh)
p46

Understanding the image forming process.
BROOKS, M.J. (Essex)
p51

A sequent calculus for modal quantificational logic.
BROWN, F.M. (Edinburgh)
p56

Analysing and representing Natural Language in Logic.
BROWN, F.M. (Edinburgh) SCHWIND, C.B. (Munchen)
p66

Knowledge about knowledge: making decisions in Mechanics Problem Solving.
BUNDY, A. (Edinburgh) LUGER, G. MELLISH, C. PALMER, M.
p71

A theorem prover dating a semantic network.
CHAMPEAUX, D de (Amsterdam)
p82

Determining the orientation of surfaces from optical flow.
CLOCKSIN, W.F. (Edinburgh)
p93

Problem solving by hyperplanning.
EICK, Ch (Bonn) RAULEFS, P. (Karlsruhe)
p103

Paramodulated connection graphs.
EISINGER, N. (Karlsruhe) SIEKMANN, J. WRIGHTSON, G.
WRIGHTSON, G.
p105

A system for visual-like understanding of LISP programs.
GOOSSENS, D. (Paris)
p110

A method for computing heuristics in problem-solving.
GUIDA, G. (Milan) SOMALVICO, M.
p115

HAM-RPM: Natural dialogues with an artificial partner.
HAHN, W.V. (Hamburg) HOEPPNER, W. JAMESON, A. WAHLSTER, W.
p122

A multiprocessor system for knowledge representation and manipulation.
HANNA, F.K. (Canterbury) HARWOOD, W.T.
p132

Program synthesis in predicate logic.
HOGGER, C.J. (London)
p138

Vision
HORN, B.K.P. (MIT)
p147

Ani: An example of computational creativity.
KAHN, K.M. (MIT)
p164

A sane algorithm for the synthesis of LISP functions from example problems; the "BOYER and MOORE algorithm".
KODRATOFF, Y. (Paris) FARGUES, J.
p169

A study in the applications of theorem proving.
LUCKHAM, D.C. (Stanford) MORALES, J.J. (Rutgers)
SCHREIBER, J.F. (Munchen)
p176

A quasi-parallel countour following algorithm.
MERO, L. (Budapest)
p189

Advice table representations of chess end-game knowledge.
MICHIE, D. (Edinburgh) BRATKO, I. (Ljubljana)
p194

Representation of phonetic and phonemic knowledge in a speech understanding system.
MORI, R DE (Torino) LAFACE, P.
p201

The last survey of Representation of Knowledge.
McDERMOTT, D. (Yale)
p206

Best-first interpretation of imperfect object contours.
NEUMANN, B. (Hamburg)
p222

A production rule account of errors in children's subtraction.
O'SHEA, T. (Edinburgh) YOUNG, R.M.
p229

A philosophy of 'wicked' problem implementation.
PARTRIDGE, D. (New Mexico)
p238

A humanlike approach for automatic theorem proving.
PASTRE, D. (Paris)
p248

A network representation for procedural knowledge.
PHILLIPS, B. (Illinois/CC)
p253

A computational study of a period of infant development.
PRAZDNY, K. (Essex)
p258

Parameterized region extraction for the description of moving objects.
RADIG, B. (Hamburg)
p263

Predictions and procedures in semantically-based grammar.
RITCHIE, G.D. (Edinburgh)
p273

What's in a social action for U?
ROSNER, M. (Essex)
p283

A conversational system which understands short stories.
SABAH, G. (Paris)
p293

What structure does a LISP program have - besides being a set of procedures?
SANDEWALL, E. (Linkoping)
p298

Representing Actions by State Logic.
SCHWIND, C.B. (Munchen)
p304

Representation and Control in Vision.
SLOMAN, A. (Sussex) OWEN, D. HINTON, G. BIRCH, F. O'GORMAN, F.
p309

Design for a plan hypothesizer.
SRIDHARAN, N.S. (Rutgers) SMITH, D.
p315

The frame shift mechanism in Natural Language.
STEELS, L. (MIT)
p325

Planning in uncertain environments through situation calculus.
STEPANKOVA, O. (Prague)
p.330

Completeness and soundness of the connection graph proof procedure.
STEPHAN, W. (Karlsruhe) SIEKMANN, J.
p.340

Common-sense Representation.
SUZMAN, F.L. (Edinburgh)
p.345

Toward greater generality in AI.
UHR, L. (Wisconsin) KOCHEN, M. (Michigan)
p.351

AI Programming Methodology.
WIELINGA, B.J. (Amsterdam)
p.355

Grammar discovery as data compression.
WOLFF, J.G. (Dundee)
p.375

Author Index.

ANZAI, Y. p1
BARSTOW, D.R. p15
BIBEL, W. p22
BIRCH, F. p28
BIRCH, F. p309
BOHM, H.P. p33
BOLEY, H. p35
BRADY, J.M. p42
BRATKO, I. p194
BRAZDIL, P. p46
BROOKS, M.J. p51
BROWN, F.M. p56
BROWN, F.M. p66
BUNDY, A. p71
CHAMPEAUX, D de p82
CLOCKSIN, W.F. p93
EICK, Ch p103
EISINGER, N. p105
FARGUES, J. p169
FISCHER, H.L. p33
GOOSSENS, D. p110
OJIDA, G. p115
HAHN, W.V. p122
HANNA, F.K. p132
HARWOOD, W.T. p132
HINTON, G. p309
HOEPPNER, W. p122
HOGGER, C.J. p138
HORN, B.K.P. p147
JAMESON, A. p122
KAHN, K.M. p164
KOCHEN, M. p351
KODRATOFF, Y. p169
LAFACE, P. p201
LUCKHAM, D.C. p176
LUGER, G. p71
MELLISH, C. p71
MERO, L. p189
MICHIE, D. p194
MORALES, J.J. p176
MORI, R DE p201
McDERMOTT, D. p206
NEUMANN, B. p222
O'GORMAN, F. p309
O'SHEA, T. p229
OWEN, D. p309
PALMER, M. p71
PARTRIDGE, D. p238
PASTRE, D. p248
PHILLIPS, B. p253
PRAZDNY, K. p258
RADIG, B. p263
RAULEFS, P. p103
RAULEFS, P. p33
RITCHIE, G.D. p273
ROSNER, M. p283
SABAH, G. p293
SANDEWALL, E. p298
SCHREIBER, J.F. p176
SCHWIND, C.B. p304
SCHWIND, C.B. p66
SIEKMANN, J. p105
SIEKMANN, J. p340
SLOMAN, A. p309
SMITH, D. p315
SOMALVICO, M. p115
SRIDHARAN, N.S. p315
STEELS, L. p325
STEPANKOVA, O. p330
STEPHAN, W. p340
SUZMAN, F.L. p345
UHR, L. p351
WAHLSTER, W. p122
WIELINGA, B.J. p355
WOLFF, J.G. p375
WRIGHTSON, G. p105
YOUNG, R.M. p229

HOW ONE LEARNS STRATEGIES: PROCESSES AND REPRESENTATION OF STRATEGY ACQUISITION[1]

Yuichiro Anzai

Department of Psychology
Carnegie-Mellon University

Schenley Park
Pittsburgh, Pa., 15213
U.S.A.

Abstract

A general theory is presented for the process and representation of strategy acquisition by human and artificial intelligence systems. The process is characterized by inductive generalization and organization of new rules through successive construction of new problem spaces. Various subtasks involved in the process can be naturally represented by subroutines in a hierarchical adaptive production system (HAPS), which is proposed as a representational scheme for strategy acquisition. HAPS has capabilities for subroutine calls and creating new productions in any subroutine. Results from strategy acquisition in the Tower of Hanoi puzzle by a HAPS program are given.

Descriptive terms: strategy acquisition, rule learning, production systems, problem solving.

To find a solution for a given problem, we need procedural knowledge that leads successfully to the solution. To find it efficiently, We need a ***strategy***, i.e., a set of procedural knowledge, that solves the problem systematically. Experts in a domain are well supplied with effective strategies, whereas novices know only inefficient ones. One who has never been exposed to the problem tries to induce rules that help problem solving. Experts and novices sometimes follow different strategies in complex problem solving tasks (Bhaskar & Simon, 1977; Simon & Simon, 1977). Efficient strategies may not only be taught, but ***acquired*** through

[1] This research was supported by Research Grant MH-07722 from the National Institute of Mental Health. The author is greatly indebted to Herbert Simon for helpful discussions and comments on this work.

experience. Then, the question is how? It is a crucial question, both for artificial intelligence and information processing psychology. Strategy acquisition is important in artificial intelligence systems because storing strategies in memory by hand requires ad hoc knowledge about efficient procedures to solve problems, which is not always available, and efficient strategies are also necessary to reduce execution time, which is limited practically. In cognitive psychology, there has been little work on acquisition processes of complex procedural knowledge, though it is one of the main issues in human problem solving.

This paper presents a general theoretical framework for the processes and representation of strategy acquisition in human and artificial intelligence systems. The framework described here was derived from an experimental analysis of strategy learning in solving the Tower of Hanoi puzzle. This well-known puzzle was used as an experimental vehicle because it can be solved by various different strategies (Simon, 1975) and is ***complex enough*** as a strategy acquisition task. Though the examples given below are taken solely from this task, the proposed theory is general, and believed to be generally extendable to strategy acquisition processes in complex problem solving tasks.

First, we present a brief review of the strategy learning process observed in an experiment. Second, we provide processing mechanisms for strategy acquisition in general terms. Third, we propose a representational scheme that realizes them. The scheme is a production system representation, but is characterized by an adaptive and hierarchical structure. Last, results from strategy acquisition by a computer in the Tower of Hanoi problem are summarized. They are followed by working examples extracted from the results[2].

Review of Experimental Analysis

Anzai & Simon (1977) used a subject who had never solved the Tower of Hanoi puzzle to observe the strategy acquisition process in this task. The subject tried to solve the five-disk problem four times from the initial configuration, including one failure and one trial that involved successive solutions of one-, two-, three-, four- and five-disk problems. Her first trial is basically a ***selective search*** for moves, but it already includes ***planning*** a sequence of future moves. Before solving the problem, she turned back to the initial disk configuration, and started the second trial. The strategy concealed in her first trial closely resembles the ***move-pattern strategy*** referred to by Simon (Simon, 1975). The second trial was devoted to finding good subgoals for a means-ends strategy. The third trial, in which the smaller problems were successively solved, led to the discovery of the concept of recursion of subgoals, and thus a ***recursive subgoal strategy***. In the fourth trial, the subject discovered the concept of moving a ***set*** of disks, which led to a strategy similar to the ***inner-directed goal recursion strategy*** in Simon's paper (1975).

[2] Because of the space limitation, computational results are summarized only briefly. They are described in detail elsewhere (Anzai, 1978).

It should be noted that the four trials were not independent. Each trial was executed using information collected during the previous trials. Efficient strategies like recursive ones did not appear abruptly, and even the first (incomplete) trial was necessary for discovery of them. Also more than one strategy was used in each trial. Various procedures, e.g., trial and error search, planning by looking ahead, constructing subproblems by generating subgoals, were mixed together in the trials.

General Mechanisms of Strategy Acquisition

A strategy is acquired through repeated solution of a given problem. A strategy acquisition process is not an independent process, but is embedded in the problem solving process. As suggested in the previous section, strategy acquisition gradually proceeds through reformulation and revision of *problem spaces* in which the problem solver, P, works not only for acquiring new strategies but also for solving the problem. A problem space consists of elements, each of which represents a knowledge state of P, and operators that transform one state to another. We hypothesize that strategy acquisition involves inductive generalization, and organization of rules useful for solving the problem through successive construction of new problem spaces. Previously learned rules are used extensively to generate new problem spaces, and the old problem spaces are not discarded but used whenever needed.

Based on these hypotheses, our theory asserts that strategy acquisition begins with construction of the initial problem space, proceeds to formation of *weak* strategies such as those for avoiding poor moves, and then to *strong* ones like means-ends and pattern-based strategies. We will explain below this general process of strategy acquisition.

Construction of the initial problem space

The sort of information that is available from the initially given task description is crucial for any strategy acquisition process. The information may include:

1. the initial knowledge state
2. initially available operators
3. constraints for application of operators
4. goals of the problem.

The information is important not because it directly generates a strategy, but because it makes it easy to construct the initial problem space S_1, and also to search efficiently in S_1. Once search becomes efficient, we can have, in the case of humans, some room in short term memory to consider generation of new rules, and in the case of computers, time necessary for rule generation.

Collection of bad instances

If P has never been exposed to the problem, S_1 is usually devoted to a heuristic search for better (or final) states. P needs some heuristics to decide when one state is better or worse than another. Typical examples are:

1. Returning to a previously visited state is bad.
2. Visiting a state that can be attained by another shorter route is bad.

Note that both of these rules are concerned with determining "bad" moves in S_1. This is not accidental. If P is unfamiliar with the problem, it is impossible for him to decide initially what is a "good" state. What P can do initially in S_1 is to collect "bad" examples of state transitions while trying to reach "good" states: "good" in the sense of the task-dependent measure chosen by P, or even, as above, in terms of a task-independent measure like "avoiding loops." Thus strategy acquisition will be directed initially toward finding strategies for avoiding poor moves.

Formation of rules for avoiding bad moves

A strategy is a set of rules. A rule can be represented by a ***production*** (Newell & Simon, 1972; Davis, Buchanan & Shortliffe, 1977):

Condition --> Action,

which expresses a causal relation that, if the Condition is met, then the Action is executed.

The action we are considering here is avoidance of a bad move observed during the search in S_1. What is an appropriate condition for a rule that executes this avoidance action? The current knowledge state consists of various kinds of information, most of which is irrelevant to the considered action. For example, a knowledge state generated while solving the Tower of Hanoi puzzle includes at least information about:

> the initial and final disk configurations, the current and past disk configurations, several imagined future disk configurations generated in the heuristic search, corresponding imaginary moves of disks, one or more past moves, some subgoals and move constraints extracted from the problem instructions, examples of bad moves, and so on.

Some subset of the knowledge that is present at the time a bad move has just been made can serve as the condition for the action of avoiding that move. Any of the standard approaches to mechanical induction could be used to select this subset

if all necessary information were explicitly present and if the induction did not call for excessive search. Since these conditions are seldom met, we propose as an alternative solution, a general semantic inference mechanism that is especially adapted to inferring conditions for productions[3].

Suppose that P has available in memory a problem-solving *script* (Schank & Abelson, 1977) that postulates semantic, causal relations between some subset of currently available information and intended actions. Then, in each successive context, as P is trying to solve the problem by moving disks, the script will define what aspects of the situation (e.g., the current disk configuration and one or more past moves) are relevant to reaching a new situation. These relevant aspects are extracted to define the condition side of a new production for avoiding an undesirable situation. Notice that the script is instantiated by the information available at the time when the situation to be avoided has been reached and noticed. Hence the induction requires no combinatorial search, and may utilize information not explicitly represented in the current knowledge state.

Collection of good instances and
formation of subgoal generation rules

As productions for avoiding bad moves are introduced, search will become more efficient and P will reach some state that is recognizably desirable. Once he attains such a state, he can generate subgoals effective for problem solving. Thus the problem solving process starts with looking ahead by a heuristic search in S_1, and moves to a means-ends type method. The former is inefficient but necessary for collecting bad instances. Rules generated from the instances are useful to avoid bad transfers in S_1, and thus to make the search process effective. With the aid of those rules, P is able to find good states, and generation of good subgoals follows.

Once various subgoals are generated, P tends to seek not for direct transfers among knowledge states, but for subgoals to be applied. A subgoal now becomes an operator that generates a *subproblem* whose final goal is that subgoal. The set of subproblems and subgoals constitutes a second problem space S_2, in which P tries to search by generating and applying subgoals to subproblems. Solution of a subproblem requires either transition in S_2, or search in S_1. S_1 does not disappear after S_2 is produced, but is used extensively whenever search in S_2 is unsuccessful. For example, a means-ends strategy may consist of several productions, most of which are for generating subgoals. While only some of subgoal-generating productions are acquired in the process of learning this strategy, the system has to explore in S_1 by using previously learned strategies whenever subgoal-generating productions do not provide proper subgoals.

If subgoals are discovered only within the context of the initial heuristic search in S_1, they are not necessarily well organized. Usually they are only roughly related, e.g., ordered in the order of application to states, and P needs more detailed

[3] This idea owes much to Langley (1977) and Neves (1977).

subgoals, and hence procedures to generate them, to construct a well-defined means-ends strategy. The mechanism proposed below is a feasible procedure for generating productions that produce subgoals. Suppose that we have a production which produces an already known subgoal G_1:

C_1 & C_2 & . . . & C_m --> generate G_1.

Then if conditions C_i, i = 1, . . . , m, are met, G_1 is generated. Now in the current situation, we may suppose that at least one C_k is not satisfied, otherwise G_1 is immediately generated. Let us store at this point of time the information:

(CONDITION : NOT(C_1 & . . . & C_m)).

Then let us wait until the search in S_1 attains G_1. At that moment, we may pick up the knowledge state which just preceded the state G_1. Let us call this state G_2, and store the information:

(ACTION : generate G_2).

Now, since the necessary condition and action information resides in memory, a production can be produced:

NOT(C_1 & . . . & C_m) --> generate G_2[4].

This algorithm is too general because the above production fires whenever G_1 cannot be generated directly, but we can replace its condition by the stronger one:

ALL-BUT-ONE-OF(C_1 & . . . & C_m) --> generate G_2.

In the strategy learning program for the Tower of Hanoi puzzle described later, this procedure successfully generated several subgoal generation rules that constitute a part of a means-ends strategy for the Tower of Hanoi puzzle.

Pattern discovery in subgoal structures

As search in S_2 is usually highly selective, generation of adequate rules for subgoal generation tends to provide directly an efficient means-ends strategy. During the search in S_2, it is not hard to collect sequences of good moves, i.e., correct sequences of subgoals. If pattern can be found in these sequences, a new strategy may be discovered, in which only a small number of operators are applied. There is needed here a general pattern induction algorithm like that discussed by Simon & Lea (1974). The only point of relevance here is that the amount of

[4] Note that, unlike productions generated using a problem solving script discussed earlier, the condition and the action of this production are not necessarily directly related in the time domain; information for the condition may be extracted from a knowledge state visited earlier.

information handled in the search in S_2 may be very small, so that pattern discovery can easily be tried.

A set of new subgoals, perhaps made from chunks of the old subgoals by an encoding mechanism, constitute operators for a third problem space S_3. If the subgoal sequence for attaining a final goal state in S_3 is very short, it would be more efficient to remember the sequence itself and apply it directly, than to apply operators backwards in a means-ends manner. In this case, ***working forward*** --- direct forward application of a small number of operators to the initial state in S_3 --- is highly efficient, comparing with the previously generated heuristic search and means-ends strategies, and can be seen often in human systems in everyday life (e.g., "strategy" for driving a car) or experts' problem solving (Simon & Simon, 1977).

Representation of Strategy Acquisition: Hierarchical Adaptive Production System

In strategy acquisition, P's tasks can be decomposed into various subtasks: selective search, collection of instances, rule formation, goal formation, interaction with the task environment, and so on.

As a representation of such systems, we propose here production systems with hierarchical structure: productions with subroutine calls. There have been strong arguments for the homogeneity of production system structure (Davis & King, 1977; Lenat & McDermott, 1977). Structural homogeneity is a powerful property for adding new productions and also for simulating human cognitive systems (Forgy & McDermott, 1977). On the other hand, homogeneity makes programs hard to write and understand (Moran, 1973), while subroutine structure facilitates understanding. The reader of a program with subroutines can attend to a part of the program without confusion. Furthermore, a strategy acquisition process, which is an amalgam of various different tasks, can readily be represented by a hierarchical production system.

It has been argued that structured production systems have a demerit in adding new rules. Production systems that learn new rules, i.e., ***adaptive production systems***, have usually been of a highly homogeneous character (Waterman, 1974; Buchanan & Mitchell, 1977). However, if a hierarchical production system had an ability to generate new productions within each subroutine by calling from it another subroutine for generating productions, then the system could be adaptive without any restriction.

An important problem for hierarchical production systems is whether working memory should be single or multiple. If each subroutine has its own working memory, i.e., the system has multiple working memory, a problem of inter-memory communication arises. Transmission of information among multiple memories is so frequent that it is not efficient to have a multiplicity of memories. Hence, we suppose here that a single working memory is shared by all subroutines.

Hierarchical adaptive production system (HAPS)

HAPS is an interpreter, written in LISP, for hierarchical adaptive production systems with a single working memory. The structure of HAPS is similar to traditional production systems except for a few characteristics of the syntax of actions. An action takes one of the following forms:

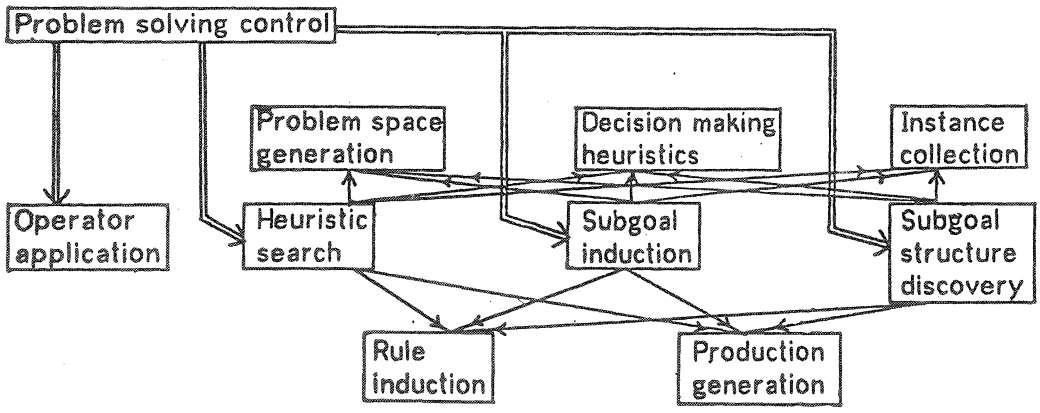
1. (DEP <primitive> <variable list>$_1$ . . . <variable list>$_k$)
2. (REM <primitive> <variable list>$_1$ . . . <variable list>$_l$)
3. (<function> <argument>$_1$ . . . <argument>$_m$; <productionname>$_1$ . . . <productionname>$_n$)
4. (PUTPRODUCTION (<primitive list>$_1$ <primitive list>$_2$))
5. (STOP).

<primitive> ::= any LISP list | <variable>
<variable> ::= any LISP atom beginning with $
<variable list> ::= non-existent | (<primitive>$_1$ <atom>$_1$ . . . <atom>$_i$: <atom>$_{i+1}$ . . . <atom>$_j$)
<atom> ::= any LISP atom
<function> ::= <atom>
<argument> ::= non-existent | <primitive> | <atom>
<productionname> ::= <atom>
<primitive list> ::= NIL | any list of LISP lists | <variable>.

Among the above actions, (DEP . . .) deposits <primitive> into working memory after all <variable list>s are executed: each <variable list> is said to be executed when <primitive>$_1$ is substituted by *any* of <atom>$_{i+1}$ . . . <atom>$_j$ excluding <atom>$_1$ . . . <atom>$_i$[5]. Thus the execution is essentially *nondeterministic*. (REM . . .) removes from working memory any <primitive> after executing all <variable list>s. (<function> . . .) is responsible for subroutine calls. <Function> is a subroutine name, and <argument>s substitute special variables denoted by $ARG1, $ARG2, . . . , appearing in the subroutine. They work as arguments for subroutine calls in the ordinary sense. The subroutine is defined by the names of productions; <productionname>s. As soon as this action is executed, control is transferred to those productions. Control is returned to the current subroutine as soon as the action (STOP) is executed in the called subroutine. Adaptivity is characterized by the action (PUTPRODUCTION . . .). As soon as it is executed, the production with <primitive list>$_1$ in its condition side and <primitive list>$_2$ in its action side is produced and put at the top of the current subroutine. Thus, using this action, a new production can be generated in any subroutine.

HAPS was designed as a tool for the study of strategy acquisition. However, its structure is general, and can be used for most production system programs described in the literature. The diagram below shows a general HAPS subroutine structure for a strategy acquisition process:

[5] For example, (ATTEND 3) or (ATTEND 5) will be deposited into working memory when the action (DEP (ATTEND $X) ($X 1 2 4 : 1 2 3 4 5)) is executed.



Each subroutine can be programmed as a set of productions. A search-based strategy may be acquired within the subroutine of "Heuristic search", whereas a means-ends type strategy can be learned within "Subgoal induction". Note that different strategies may be acquired in different subroutines, while utilizing the same resources of "Production generation" and "Rule induction".

Strategy Acquisition by Computer for the Tower of Hanoi Puzzle

A HAPS program was written for discovering strategies for the Tower of Hanoi puzzle. The current version consists of 164 HAPS productions before learning, which are classified into 25 subroutines. It has a structure generally following the scheme shown in the diagram in the previous section: The top routine is responsible for repeatedly solving the problem until the system is satisfied with the strategies it has acquired. The other subroutines are related either to: (1) heuristic search, (2) decision making heuristics for the search, (3) inducing conditions for avoiding bad moves, (4) inducing subgoal generation rules, (5) inducing patterns in subgoal structure, (6) generating new productions, and (7) manipulating actual moves and updating past move information. Mechanisms of condition inference and formation of subgoal generation rules described earlier are incorpolated in appropriate subroutines.

No facility for interpreting natural language problem instructions is incorporated in the current version. It assumes subgoals of taking each disk off the initial peg, and putting each disk on the goal peg. Another program, say UNDERSTAND (Hayes & Simon, 1974), may be able to extract these subgoals from the problem instructions. Also the program is given the rule for admissible moves: if a disk X is at the top of a peg P and is smaller than the top disk of another peg Q, or no disk is on Q, then X can be transferred from P to Q. Based on these kinds of information and the mechanisms described earlier, the program learned 13 new productions in six repeated solution trials. Productions fired 2948 times during the

process. It succeeded in generating three strategies successively: those similar to the move-pattern strategy, the goal recursion strategy, and the inner-directed goal recursion strategy in Simon (1975). Among the learned productions, seven were used for a part of the first strategy, four for a part of the second one, while the other two a part of the third. The program used the generated rules extensively to build a new problem space and then a new strategy as the theory indicated.

Examples

Let us illustrate by examples how the strategy acquisition process proceeds in the five-disk Tower of Hanoi problem. The examples below are excerpted from the computational results summarized above[6], and shown in parallel with the theory described earlier in this paper.

1. Construction of the initial problem space

(INITIAL (1 2 3 4 5) NIL NIL) is an element of the initial knowledge state. It implies that disks, 1, . . . , 5, are on Peg A and no disk is on Peg B or Peg C in the initial disk configuration. By assumption, Peg A denotes the initial peg, Peg C is the goal peg, and Peg B denotes the other peg. (GOAL-OFF 1 PEGA) is an example of initially available operators. It means that taking Disk 1 off Peg A is good to consider. (FINAL-GOAL NIL NIL (1 2 3 4 5)), indicating the final disk configuration, is the final goal of the problem. A legal move constraint is, for instance, given by:

(PEGS $A $B $C) (ATTEND $X) (SEARCH $X $Y) (*LESS (*TOP $X $A $B $C) (*TOP $Y $A $B $C)) --> (DEP (CAN (*TOP $X $A $B $C) $X $Y)).

The production implies that if the current disk configuration is (PEGS $A $B $C), the system is attending to the peg $X, searching for a move from $X to the peg $Y, and the top disk at $X is smaller than the top disk at $Y or no disk is on $Y, then the production deposits (CAN <top disk at $X> $X $Y) into working memory. $'s at the head of symbols indicate variables.

These kinds of a priori information defines the initial problem space, and the system begins heuristic search in this space.

2. Collection of bad instances

In the heuristic search, the system tries to collect bad instances. For example, returning to a previously visited state can be detected by the following production:

[6] A part of them is slightly simplified for easy understanding.

(IMAGE $A $B $C) (PIMAGE $A $B $C) --> (DEP (BAD)).

It means that if an imagined disk state has the same configuration as one of the configurations encountered in the past, then the system deposits (BAD) into working memory.

3. Formation of rules for avoiding bad moves

Suppose that the system found return to a previous state at the configuration (PEGS (3 4 5) (1) (2)), and at that time, the most recent moves are, in the order from the past, (2 PEGA PEGC) (1 PEGB PEGA) and (1 PEGA PEGB). Using a problem solving script implying that these operators might be responsible for the return to a previous state, the system creates a production instance of the form:

(PEGS (1 3 4 5) NIL (2)) (PASTPASTMOVE 2 PEGA PEGC) (PASTMOVE 1 PEGB PEGA) (ABSENT (TRIED 1 PEGA PEGB (1 3 4 5) NIL (2))) --> (DEP (TRIED 1 PEGA PEGB (1 3 4 5) NIL (2))).

A condition, (ABSENT <list>), is true if and only if <list> does not exist in working memory.

This production is generalized, and used as a part of the weak move-pattern strategy for avoiding poor moves. The resulting generalized production has the form:

(PEGS $A $B $C) (PASTPASTMOVE $X $P $Q) (PASTMOVE $Y $R $P) (ABSENT (TRIED $Y $P $R $A $B $C)) --> (DEP (TRIED $Y $P $R $A $B $C)).

4. Collection of good instances and
formation of subgoal generation rules

The system is supposed to have a production for deciding if a subgoal is applicable:

(PEGS $A $B $C) (GOAL $X $P $Q) (*EQUAL $X (*TOP $P $A $B $C)) (*LESS $X (*TOP $Q $A $B $C)) --> (DEP (CAN $X $P $Q)).

It implies that, if the current disk configuration is (PEGS $A $B $C), there exists a subgoal (GOAL $X $P $Q) in working memory, the disk $X is at the top of the peg $P at (PEGS $A $B $C), and $X is smaller than the top disk of the peg $Q or no disk is on $Q at (PEGS $A $B $C), then the system deposits (CAN $X $P $Q) into working memory.

Suppose that the current disk configuration is (PEGS (4 5) (1 2) (3)), and

(GOAL 4 PEGA PEGB) exists in working memory. Then, since (*LESS 4 (*TOP PEGB (4 5) (1 2) (3))) is false and all other conditions in the above production are true, the system deposits (COND (PEGS (4 5) (1 2) (3))), (COND (GOAL 4 PEGA PEGB)), (COND (*EQUAL 4 (*TOP PEGA (4 5) (1 2) (3)))) and (COND (*NOTLESS 4 (*TOP PEGB (4 5) (1 2) (3)))) into working memory. Then, after a while, when (*LESS 4 (*TOP PEGB $A $B $C)) becomes true, the production

(PEGS $A $B $C) (GOAL $X $P $Q) (PASTMOVE $Y $R $S) (COND (*NOTLESS $X (*TOP $P $D $E $F))) (*LESS $X (*TOP $P $A $B $C)) --> (DEP (ACTION (DEP (GOAL $Y $R $S)))) (DEP (COND (ABSENT (GOAL $Y $R $S))))

fires, and deposits (ACTION (DEP (GOAL 2 PEGB PEGC))) and (COND (ABSENT (GOAL 2 PEGB PEGC))) into working memory. It is the move (2 PEGB PEGC) that makes (*LESS 4 (*TOP PEGB $A $B $C)) true. Hence that move is copied from the PASTMOVE to a GOAL by firing the above production. The system creates a production from the CONDs and ACTION generated in the above process. The new production instance takes the form:

(PEGS (4 5) (1 2) (3)) (GOAL 4 PEGA PEGB) (*EQUAL 4 (*TOP PEGA (4 5) (1 2) (3))) (*NOTLESS 4 (*TOP PEGB (4 5) (1 2) (3))) (ABSENT (GOAL 2 PEGB PEGC)) --> (DEP (GOAL 2 PEGB PEGC)).

The system generalizes this instance by relating all elements appearing in the action side to elements in the condition side, and substituting variable symbols into specific values. Relating elements are done using a priori functional relations such as OTHER and NEXTSMALLER. For instance, at the configuration (PEGS (4 5) (1 2) (3)), 2 is NEXTSMALLER than 4 on Peg B. The generalized production of the above instance takes the form:

(PEGS $A $B $C) (GOAL $X $P $Q) (*EQUAL $X (*TOP $P $A $B $C)) (*NOTLESS $X (*TOP $Q $A $B $C)) (ABSENT (GOAL (*NEXTSMALLER $X $Q $A $B $C) $Q (*OTHER $P $Q : PEGA PEGB PEGC))) --> (DEP (GOAL (*NEXTSMALLER $X $Q $A $B $C) $Q (*OTHER $P $Q : PEGA PEGB PEGC))).

In the above production, (*NEXTSMALLER $X $Q $A $B $C) is a disk on the peg $Q and nextsmaller than the disk $X at the disk configuration of (PEGS $A $B $C). (*OTHER $P $Q : PEGA PEGB PEGC) is a peg which is other than $P or $Q among the pegs, Peg A, Peg B and Peg C. This production constitutes a part of the means-ends strategy acquired by the system.

5. Pattern discovery in subgoal structures

The experiment (Anzai & Simon, 1977) shows that discovery of pattern in subgoal structures in the Tower of Hanoi puzzle is related to visual perception. For example, if the current disk configuration is (PEGS (5) (4) (1 2 3)) and the system

retains (GOAL 5 PEGA PEGC) in working memory, the GOAL triggers perception of a *subpyramid* (1 2 3) on Peg C. Then the system creates a *subgoal stack* in working memory, and pushes down subgoals applied into the stack until the perceived subpyramid (1 2 3) is transferred to Peg B and the GOAL becomes applicable. The sequence of subgoals retained in the subgoal stack is then considered as a newly constructed subgoal; a subgoal of moving a *set* of disks. Using this kind of subgoals and generalizing production instances, the system creates a production:

(PEGS $A $B $C) (ABSENT (GOAL (*NEXTLESS $X) (*OTHER (*XTR3 (PGOAL $X $Q $R)) (*XTR4 (PGOAL $X $Q $R)) : PEGA PEGB PEGC) $R)) (ABSENT (GOAL $X $Q $R)) (ABSENT (GOAL (*NEXTSMALLER $X $Q $A $B $C) $Q (*OTHER (*XTR3 (PGOAL $X $Q $R)) (*XTR4 (PGOAL $X $Q $R)) : PEGA PEGB PEGC))) (PGOAL $X $Q $R) (*P-ON $X $Q $A $B $C) --> (DEP (GOAL (*NEXTLESS $X) (*OTHER (*XTR3 (PGOAL $X $Q $R)) (*XTR4 (PGOAL $X $Q $R)) : PEGA PEGB PEGC) $R)) (DEP (GOAL $X $Q $R)) (DEP (GOAL (*NEXTSMALLER $X $Q $A $B $C) $Q (*OTHER (*XTR3 (PGOAL $X $Q $R)) (*XTR4 (PGOAL $X $Q $R)) : PEGA PEGB PEGC))) (REM (PGOAL $X $Q $R)),

which is used as a part of the third strategy based on recursion of subpyramid moves. In the above production, (*NEXTLESS $X) is the disk one size smaller than the disk $X, *XTRi means extracting the i-th element in the argument, (*P-ON $X $Q $A $B $C) is a percept which is true if and only if Disks 1, . . . , $X are on the peg $Q at the configuration (PEGS $A $B $C), and PGOAL denotes a new subgoal of moving a subpyramid from one peg to another.

Conclusion

Computers can learn efficient strategies for complex problem solving tasks through experience. But, to do this, they need a basic theory and a representational scheme for strategy acquisition processes. This paper has presented both of these based on an experimental analysis of the Tower of Hanoi puzzle. The theory is based on the hypothesis that strategies are acquired through repeated solution of the same problem. During problem solving, new information is collected, induced and generalized to provide good rules that can be a part of a strategy.

Various different subtasks are involved in this process: a selective search, rule formation, subgoal generation, pattern discovery, and so on. In strategy acquisition by humans, these work as if they were coded as subroutines: they call each other, and control stays on one subtask only as long as that task needs to be executed. Corresponding to this characteristic, a hierarchical adaptive production system interpreter called HAPS was developed. Its structure is general enough to implement various traditional production system programs; however, it clearly reflects the hierarchical and adaptive features embedded in strategy acquisition processes. The results from strategy acquisition by a computer in the Tower of Hanoi puzzle were briefly mentioned. The computer succeeded in generating three very different kinds of strategies successively.

The study described here is still in an early stage, and we need to explore more details of the general mechanisms. Though the discussion here emphasizes the *processes* of strategy acquisition, finer algorithms for rule learning and modification are certainly necessary for elaboration of the theory. Still, perspectives of researches on strategy acquisition should go beyond studies on piecewise rule learning. They need an over-all view of acquisition processes such as is described in this paper.

References

Anzai, Y. Learning strategies by computer. submitted for publication, 1978.

Anzai, Y., & Simon, H. A. Strategy transformation in problem solving: A case study. CIP Working Paper 372, Dept. of Psychology, Carnegie-Mellon Univ., 1977.

Bhaskar, R., & Simon, H. A. Problem solving in semantically rich domains: An example from engineering thermodynamics. *Cognitive Science*, 1977, *1*, 192-215.

Buchanan, B. G., & Mitchell, T. M. Model-directed learning of production rules. Comp. Sci. Rep. STAN-CS-77-597, Stanford Univ., 1977.

Davis, R., Buchanan, B., & Shortliffe, E. Production rules as a representation for a knowledge-based consultation system. *Artificial Intelligence*, 1977, *8*, 15-45.

Davis, R., & King, J. An overview of production systems. In E. W. Elcock & D. Michie (Eds.), *Machine Intelligence 8*, Ellis Horwood, 1977, 300-332.

Forgy, C., & McDermott, J. OPS, a domain-independent production system language. Proc. 5th IJCAI, 1977, 933-939.

Hayes, J. R., & Simon, H. A. Understanding written problem instructions. In L. W. Gregg (Ed.), *Knowledge and Cognition*, Lawrence Erlbaum Assoc., 1974, 167-200.

Langley, P. W. BACON: A production system that discovers empirical laws. Proc. 5th IJCAI, 1977, 344.

Lenat, D. B., & McDermott, J. Less than general production system architectures. Proc. 5th IJCAI, 1977, 928-932.

Moran, T. P. The symbolic nature of visual imagery. Proc. 3rd IJCAI, 1973, 472-477.

Neves, D. Dissertation proposal. Dept. of Psychology, Carnegie-Mellon Univ., 1977.

Newell, A., & Simon, H. A. *Human Problem Solving*. Prentice-Hall, 1972.

Schank, R. C., & Abelson, R. P. *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum Assoc., 1977.

Simon, D. P., & Simon, H. A. Individual differences in solving physics problems. CIP Working Paper 342, Dept. of Psychology, Carnegie-Mellon Univ., 1977.

Simon, H. A. The functional equivalence of problem solving skills. *Cognitive Psychology*, 1975, *7*, 268-288.

Simon, H. A., & Lea, G. Problem solving and rule induction: A unified view. In L. W. Gregg (Ed.), *Knowledge and Cognition*, Lawrence and Erlbaum Assoc., 1974, 105-127.

Waterman, D. A. Adaptive production systems. CIP Working Paper 285, Dept. of Psychology, Carnegie-Mellon Univ., 1974.

EXPERIENCE WITH A REFINEMENT PARADIGM IN A KNOWLEDGE-BASED AUTOMATIC PROGRAMMING SYSTEM

David R. Barstow
Yale University
New Haven, Conn. 06520
U.S.A.

ABSTRACT: PECOS is a knowledge-based system that constructs concrete implementations of abstract algorithms in the domain of symbolic programming. The rules in PECOS's knowledge base deal explicitly with many details of symbolic programming, including intermediate-level constructs and certain design decisions. Programs are synthesized by gradually refining the original specification into a program in the target language. By applying the refinement rules in its knowledge base, PECOS constructs a refinement tree: the root is the abstract specification, each leaf is an alternative implementation, and each path is a different sequence of rule applications. Thus, PECOS can construct a variety of different implementations for the same abstract specification.

ACKNOWLEDGEMENTS: Discussions with Eugene Charniak, Cordell Green, Elaine Kant, and Drew McDermott have helped clarify the issues discussed here. This research was supported in part by ARPA under Contract MDA 903-76-C-0206.

1. INTRODUCTION

The simple observation that "Good programmers seem to know a lot about programming" suggests that one method for developing automatic programming systems would be to endow them with a lot of knowledge about programming; i.e., to apply to programming tasks the same knowledge-based approach that has worked well for other tasks (e.g., MYCIN for medical diagnosis [Shortliffe 1976]). With this motivation, a knowledge-based automatic programming system (called PECOS) has been designed and implemented [Barstow 1977b, 1977c]. PECOS's programming task is the construction of concrete implementations of abstract algorithms in the domain of symbolic computation. The specification language is essentially a high-level language for symbolic programming: its data structures are collections and mappings; its operations manipulate such structures (e.g., enumerate the elements of a set, compute the inverse of a mapping).

PECOS's rules (currently about 400) embody knowledge about a variety of implementation techniques, including several sorting and enumeration algorithms, as well as the representation of collections as linked lists and arrays (both ordered and unordered), and the representation of mappings as tables, sets of pairs, property list markings, and inverted mappings. In addition low-level details are also covered (e.g., the use of variables). As is common in such rule-based systems, the rules themselves deal primarily with specifics rather than with generalities. Several of PECOS's rules are given below (in English, for the sake of clarity):

* If a sequential collection is represented as an array subregion, an element may be inserted before a given location by expanding the allocation by 1, vacating the position before the location, and inserting the element into the vacated position.

* If the enumeration order is the same as the stored order of a sequential collection, the state of the enumeration may be saved as a location in the collection.

* If the domain of a stored mapping is a fixed set of known alternatives, the mapping may be represented as a record structure, with one field for each of the alternatives.

* In LISP, a distributed mapping may be represented using property list markings.

Despite the specificity of its rules, PECOS has been successfully applied to programs in a variety of domains. Among the programs that PECOS has constructed are the following:

CLASS (simple classification) From an abstract algorithm in terms of set operations, PECOS has produced several different implementations based on a variety of representations.

TF (simple concept formation) From an algorithm in terms of tags on descriptors, PECOS has constructed an implementation using a separate collection of descriptors for each tag.

SORT (sorting) PECOS has constructed several different insertion and selection sort programs using both linked list and array representations for the input and output sequences.

REACH (reachability in a directed graph) Given an algorithm in terms of a mapping of nodes to tags, PECOS has constructed several different implementations based on an inversion of the nodes-to-tags mapping. One of the implementations also involved converting the input representation of the graph into a more efficient form for internal use.

PRIMES (determining the primes less than a given integer) From an algorithm expressed in terms of operations on two sets, PECOS constructed several different implementations. The most efficient of these (linear) involved representing one set as a Boolean array and the other as a linked list. The others (all nonlinear) involved using the same representation for both sets.

2. AN EXAMPLE

PECOS constructs implementations by gradually refining the abstract algorithm into a concrete implementation in LISP. This is perhaps best clarified by an example. As the focus of this dicussion is the refinement paradigm, rather than PECOS's internal representations, the example will be presented in English; the interested reader is referred elsewhere for discussions of the representation scheme, as

well as the rule retrieval mechanism and associated pattern matcher [Barstow 1977a, 1977c]. For the moment, the reader is also urged to ignore questions about why a particular rule may have been applied when several are applicable in the same situation.

Although PECOS has been applied to fairly complex specifications, a simple example of data structure refinement is presented here in the interest of clarity. The abstract specification is given below:

< collection; elements: integers (1 to 100) >

(i.e., a collection whose elements are integers in the range 1 to 100.) The first rule to be applied reflects a particular decision about the kind of representation that will be used for the collection:

* A collection may be represented as a Boolean mapping.

This rule refines the data structure description into the following:

< mapping; domain: integers (1 to 100); range: Booleans >

The image of an integer under this mapping is "True" if the integer is a member of the collection and "False" if not. (There are, of course, many other ways to represent collections: one of PECOS's rules suggests representing a collection by storing all of the elements in some kind of unified structure; other rules suggest the use of linked lists and arrays to implement such stored structures.) Having chosen to represent the collection as a Boolean mapping, knowledge about representations for mappings in general can be used for this particular mapping. We can thus apply the following rule:

* A mapping may be stored in a single structure.

(An interesting alternative here is to use a distributed mapping, such as property list markings if the domain elements are atoms.) The next decision involves selecting the kind of structure that will be used. The following rule gives one alternative:

* A stored mapping may be represented as an association table.

(An alternative here is to use something like LISP's association list.) The next rule suggests a particular kind of table:

* If the keys are integers in a fixed range, an association table may be represented as an array.

Note that, up to this point, all of the rules represent a kind of general programming knowledge in the sense that they are applicable for a variety of different target languages. Eventually, of course, rules specific to LISP must be applied:

* An array may be represented directly in LISP.

(PECOS's target language is INTERLISP, which includes arrays as primitive data structures.) Thus, applying all of these rules yields a sequence of five steps, whose end result is the following description

of a concrete representation for the collection:

< LISP array; index: 1 to 100; type: Boolean >

The important point to notice is that the refinement process went through several steps, rather than going directly from "collection" to "LISP array of Booleans." Each step represented a separate decision, the addition of a small amount of detail to the abstract description.

3. THE REFINEMENT PARADIGM

Just as the notion of refinement has gained increasing importance in programming methodology (e.g., [Wirth 1971, Dahl et al 1972]), refinement has been a central organizing principle in PECOS's design.

3.1. Refinement Rules

Most of PECOS's knowledge about programming is represented in the form of "refinement rules," each describing a way to add detail to some particular part of a program description. (This is usually expressed by describing a way to implement an abstract construct in terms of slightly more refined constructs.) The amount of detail added by a refinement rule is relatively small. A deliberate effort has been made to identify and separate out the individual decisions that are often only implicit in much larger decisions. For example, refining a collection into a linked list of CONS cells takes five steps, each corresponding to a particular design decision.

Internally, refinement rules are represented as condition/action pairs, where the conditions are patterns of abstract concepts that may appear in a program description, and the actions are changes to be made in the description when applying the rule. Again, the details of PECOS's representation scheme are not particularly relevant to this discussion and will be omitted.

3.2. Refinement Sequences

As seen above, PECOS constructs implementations through gradual refinement: the original specification, described in terms of abstract concepts, is transformed through a sequence of descriptions using successively more refined concepts. The end result is the fully implemented program using the constructs of the target language. The process may be diagrammed as shown below:

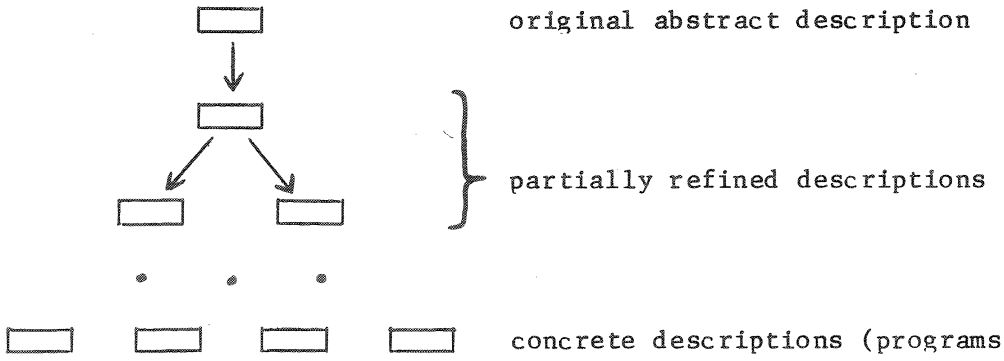


abstract partially refined concrete

The first description is the abstract specification; the last is the fully implemented program, and each intermediate description is slightly more concrete than the previous one. As illustrated earlier, each step in such a refinement sequence is produced by applying a single rule from PECOS's knowledge base. Since the amount of change involved in a single rule is not great, refinement steps generally also involve only small changes.

3.3. Refinement Trees

As noted above, PECOS's rules cover a variety of implementation techniques. This is reflected in the refinement paradigm when more than one rule is applicable at some step in a refinement sequence. Thus, PECOS actually constructs "refinement trees," as shown below:



The root of the tree is the original specification, the leaves are alternative implementations, and each path is a refinement sequence. Through the construction and exploration of such trees, then, PECOS can produce a variety of different implementations for the same abstract specification. For example, in the tree constructed for the PRIMES algorithm, one leaf was the linear implementation and the other leaves were nonlinear implementations.

An important feature of such trees is that the nodes (program descriptions) all represent "correct" programs (assuming correctness of the rules, of course). Each node represents a step in a path from the abstract specification to some concrete implementation of it. When paths cannot be completed (as happens occasionally), the cause is generally the absence of rules for dealing with a particular program description, rather than any inherent problem with the description itself. The use of refinement rules enables PECOS to proceed directly toward correct implementations, without any need for testing different combinations of primitive concstructs to see if they are correct. On the other hand, the possibilities for "inventing" a new use for particular operators in the target language are considerably reduced.

4. A SEARCH SPACE OF CORRECT PROGRAMS

Different implementations of abstract concepts are often appropriate under different situations (e.g., the two sets in PRIMES). Given some measure of "appropriateness" (perhaps an efficiency measure), PECOS's refinement tree provides a space that can be searched for the most appropriate implementation. While it is too early to make a general conclusion, experience with such trees suggests that they provide a relatively convenient way to deal with the variability and complexity that seem a necessary part of real-world programming.

Theoretically, PECOS could build the entire tree, compute the "appropriateness" measure of each leaf, and select the best. Since the number of reasonable implementations of a given abstract algorithm can be quite high (tens or even hundreds), however, techniques for

pruning the tree and limiting the search have a clear utility. Both analytic and heuristic techniques have been tried and PECOS's use of intermediate-level abstractions has proved helpful in both cases. Work on LIBRA, PSI's Efficiency Expert, has included the development of efficiency estimators for the intermediate-level constructs and the application of standard techniques (e.g., branch-and-bound) to the exploration of PECOS's refinement trees [Kant 1977]. Although most of the initial experimentation with PECOS has been done in an interactive mode (with the user selecting the path(s) to be followed), about a dozen choice-making heuristics were added to PECOS for the user's convenience. Experimental evidence indicates that even such a small set of heuristics is capable of dealing with about two-thirds of the choices encountered in a typical run. The ease with which these heuristics were added (less than a week's work) suggests that the use of several levels of abstraction is of significant value.

The biggest problem with PECOS's use of the refinement paradigm seems to be a potential for redundant work when several parts of a program are totally independent. This has turned out to be a smaller problem than originally expected, and some relatively simple techniques (e.g., postponing decisions as long as possible, finishing one part before working on another) have proved sufficient to deal with it. Should this problem become more significant in future development, recent work on "data dependencies" may prove quite useful [Doyle 1977].

5. DISCUSSION

PECOS's refinement paradigm, together with the knowledge base expressed as refinement rules, provide a search space in which the occurrences of "reasonable" solutions are relatively dense; most branches actually lead to valid solutions and the rest lead to dead-ends rather than invalid solutions. In part, this is due to the programming task that PECOS performs: the implementation of abstractly specified programs. The specifications are relatively algorithmic: programs in a very high-level language. (Although as the level increases, the specifications become less algorithmic; e.g., "Compute the inverse image of X under the mapping Y.") As a part of Yale's Knowledge-based Automatic Programming project, we are investigating the utility of the refinement paradigm when applied to even less algorithmic specifications, such as "Enumerate the nodes in a tree in order of their distance from the root" (i.e., breadth-first). While it is too early to do more than speculate, one possible outcome of this investigation may be the development of a refinement tree in which "completeable" solution paths are relatively sparse near the root and more dense near the leaves. If such is the case, then the likelihood of success may be the primary basis for making choices early in the process and the anticipated efficiency of the final solution may be more important in later stages.

The refinement paradigm may be compared with the debugging paradigm which has been developed for use in problem-solving systems (e.g., HACKER [Sussman 1975], NASL [McDermott 1976]). In the debugging paradigm, decisions are made with the expectation that any problems caused by the decision can be fixed up (debugged) if necessary. The principal difference between the two paradigms lies in the action

taken when something goes wrong: with refinement, the path is abandoned and another followed; with debugging, an attempt is made to fix the problem. As this may involve changing a decision previously made, comparing alternative branches is more complicated than in the refinement paradigm. Note, however, that the debugging paradigm has been applied to tasks of a somewhat different character: in most problem-solving situations, the task is to find any problem solution at all in a space that is quite sparse, rather than to find the best solution from a wide variety of alternatives. From these observations, the obvious conclusion is that the "ultimate" automtic programming system will employ both of these paradigms, in addition to others not yet developed.

REFERENCES

Barstow, D.R. "A knowledge base organization for rules about programming," SIGART: Pattern-directed Inference Systems, June 1977.

Barstow, D.R. "A knowledge-based system for automatic program construction," Fifth IJCAI, August 1977.

Barstow, D.R. "Automatic construction of algorithms and data structures using a knowledge base of programming rules," AIM-308, Stanford AI Lab, November 1977.

Dahl, O.-J., Dijkstra, E.W., and Hoare, C.A.R. Structured Programming, Academic Press, New York, 1972.

Doyle, J. "Truth maintenance systems for problem solving," TR-419, MIT AI Lab, 1977.

Green, R[illegible] "A summary of the PSI program synthesis system," Fifth IJCAI, August 1977.

Kant, E. "The selection of efficient implementations for a high-level language," SIGART/SIGPLAN: Artificial Intelligence and Programming Languages, August 1977.

McDermott, D.V. "Flexibility and efficiency in a computer program for designing circuits," TR-402, MIT AI Lab, December 1976.

Shortliffe, E.H. MYCIN: Computer-based Medical Consultation, American Elsevier, New York, 1976.

Sussman, G.J. A Computer Model of Skill Acquisition, American Elsevier, New York, 1975.

Wirth, N. "Program development by stepwise refinement," CACM, April 1971.

ON STRATEGIES FOR THE SYNTHESIS OF ALGORITHMS

W. Bibel

Technische Universität München, Germany

Abstract. A number of strategies for the construction of algorithms from a given input-output specification of a problem are presented. Their application is illustrated by a detailed synthesis of the usual MAXIMUM algorithm and Hoares FIND algorithm.

INTRODUCTION

Starting in 1974 the author has outlined a system for assisting the synthesis of programs [1,2,3]. The underlying idea is to design a mechanism, in which a given problem description is transformed in several steps into a form which can be efficiently executed by a machine. Moreover, this mechanism should be close to the human way of reasoning such that at each point the programmer can decide whether to interact or ask the machine to make its attempts and report on the results.

In the present note which is a condensed and slightly modified version of [4] we are exclusively concerned with the first major step of such a mechanism, which is regarded as the hardest one: how to transform a purely descriptive statement, which may be assumed to be formulated in predicate logic, into one which at least implicitly is of an algorithmic nature [2]. Several strategies are proposed, and their mechanical application in a standard sequence and the resulting transformation are given for two selected examples: the MAXimum problem and Hoare's FIND problem [9]. Very recently a considerable amount of independent but related work has been done like that reported in [5,6,7,8,11].

BASIC STRATEGIES

In the following examples the formula to be worked upon will be of the form

1.1. $\forall$input $\exists$output (input-condition $\rightarrow$ output-condition)

which allows to suppress the quantifiers in most cases (according to common practice). We will also drop those parts from the input-condition which are understood by the context or irrelevant for the solution. With these conventions the formula for the well-known maximum problem reads

1.2. $\forall S \exists m(S \neq \emptyset \rightarrow m \in S \wedge S \leq m)$

where $S \leq m$ is short for $\forall x(x \in S \rightarrow x \leq m)$. This description determines a function $MAX(S)=m$ but it does not specify how m could be determined for given S.

It is very natural in such a situation to make a guess, say m', for m and see how lucky we are. But we must somehow specify the domain where to take m' from. Obviously, it is reasonable to choose such a specification from those given output-conditions which specify m. Here we have two of them: $m \in S$ and $S \leq m$. Requiring both also for m' would not reduce the problem. So a choice between two alternatives is left.

The probabilities for the correctness of the guess are $p_1 = 1/|S|$ and $p_2 = 1/|\{m|S \leq m\}|$, resp. If we assume $p_2 < p_1$ which is true e.g. for a finite set of integers then obviously the first alternative is more restrictive, i.e. advantageous. Moreover, $m' \in S$ is a condition testable in one step, while $S \leq m'$ requires $|S|$ steps which additionally favours $m' \in S$. Let us put these considerations, which apparently are of a general nature and not specific to this example, into the following two strategies.

GUESS: Transform a problem of the form
$\forall$input $\exists y$ (input-condition $\rightarrow$ output-condition) to
$\forall$input $\forall y' \exists y$ (input-condition $\wedge$ domain-specification $\rightarrow$ output-condition $\wedge$ $(y \neq y' \veebar y = y')$)
where domain-specification is determined by the following substrategy (and $\veebar$ denotes exclusive disjunction).

DOMAIN: For a problem as in GUESS consider the set P of subsets of those conjuncts in output-condition which specify y. By conjuncting the elements of each of these subsets P becomes a set $\hat{P}$ of formulas. From $\hat{P}$ choose an element C as domain-specification such that C is minimal with respect to the following relation $<_k$.
For $C \in \hat{P}$ let $q_C := |\{y|C \text{ holds}\}|$, and s_C be the number of steps required to test whether C holds for given y. Then
$C <_k D$ iff $q_C \cdot s_C < k \cdot p_D \cdot s_D \wedge p_D \cdot s_D \not< k \cdot p_C \cdot s_C$.

For the purpose of this paper we will not specify in more detail what is meant by a "step" and how to determine q,s in general. The reader may assume that this information is provided to the mechanism from outside. k is a weight factor; $k=1$ is good enough for the examples in this paper.

Applying these two strategies to 1.2 gives

1.3. $\forall S \forall m' \exists m(S \neq \emptyset \wedge m' \in S \rightarrow m \in S \wedge S \leq m \wedge (m \neq m' \veebar m = m'))$

This new version is already of an implicit algorithmic na-

ture [2] which can be made explicit by an equivalence transformation. Such a transformation will be activated by the following strategy.

GET-G (goal-oriented equivalence transformation with goal G). Rewrite a given formula according to domain and goal dependent equivalence transformation rules until goal G is achieved.

As a matter of fact GET stands for a collection of strategies. In the following some instances are given.

GET-DNF: the goal is that the resulting formula is in disjunctive normal form (alternative cases).

Applying that to the conclusion in 1.3 yields

1.4. $S\neq\emptyset \wedge m'\in S \rightarrow (m\in S \wedge S\leq m \wedge m\neq m') \dot{\vee} (m\in S \wedge S\leq m \wedge m=m')$

GET-REC: the goal is to find some kind of recursion.

In some more detail, GET-REC would consider the first alternative in 1.4 (and not the second one since m is explicitly determined by input m' in it). There $m\in S \wedge m\neq m'$ can be rewritten to $m\in S\setminus m' \wedge m\neq m'$. Comparison with the original specification 1.2 indicates that the term S in this alternative should be substituted by $S\setminus m'$ to match the desired goal. Noting additionally that under the premise $m'\in S$ the inequality $m\neq m'$ in fact can be read as $m'<m$ the theorem prover would establish that 1.4 is equivalent with

1.5. $m'\in S \rightarrow ((S\setminus m'\neq\emptyset \rightarrow m\in S\setminus m' \wedge S\setminus m'\leq m) \wedge m'<m) \dot{\vee}$
$((S\neq\emptyset \rightarrow m\in S \wedge S\leq m) \wedge m=m')$

Using the function name this simply reads

1.6. $m'\in S \rightarrow (MAX(S)=MAX(S\setminus m') \wedge m'<MAX(S)) \dot{\vee} MAX(S)=m'$

GET-EP: the goal is that predicates become (more easily) evaluable.

Since $MAX(S\setminus m')$ is easier to be evaluated than $MAX(S)$ this strategy can be applied to 1.6 to yield

1.7. $m'\in S \rightarrow (m'<MAX(S\setminus m') \wedge MAX(S)=MAX(S\setminus m')) \dot{\vee} MAX(S)=m'$

A corresponding algorithmic version is

1.8. MAX(S) = begin choose m' such that m'∈S;
if $S\setminus m'\neq\emptyset \wedge m'<MAX(S\setminus m')$ then $MAX(S\setminus m')$
else m' end

We have used MAX to illustrate these strategies but the

point is that the pattern: "apply GUESS and DOMAIN, then GET-DNF, then GET-REC, finally GET-EP" (or extensions of it) may be successful also for other problems. Several other examples support this thesis. Two more (PART,FIND) are contained in the next section and a further one (LOCATE) which involves the more complicated data-structure of ordered arrays can be found in [4]. We do not know of any other sequence of strategies which would be successful for all these problems. E.g. the divide-and-conquer paradigm [8] provides an equally satisfactory synthesis for MAX but not for LOCATE and PART.

THE SUBPROBLEM MECHANISM

Upto now we have studied problems with only one existential quantifier. In this section we study a well-known problem [9] which is more complicated in that respect and introduce additional strategies for this type of problems. FIND asks for separating a given set S into the two subsets of those elements which are smaller/greater than the i-th element from S with respect to a given ordering. Formally the problem reads

2.1. $\forall S \forall i \exists S_1 \exists a \exists S_2$ FIND$(S,i) = (S_1,a,S_2)$ where

2.2. FIND$(S,i)=(S_1,i,S_2) \leftrightarrow$
$(1 \leq i \leq |S| \rightarrow S=S_1 \dot{\cup} a \dot{\cup} S_2 \wedge |S_1|=i-1 \wedge S_1<a \wedge a<S_2)$, and

2.3. $x \in A \dot{\cup} B \leftrightarrow (x \in A \vee x \in B) \wedge A \cap B = \emptyset$.

In this case it is necessary to study the dependency of the output-variables among each other. A special case of such a dependency is given when one of the variables can be explicitly expressed in terms of the remaining ones according to the output-conditions. Of course this variable can then be substituted.

RED#VAR: apply GET to output-condition with the goal of explicitly expressing one variable by the remaining ones and substitute that variable by the corresponding expression.

Since $S=S_1 \dot{\cup} a \dot{\cup} S_2 \leftrightarrow S_2=S \setminus (S_1 \cup a) \wedge a \in S \wedge S_1 \leq S \wedge a \notin S_1$ application of RED#VAR to 2.1 gives

2.4. $1 \leq i \leq |S| \rightarrow a \in S \wedge S_1 \leq S \wedge a \notin S_1 \wedge |S_1|=i-1 \wedge$
$S_1<a \wedge a<S \setminus (S_1 \cup a)$

In this new version a and S_1 are left as output-components and RED#VAR cannot be applied a second time. So the next task is to explore a dependency of S_1 on a or vice versa. Which one of these two alternatives should one try first? Again we use a general cardinality argument to get the following strategy, which simply says that one should try a guess on that variable first where the guess is more likely to be correct.

CHVAR: if there is a choice between two (or more) existentially quantified variables v_1, v_2 and the output-condition is of the form $F(v_1) \wedge G(v_2) \wedge H(v_1,v_2)$ where F and G does not contain v_2 and v_1, resp., and if $|\{v_1|F(v_1)holds\}| < |\{v_2|G(v_2)holds\}|$ then choose variable v_1 else v_2.

Since $|S| = |\{a|a\in S\}| < |\{S_1|S_1\subseteq S\}| = 2^{|S|}$ CHVAR chooses a in 2.4.

The user and/or a theorem prover has now to explore the desired dependency which will be formulated in the following strategy.

DEPEND: Given a problem of the form
$\forall input\ \exists v_1 \exists v_2 (input\text{-}condition \rightarrow F(v_1) \wedge C_1 \wedge \ldots \wedge C_n)$
where v_1 has been chosen by CHVAR and v_2 occurs in each C_i but not in $F(v_1)$, then determine an index set $\{i_1,\ldots,i_m\} \subseteq \{1,\ldots,n\}$ such that
$\forall input\ \forall v_1 \exists v_2 (input\text{-}condition \wedge F(v_1) \rightarrow C_{i_1} \wedge \ldots \wedge C_{i_m})$
is a theorem but for all $j \in \{1,\ldots,n\}\setminus\{i_1,\ldots,i_m\}$
$\forall input\ \forall v_1 \exists v_2 (input\text{-}condition \wedge F(v_1) \rightarrow C_{i_1} \wedge \ldots$
$\ldots \wedge C_{i_m} \wedge C_j)$ is not a theorem.

Note the maximality in the resulting dependency. The alert reader will see in the following how that contributes to the efficiency of the resulting algorithm.

Applying DEPEND to 2.4 gives the new problem.

2.5. $\forall S \forall i \forall a \exists S_1 (1\leq i\leq |S| \wedge a\in S \rightarrow S_1\subseteq S\setminus a \wedge S_1<a \wedge a<S\setminus(S_1 \cup a))$.

This defines a new problem which will get a name: PART. As a subproblem it has to be solved before returning to FIND. Its structure is like that for MAX except that S_1 is a set (while m in MAX was an element). But since a set can be displayed only by constructing its elements, the basic strategies simply require a straightforward adaption. So GUESS introduces a new quantification $\forall a_1'$ as before but "success or failure" reads "$a_1'\in S_1 \veebar a_1'\notin S_1$" (and not "$a_1'=S_1 \veebar a_1'\neq S_1$"). The cardinality argument in DOMAIN similarly adapted gives $a_1'\in S\setminus a$ as domain-specification. These, GET-DNF, -REC, and -EP applied to 2.5 yield

2.6. $a_1'\in S \rightarrow (PART(S\setminus a_1',i,a)=S_1\setminus a_1' \wedge a_1'<a) \veebar$
$(PART(S\setminus a_1',i,a)=S_1 \wedge a\leq a_1')$

in the same way as in the previous section which represents a solution for $PART(S,i,a) = S_1$. With this result RED#VAR can now be applied to 2.4 yielding

2.7. $1\leq i\leq |S| \rightarrow a\in S \wedge |PART(S,i,a)|=i-1$

In 2.7 only a is existentially quantified; so again we have obtained a structure like that discussed in the previ-

ous section. For lack of space we must leave it to the reader to verify that the standard sequence GUESS, DOMAIN, GET-DNF, -REC, and -EP results in

2.8. $a' \in S \wedge FIND(S,i)=(S_1,a) \wedge PART(S,i,a')=S_1' \rightarrow$
$(|S_1'|<i-1 \wedge (S_1 \setminus (S_1' \cup a'),a)=FIND(S \setminus (S_1' \cup a'),i-|S_1'|-1)) \vee$
$(i-1 \leq |S_1'| \wedge (S_1,a)=FIND(S_1',a))$

Two of the premisses just introduce abbreviations. A corresponding algorithmic form is

2.9. FIND(S,i)=
begin choose a' **such that** $a' \in S; S_1' \leftarrow PART(S,i,a')$;
if $|S_1'|<i-1$ **then**
$(S_1'',a,S_2) \leftarrow FIND(S \setminus (S_1' \cup a'),i-|S_1'|-1); (S_1' \cup a' \cup S_1'',a,S_2)$ **else**
if $i-1<|S_1'|$ **then**
$(S_1,a,S_2') \leftarrow FIND(S_1',i); (S_1,a,S \setminus (S_1 \cup a))$ **else**
$(S_1',a',S \setminus (S_1' \cup a))$
end

Transformation of the recursions in 2.9 into loops and implementation of the sets as arrays would give essentially Hoare's algorithm for FIND.

Acknowledgements. U. Furbach and J. Schreiber took part in the preparation of the previous version [4] of this paper by many discussions, helpful criticism, and valuable suggestions. I thank them for their stimulating cooperation.

REFERENCES

[1] Bibel,W., Prädikatives Programmieren, Lecture Notes Comp. Sc., 33, Springer-V., Berlin-Heidelberg-NewYork, 274-283, 1975.
[2] Bibel,W., Synthesis of strategic definitions and their control, report 7610, Techn. Univ. München, FB Mathematik, May 1976.
[3] Bibel,W., A uniform approach to programming, report 7633, Techn. Univ. München, FB Mathematik, Sept. 1976.
[4] Bibel,W., Furbach,U., Schreiber,J.F., Strategies for the synthesis of algorithms, Informatik-Fachberichte, 5, Springer-V., Berlin-Heidelberg-NewYork, 97-109, 1978.
[5] Brown,F.M., Tärnlund,S.A., Inductive reasoning in mathematics, report 30, Univ. of Edinburgh, Dept. Artificial Intelligence, 1977.
[6] Clark,K., Darlington,J., Algorithm analysis through synthesis, Imperial College London, Dept. Computing and Control, 1977.
[7] Derschowitz,N., Manna,Z., Inference rules for program annotations, AI-memo 303, Stanford Univ., March 1977.
[8] Green,C., Barstow,D., Program synthesis knowledge for efficient sorting, Stanford Univ., Artificial Intelligence Lab., 1977.
[9] Hoare,C.A.R., Algorithm 63 (Partition) and Algorithm 65 (Find), CACM, 4, 321-322, 1961.
[10] Kowalski,R., Algorithm = Logic + Control, Imperial College London, Dept. Computing and Control, 1976.
[11] Manna,Z., Waldinger,R., Synthesis: Dreams = Programs, Techn. Note 156, Stanford Research Institut, 1977.

A (SELF-ADAPTING) NETWORK FOR RECOGNITION OF VISUAL STRUCTURES.

By F. Birch and the POPEYE group,
Cognitive Studies Programme,
School of Social Sciences,
University of Sussex.

ABSTRACT: A system is presented that recognises emerging scene structures as a picture is interpreted. Emphasis is placed on using local gross features (which are cheap to compute) to guide the building of an interpretation. When this results in ambiguity, more global considerations can be used. Finer features (more expensive to compute) can also be utilised when present.

KEYTERMS: Perception, ambiguities, intermediate descriptions, self-adapting, alternative interpretations, 'Gestalt', recognition.

INTRODUCTION

The system that I describe called ASH, was built in the environment of the POPEYE project, Sloman and Hardy (1976) and Sloman (1978). POPEYE, inhabits a world consisting of capital letters formed from flat overlapping 'bar shaped' laminae. POPEYE finds the bars in such pictures and can pass them to ASH as potential letter strokes.

The problem then, is how to combine these strokes together, aggregating them into meaningful structures in terms of the letters that the system knows about. One sensible idea is to start from the clearest parts of the picture and trust that the interpretations that grow from these areas will help to disambiguate those less clear intervening regions. In this domain, a definition of 'clear picture part' could be any stroke with only one 'neighbour'.

Of course we need a good idea about where all the strokes in the picture are before we can locate the clear parts. As our pictures can be noisy, this may be impossible. It is also possible to construct pictures in which every stroke has more than one 'neighbour'. However we define 'clear part of the picture', we will run into problems of this kind.

A strategy has been developed that permits recognition of picture fragments as their member strokes are discovered. Segmentation is achieved by recognition of 'impossible' fragments. In the event of ambiguities occurring this can be postponed until clearer regions of the picture are discovered which allow the ambiguities to be resolved. If it becomes apparrent that not all the strokes are going to be discovered by normal methods, due to noise or occlusion, ASH can use recognised fragments to guide 'top-down' processing.

ASH uses a network of concepts to aggregate strokes together into letters and letter fragments. When ambiguities of segmentation occur, explicit alternatives are produced. Several alternatives can be held, only one of which is actively pursued at any particular time. This is necessary

because some pictures can have more than one 'good' global interpretation. Another reason is that the interpretation of some picture fragments can be ambiguous in isolation. In such cases it is useful to be able to maintain alternatives so that an interpretation which at some stage was 'abandoned' as being unlikely, and which comes back into favour doesn't have to be rebuilt from scratch.

ASH

I call the nodes of the network that ASH uses, ELEMENT's. Each element represents a concept which can be either a whole letter or some letter fragment. The elements are thus of two types, (i) the objects that the system knows about and (ii) intermediary concepts (i.e. parts of known objects). An element can belong to both categories, for example, an 'L' is part of an 'E'.

As strokes are presented to ASH, they are aggregated together into units called fragments. As each fragment grows it is associated with a succession of elements, hopefully ending at one which represents a whole letter. An element is a model of a 'letter fragment' and each fragment associated with it is an instance of that model. Fragments contain strokes, elements representations of those strokes. Below, I use the term, "stroke", for both, allowing context to disambiguate.

The network has an entry point for each basic type of fragment. Within POPEYE's world, these consist of elements representing vertical, horizontal and two sorts of diagonal strokes. Routes through the network consist of chains of pointers (ptr's) from element to element. The four start elements contain pointers to those elements representing two-stroke letter fragments. These pointers indicate how a new stroke can be joined to letter fragments containing just one stroke. Thus the "bottom end - right side" of the stroke in the element representing a vertical stroke points to the element that represents the letter L. The existence of the pointer says you can add a stroke here, and the pointed to element shows you what you then get.

There is a slot for a pointer at every position within an element where a stroke could feasibly be added according to the grammar of the domain: i.e. Strokes can be connected to other strokes at one of three points (the two ends and middle) along their length.

The ptr's have a dual role. In a clean picture where all the strokes are clearly depicted, they are found 'easily' by autonomous 'bottom-up' processes. Here the ptr's merely inform ASH how the strokes can be linked. In such a picture, it is not necessary to search 'top-down' for strokes. If there is evidence that some strokes haven't been uncovered, as when we have some incomplete letter fragments and no more strokes can be found easily, then it becomes necessary to search 'top-down'. Here the ptr's can tell POPEYE where to expect strokes, so that directed searching can occur.

This isn't sufficient on its own. We know where to look but not what to look for. There are several reasons why a stroke could be missing. It could be due to occulsion, or perhaps to a lot of noise in this region of the picture. Theories are needed about why a stroke

hasn't turned up to aid in searching for appropriate evidence. For steps in this direction see Paul (1977).

Growing the Net

If ASH knows about the letter E then the elements for single horizontal and vertical strokes might contain pointers to those elements shown below. I say "might" for although all these elements are in some sense known about by ASH, they need not all be represented explicitly as elements.



When ASH starts life the only elements that are represented explicitly are those representing (a) the letters ASH knows about and (b) the four basic stroke types. There are no links between elements. As pictures are processed, ASH uses complex and expensive searching and structure matching procedures to build those parts of the net that are needed. Useful pieces of information are thus re-represented in a more accessible form for future use.

Initially, the fields associated with all points on an element where a stroke might be added to a letter fragment represented by that element are blank. When it becomes necessary to know whether a stroke can be joined to a letter fragment the appropriate element and the field therein are consulted. If the field is blank (as it will be at first) the question will be resolved by an extended search for relevant sub-structures in known letters. Thereafter, ASH will find either a pointer to the element that represents letter fragments that would be created if a stroke was added in that place, or an indicator that adding a stroke here would result in the formation of a MIF, (minimal impossible fragment).

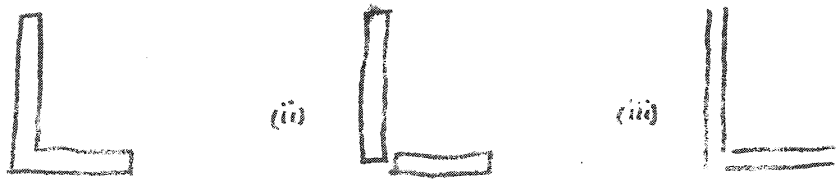
One of the main reasons for doing things in this fashion, is that if a fair number of objects are to be known by the system then there will be a large number of possible elements, and an even larger number of MIF's. If a world is friendly, then most of these MIF's will probably never be encountered, whereas a few will frequently turn up. If we were dealing with English script formed from capital letters then the MIF's shown below in (i) caused by the juxtaposition of a T with an H would be encountered frequently whereas we might well never encounter that shown in (ii). It is thus not sensible to compute all possible MIF's in advance.



A Simple Example

Consider (i) below. Suppose POPEYE finds the vertical bar first as this is slightly larger. POPEYE passes the bar-skeleton of the hypothesised bar to ASH, who attempts to absorb it into the current interpretation. In this case the interpretation is empty so a 'fragment record' is created for the new stroke which is then linked, as an instance, to the vertical stroke element. ASH interprets the single 'bar' as a vertical stroke. The horizontal bar is then found. ASH asks, are any other known skeletons 'significantly close' to this one?

(i) (ii) (iii)



"Significantly close" is loosely defined. Almost all skeletons which should be combined, together with many that shouldn't, will be considered. The latter are thrown out by a combination of factors including the detection of impossible fragments, and the process of building alternative interpretations which allows more global considerations to be invoked. Those bar-skeleton-joins that aren't made due to the definition being too restrictive should be picked up later, in top-down mode.

The second stroke is significantly close to the first so ASH looks at the element with which the first stroke is associated. The relationship between the old stroke and the new stroke, is used to decide which part of the element to address. If doing this results in a YES, then the fragments association with the vertical stroke element is broken. The horizontal stroke is added to it, and it becomes associated with the element 'L'. The fragments two member strokes become associated with their representations in 'L'.

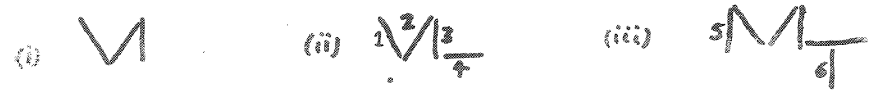
ASH can ask POPEYE to look carefully at the junctions between the two bars and build a precise description of their relationship. In this case this isn't necessary, for there are no global considerations conflicting with the desire to 'join' these two strokes. (ii) and (iii) above would probably, also be interpreted as L's by most people expecting capital letters. In some sense a GESTALT can be said to have occurred with a whole emerging prior to connectivity details. These can then be discovered by reference to that whole.

A picture like (iii) above suffices to produce the idea of an 'L' when POPEYE is combined with ASH. Other processes can then distinguish between this and alternative cases such as (i) and (ii).

Using Global Context

Consider the figures below.

(i) (ii) (iii)



(i) is ambiguous. ASH can resolve the ambiguity by using context. Suppose ASH is presented with the strokes of (ii), in the order 1,2,3,4. 1,2 and 3 are aggregated together as they turn up and interpreted as part of an M. When 4 appears a new interpretation is built, for it cannot be joined to anything in the current one without generating a MIF. It can however be joined to 3 if the idea that 3 forms part of an M is abandoned. The new interpretation, V + L, is preferred to the interpretation, 'part of an M + a horizontal stroke' as it contains more complete objects. If (ii) was really part of (iii) the discovery of 5 and 6 would cause a swing back to the original interpretation accomplished by continuing the 'part of M + horizontal stroke' interpretation that was rejected earlier but saved.

If a picture contains sufficient detail, then ASH can also utilise this source. For example, if the letters were formed from bars (as in POPEYE's domain), then when stroke 4 is discovered we can decide whether 4 should be joined to 3 and the 'part of an M' interpretation abandoned by examining in detail the junctions formed by stroke 2 with 3, and 3 with 4. We can thus decide whether we have something like (ii) or something like (iii).

Usually/
the process of building a new interpretation, called REFRAGMENTATION, starts from a stroke which for some reason can't be interpreted as anything but a single stroke in the current interpretation. There are several possible reasons. E.g. we have been building the 'wrong' interpretation or the problem stroke should be joined to strokes yet to be discovered or the problem stroke is spurious.

Refragmentation is the building of a new interpretation as if the problem stroke was the first to be discovered, and as if the other already known strokes were discovered roughly in the order of how close they are to it. An alternative interpretation can thus be built in which the problem stroke plays a more significant role than simply 'single stroke'. This may of course be at the expense of several other strokes which find themselves relegated to a similar role.

In certain pictures 'excessive' refragmentation is possible. (Examples on board.) This can be controlled in two ways. Firstly by examining the fine detail of the picture if available, and secondly by 'suspending' a stroke if it plays a prominent part in several refragmentations. The rationale being that other letter fragments will absorb those surrounding strokes which are causing the confusion leaving only those which should be combined with the sought after stroke.

SEE BOARD FOR (i) examples,
(ii) bibliography,
(iii) set of rules which describe how interpretations are constructed in ASH.

ACKNOWLEDGMENTS

I would like to thank Dr Aaron Sloman for his constructive criticism. This work was supported by a S.R.C. studentship and S.R.C. research grant ref BRG 8688-7.

REFERENCES

Paul, J.L. (1977) 'An image interpretation system'. Sussex University Phd Thesis.

Sloman, A. and Hardy, S. (1976) 'Giving a computer Gestalt experiences'. In proceedings AISB summer conference, Edinburgh 1976.

Sloman, A. et al. (1978) 'Representation and Control in Vision'. In proceedings AISB/GI summer conference, Hamburg 1978.

DIALOGUES IN ACTOR NETS

H. P. BÖHM, H. L. FISCHER and P. RAULEFS

Institut für Informatik III
Universität Bonn
D-5300 Bonn 1, West Germany

Institut für Informatik I
Universität Karlsruhe
D-7500 Karlsruhe 1, West Germany

Abstract

The actor society representation of knowledge is introduced. In a self-organizing society of actors, knowledge is *distributed* s.t. each individual actor has strictly local information. Although each actor exclusively tries to achieve its own private goal, it exhibits a communication behavior making the entire society form a well-orgaized system having global properties no individual actor is aware of. Actor societies are realized as networks of CSSA-actors /1/.

Major innovations introduced in this work are:

1. CSSA-actors realize *operational abstractions:* An operational abstraction is an object which is completely determined by the way it responds to *operation requests*. All data and control structures can be uniformly described as retracts of operational abstractions. Hence, any data/control type can be "programmed" as a CSSA-script s.t. all actors generated from such a script are objects belonging to that type.
2. The *facetting mechanism* of CSSA-actors allows to realize data/control types by scripts in a representation-independent way which is (a) very difficult resp. (b) impossible to achieve by (a) algebraic specification techniques resp. (b) languages using class-constructs, such as ALPHARD and CLU.
3. Dynamically administrable *access-rights* allow to manipulate access restrictions in an extremely flexible way while retaining a tight discipline.
4. Access-rights, and correctness specifications incorporated in patterns and messages for evaluation in activation and creation pattern matches allow to combine and extend known validation and verification techniques.

When developing actor societies, CSSA supports a particular style of programming which we claim to contribute significantly to knowledge representation and programming methodology. Our paper gives an informal introduction to the central language concepts of CSSA, and by carefully working through a detailed example we provide evidence for our claim.

/1/ H. P. Böhm, H. L. Fischer, P. Raulefs. CSSA: Language concepts and programming methodology. Proc. Symp. on Artificial Intelligence and Programming Languages, Rochester, 1977. ACM SIGPLAN Notices:12/ SIGART Newsletter:64, pp. 100 - 109.

THE FIT APPROACH TO AI LANGUAGES

Harold Boley

Fachbereich Informatik, Universität Hamburg, D-2 Hamburg 13, West Germany

Abstract: *An overview of the AI language FIT is given, which attempts to unify, generalize, and advance a number of programming language concepts. Its central entities are fitters, a unification of functions, patterns, and variables, especially of predicate functions and typed variables. The latter are also viewed as new kinds of demons. Fitter definitions are assignments of bodies to complex variables viewable both as function application forms and as consequent theorem invocation patterns. Fitter computation and look-up are thus unified to fitter valuation.*

Descriptive terms: *AI languages, FIT, fitters, fitments, pattern matching, complex variables, valuation, consequent theorems, types, demons.*

INTRODUCTION

In my opinion many AI languages since PLANNER (Hewitt 1969), developed in the context of rather strong application needs, have been too implementation-minded and ad hoc. This, I think, is the crucial reason why the enthusiasm of the early Seventies (Bobrow & Raphael 1974) now has yielded to a certain disillusion as exemplified by a recent panel discussion entitled "The Bobrow-Raphael Survey Revisited" (McQueen 1978).
But this is a natural developmental crisis, which can be overcome by now - after a phase of experimental praxis - concentrating on theoretical concepts which can form the basis of 'third generation' AI languages.
PLASMA (Hewitt 1975), KRL (Bobrow & Winograd 1977), EVIL (Bornat & Wielinga 1976), FIT, and other languages may be regarded as - rather different - attempts to do this. Of course the ultimate motivation for such languages are still practical needs (a partial LISP implementation of FIT is in progress, operations on directed recursive labelnode hypergraphs (Boley 1977) have been formulated in FIT, and a FIT/DRLH based expert system is planned).
In the following only basic ideas of FIT can be presented. Some details may be found in (Boley 1978).

DESIGN PRINCIPLES, UNIFICATIONS, SUBORDINATIONS, AND INNOVATIONS

Let's now discuss a number of FIT's design principles P, unifications U, subordinations S, and innovations I in an order which permits a convenient presentation of important FIT concepts.

P1: Very-high-levelness

FIT is a very-high-level language in that it emphasizes expressive power, even if this affects computational efficency. The reason for this is our belief that - especially in the long run - the "complexity barrier" (Winograd 1975) is more a *mental* barrier of the programmer than a *computational* barrier of the computer. So we try to widen the 'programmer complexity bottleneck' by allowing him to express his ideas in a non-restrictive way.

P2: Orthogonalization of concepts, eg through unifications

Few AI languages meet present standards of (non-AI) programming language definitions. At least with respect to the orthogonality demand this has clear disadvantages (eg lack of transparency, obstacle to formal semantics). So FIT orthogonalizes some AI language concepts, eg by unifying certain non-orthogonal ones. Thus such seemingly typical AI language ideas as pattern-directed invocation are synthesized with traditional ones.

P3: Permanence principle

Simple things can be expressed in a simple way using subsets of FIT, and only complex things have to be expressed using FIT constructs in their full generality. The defaults are set in such a way that you only see this generality when you need it. This permanence of FIT constructs from simplicity to complexity makes FIT easy to understand for beginners.

P4: Non-animistic view of data structures

While PLASMA's actors animate data structures to active objects, FIT's fitters leave them as passive ones: as generalizations of LISP functions, fitters are viewed as *working on* data structures. Still FIT allows the attachment of knowledge to specific data items: we simply use fitter definitions having these data items as constant parameters.

P5: Structured control structures

Many AI languages are experimenting with very complex control structures (Bornat/Wielinga 1976), whereas FIT does not, because these tend to make programs unstructured and very hard to understand (unrestricted gotos are still easier to understand than unrestricted continuations), so that P1 is violated. FIT presently uses LISP's hierarchical control structures, augmented by various kinds of fail-backs upwards in the hierarchy, eg failU (fail-unknown) for undefined fitters and failF (fail-false) for fitters which as LISP functions would return F.

U1: Complete integration of pattern matching and LISP functions

The concepts of PLANNERlike pattern matching and LISPlike functions have crystallized as two of the most important AI language concepts. These concepts have been separated up to now or, as Hewitt put it (Hewitt 1977), the bad readability of LISP is not due to its lack of iteration but to its lack of pattern matching. Since a simple mixture of LISP and a PLANNERlike language as eg done in QLISP (Wilber 1976) is no satisfactory solution (see P2), in FIT we have completely integrated these concepts by means of a generalizing unification of patterns and functions to fitters.

U2: Unification of predicates and typed variables

In many programming languages types are used - in form of mandatory variable declarations - for compile-time checks, which in case of violations yield unrecoverable errors and are completely separated from predicate functions. In FIT typed variables can be used - in form of optional definitions - for dynamic run-time checks, so that violations like false predicates yield failF usable as control structures to direct the computation. Typed variables and predicates only differ in a ?-prefix, which is another part of FIT's central unification of patterns, functions, and variables to fitters.

I1: Complex variables

In FIT not just atoms but arbitrarily complex expressions - possibly containing variables - are allowed as (complex) variables.

U3: Unification of look-up and computation to valuation

In other programming languages (even inside LISP's EVAL) the operations of looking-up the value of a variable and of computing the value of a function application are strictly distinguished. FIT however unifies both concepts to the valuation concept, by regarding a function application computation as little more than repeated look-ups of complex variables' values. FIT computations thereby get close to formal systems' derivations (Salomaa 1973), especially production systems' (Davis & King 1975).
In a sense the only programming activity in FIT consists in assigning values to variables (which are normally viewed as data), so that FIT may be regarded as a declarative rather than a procedural representation-language (Boley 1976).

I2: Non-deterministic variables

Assignments to non-deterministic variables (marked by a V prefix) conserve their possible former values. The valuation of such variables becomes non-deterministic, ie one of its values which in the sequel doesn't lead to a failU is finally taken.

U4: Unification of name-directedness and pattern-directedness

PLANNERlike languages add pattern-directed function invocation ('to invoke a function specify its invocation pattern') to name-directed function invocation ('to invoke a function specify its name together with possible arguments'). In FIT however both ordinary (predicate) functions and consequent theorems are defined by assigning bodies to complex (possibly non-deterministic) variables which can be viewed both as function application forms and as invocation patterns. Instances of these complex variables can be viewed therefore both as name-directed and as pattern-directed invocations of their body values. Explicit GOAL statements become superfluous this way and only anonymous functions need LAMBDAs. Since name-directedness is used in ordinary computation and pattern-directedness is used in deduction, the above is also the way FIT unifies computation and deduction to valuation.

I3: Distributed fitter definitions

A fitter in FIT may be defined in a distributed manner like a set of recursion equations, because the valuation rule 'use specific knowledge first' guarantees that the correct part of such a definition is always used. Distributability of definitions has the following advantages: additivity (incomplete fitters work and may be refined), modularity as in production systems, automatic handling of exceptions in the "generalization hierarchy" (Bobrow & Winograd 1977), speed gains thru (redundant) explicit definition parts for frequent cases, simplification of certain function definitions (eg the COND of a LISP factorial definition becomes implicit), non-procedurality (you can say *what's* instead of *how's*).

S1: Assertions as special predicate functions

While in PLANNERlike languages the data base is divided into a data base of assertions (propositions) and a data base of theorems (rules), in FIT assertions are trivial (namely constantly true) parts of distributed predicate function definitions, ie they simply are special consequent theorems, so that we have one integrated data base and FIT is a rule/proposition integrating representation-language (Boley 1976).

I4: Demon variables

Demon variables (marked by a W prefix) like non-deterministic ones conserve possible former values, but on valuation the side-effects of *all*

their values are taken, which is what we need for the definition of demons.

I5: One-eyed and many-eyed demons

FIT regards one-eyed demons as demon variables assigned to bodies. All one-eyed demons whose variable matches some instance are invoked when this instance is used and yield side-effects. In many-eyed demons many demon variables are assigned to a common body. They are invoked only after all their variables ('eyes') have matched ('seen') an instance.

S2: Types as one-eyed demons

For objects belonging to more than one superset we have to allow variables to be typed more than once, where all the type restrictions have to be fulfilled conjunctively. Such types are defined as one-eyed demons and not as ordinary predicate functions.

S3: Generalized types as many-eyed demons

While the previous types all correspond to unary predicates over variables, FIT also allows generalized types corresponding to n-ary predicates ensuring that arbitrary n-ary relationships always have to be fulfilled between n variables. They are defined as many-eyed demons.

S4: Antecedent theorems as demons

Ordinary antecedent theorems in FIT become special one-eyed demons invoked by assertion instances to make other assertions. Generalized antecedent theorems need many assertions for invocation and are defined analogously by means of many-eyed demons.

FITMENTS AS UNIFICATION OF APPLICATIONS, MATCHES, AND ASSIGNMENTS

All FIT expressions are atoms as in LISP or fitments as shown in Fig.1.

FITMENT	fits	*FITTERS*	to FITTEES	(FR_1 .. FR_m _ FE_1 .. FE_n)
	generalizing ↑ unification			for m=1: (FR FE_1 .. FE_n) generalizing ↑ unification
APPLICATION	applies	*FUNCTION*	to ARGUMENTS	(*FUNCTION* ARG_1 .. ARG_n)
MATCH	matches	*PATTERN*	to INSTANCE	(MATCH *PATTERN* INSTANCE)
ASSIGNMENT	assigns	*VARIABLE*	to VALUE	(SETQ *VARIABLE* VALUE)
(a) Verbal description				(b) Syntax

Fig.1. Fitments as generalizing unifications.

For example the APPLICATION (*SQRT* 9) ⇒ 3 , where '⇒' denotes the 'yields' relation, the MATCH (MATCH "(*A ?Y ?Z*) "(A 9 B)) ⇒ T, <Y ⇒ 9, <Z ⇒ B, or in FIT ("(*A ?Y ?Z*) "(A 9 B)) ⇒ (A 9 B), <Y ⇒ 9, <Z ⇒ B, where '"' is the quote prefix, '?' specifies that one value is to be *shoved* into a variable, whereas '<' specifies values to be *pulled* out of a variable (Winston 1977), and the ASSIGNMENT (SETQ *Z* "B), or in FIT (*?Z* B), where we exploit FIT's inverse quote mode for atoms, can be unified to the FITMENT ("(*A SQRT ?Z*) "(A 9 B)) ⇒ (A 3 B), <Z ⇒ B.
Note that the pattern (A ?Y ?Z) accepts the same set of lists as the FIT predicate function (?"(ATRIPLE ?L) "(AND (EQ (CAR <L) A) (NULL (CDDDR <L)))). Similarly the general fitter (A SQRT ?Z) returns the same values as the general function (?"(ASQRTTRIPLE ?ATRIPLE) "(LIST A (SQRT (CADR <ATRIPLE)) (CADDR <ATRIPLE))), where ?ATRIPLE is a typed variable which can only be bound to ATRIPLE lists: (?ATRIPLE "(C 9 B)) ⇒ failF but (?ATRIPLE "(A 9 B)) ⇒ (A 9 B), <ATRIPLE ⇒ (A 9 B), since FIT's typed variables and predicates replace T results by their fittees, here by (A 9 B). Similarly we allow (?ASQRTTRIPLE "(A 9 B)) ⇒ (A 3 B), <ASQRTTRIPLE ⇒ (A 9 B), where we use

the general function ASQRTTRIPLE as a 'computing variable' ?ASQRTTRIPLE. Fig.2 relates the above examples by means of a "similarity network" (Winston 1977) of the more detailed subconcepts (excluding constants) which we unified to the fitter concept.

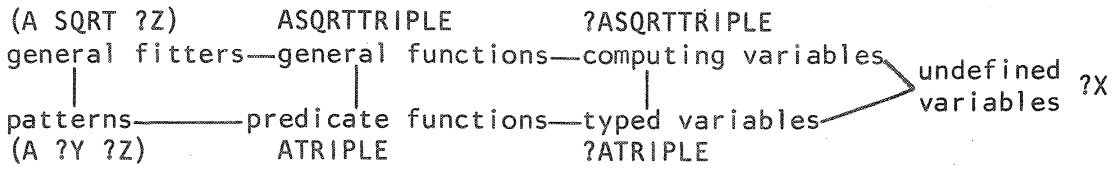


Fig.2. Similarity network relating fitter subconcepts.

While the above example fitments have the form (FR FE), the fitments (PLUS 5 9) ⇒ 14 and (>X 7 B A) ⇒ 7 B A, <X ⇒ 7 B A have the form (FR FE_1 .. FE_n). Note that '>' is like '?' but shoves arbitrarily many values. The fitments (CDR PLUS _ "(A B C) 5 9) ⇒ (B C) 14 and (7 >X _ 7 B A) ⇒ 7 B A, <X ⇒ B A have the general form (FR_1 .. FR_m _ FE_1 .. FE_n).

VALUATION FROM LOOK-UP TO COMPUTATION

We now exemplify U3 by gradually proceeding from look-up of atomic variables' values to computation of function applications.
The fitter of the fitment in Fig.3(α) is an atomic variable >SQUAREIGHT, whereas the fitter of Fig.3(β) is a complex variable >"(SQUARE 8). Fig.3 (a) looks-up the value of the atom, (b) analogously the value of the fitment. The complex variable >"(SQUARE ?X) of Fig.3(γ) itself contains the variable ?X, which 'parametrizes' the assignment (γ). We can (indirectly) pull the values of all instances of (SQUARE ?X), which are the corresponding instantiations of (TIMES <X <X), as exemplified by Fig.3(c). Since pull prefixes may be nested and (TIMES 9 9) may be regarded as a 'primitive variable' preassigned to 81, (γ) also permits (d). Finally, since for fitments without pull and quote prefixes as many pulls as necessary for obtaining a constant are assumed, (γ) permits (e). Of course (e) is normally viewed as the computation of a one-argument function SQUARE applied to the actual argument 9, though it was developed from the clear look-up example (a).
Therefore we regard (a) to (e) as varieties of the single concept of valuation. As shown in Fig.3 this concept contains four subconcepts, created by the orthogonal direct/indirect and fixed-depth/unfixed-depth dimensions.

	direct (atom)	direct (fitment)	indirect (fitment)
definition	(α) (>SQUAREIGHT 64)	(β) (>"(SQUARE 8) 64)	(γ) (>"(SQUARE ?X) "(TIMES <X <X))
fixed-depth 1 valuation	(a) <SQUAREIGHT ⇒ 64	(b) <"(SQUARE 8) ⇒ 64	(c) <"(SQUARE 9) ⇒ (TIMES 9 9)
fixed-depth 2 valuation	<<SQUAREIGHT ⇒ <64 ⇒ failU	<<"(SQUARE 8) ⇒ <64 ⇒ failU	(d) <<"(SQUARE 9) ⇒ <(TIMES 9 9) ⇒ 81
unfixed-depth valuation	impossible	(SQUARE 8) ⇒ 64	(e) (SQUARE 9) ⇒ (TIMES 9 9) ⇒ 81

Fig.3. Direct/indirect and fixed-depth/unfixed-depth valuation examples.

COMPARISON OF QUARTERABLENESS AND HUMANNESS

In a PLANNERlike language a 'mathematical predicate' QUARTERABLE would be

defined like a LISP predicate with the kernel (DE QUARTERABLE (X) (DIVIDABLE X 4)) and invoked name-directedly, eg by (QUARTERABLE 12) ⇒ T, but a 'logical predicate' HUMAN would be defined like a consequent theorem with the kernel (CONSE HUMANPROVER (X) (HUMAN ?X) (GOAL (MEDITATE <X <X))) and invoked pattern-directedly, eg by (GOAL (HUMAN TURING)) ⇒ (HUMAN TURING). In FIT however these kernels are defined - see Figs.4,5(h) - and invoked - see Figs.6,7(h) - in a unified way (cf U4).
To deepen this unification we show how both kernels can be refined (cf I3) in a completely parallel manner.

```
(W''(QUARTERABLE ?INTEGER) T)
(W''(QUARTERABLE ?X) ''(DIVIDABLE <X 2))
(W''(QUARTERABLE ?X) ''(GREATERP <X 3))

(>''(QUARTERABLE 4) T)
(>''(QUARTERABLE 10) F)
(>''(QUARTERABLE ?FIRSTSEVEN) F)
(>''(QUARTERABLE ?EIGHTDIVIDABLE) T)
(>''(QUARTERABLE ?X) ''(DIVIDABLE <X 2))

(V''(QUARTERABLE ?X)
  ''(ZEROZEROSUFFIX (BINARY <X)))
(V''(QUARTERABLE ?X)
  ''(AND (EVEN <X) (EVEN (HALF <X))))
```

Distributed definition of the QUARTERABLE predicate.

Fig.4.

```
(a) (W''(HUMAN ?MAMMAL) T)
(b) (W''(HUMAN ?X) ''(SHORTER <X  350cm))
(c) (W''(HUMAN ?X) ''(LIGHTER <X  500kg))

(d) (>''(HUMAN JESUS) T)
(e) (>''(HUMAN SHRDLU) F)
(f) (>''(HUMAN ?GOD) F)
(g) (>''(HUMAN ?POPE) T)
(h) (>''(HUMAN ?X) ''(MEDITATE <X <X))

(i) (V''(HUMAN ?X)
      ''(FEATHERLESS (BIPED <X)))
(j) (V''(HUMAN ?X)
      ''(AND (HUMAN (FATHEROF <X))
             (HUMAN (MOTHEROF <X))))
```

Fig.5. Distributed HUMAN definition.

```
(QUARTERABLE .8)⇒(?INTEGER .8)⇒failF
(QUARTERABLE 5)
⇒ (DIVIDABLE 5 2) ⇒ failF
(QUARTERABLE 2)
⇒ (GREATERP 2 3) ⇒ failF

(QUARTERABLE 4) ⇒ 4
(QUARTERABLE 10) ⇒ failF
(QUARTERABLE 6) ⇒ failF
(QUARTERABLE 8) ⇒ 8
(QUARTERABLE 12)
⇒ (DIVIDABLE 12 4) ⇒ 12

(QUARTERABLE 56)
⇒ (DIVIDABLE 56 4) ⇒ failU
⇒ (ZEROZEROPOSTFIX LLL000) ⇒ 56
(QUARTERABLE 1984)
⇒ (DIVIDABLE 1984 4) ⇒ failU
⇒ (ZEROZEROPOSTFIX LLLLL00000)⇒failU
⇒ (AND (EVEN 1984) (EVEN 992)) ⇒ 1984
```

Fig.6. QUARTERABLE valuation.

```
(a) (HUMAN FBI)⇒(?MAMMAL FBI)⇒failF
(b) (HUMAN CYCLOPS)
⇒ (SHORTER CYCLOPS 350cm) ⇒ failF
(c) (HUMAN EARTH)
⇒ (LIGHTER EARTH 500kg) ⇒ failF

(d) (HUMAN JESUS) ⇒ JESUS
(e) (HUMAN SHRDLU) ⇒ failF
(f) (HUMAN HOLYGHOST) ⇒ failF
(g) (HUMAN PAULVI) ⇒ PAULVI
(h) (HUMAN TURING)
⇒ (MEDITATE TURING TURING)⇒TURING

(i) (HUMAN ADAM)
⇒ (MEDITATE ADAM ADAM) ⇒ failU
⇒ (FEATHERLESS ADAM) ⇒ ADAM
(j) (HUMAN CAINE)
⇒ (MEDITATE CAINE CAINE) ⇒ failU
⇒ (FEATHERLESS CAINE) ⇒ failU
⇒ (AND (HUMAN ADAM) (HUMAN EVE))
⇒ CAINE
```

Fig.7. HUMAN valuation.

Definition assignments to ordinary complex variables (>-variables)

We first add Fig.4(g), saying that numbers dividable by eight are quarterable, and Fig.5(g), saying that popes are human; note that these may be regarded as abbreviations of (>''(QUARTERABLE ?X) ''(EIGHTDIVIDABLE <X)) and (>''(HUMAN ?X) ''(POPE <X)) respectively. Similarly Figs.4,5(f) say that the first seven integers are not quarterable and gods are not human. However, since this is not quite true, we add Figs.4,5(d) to cover the exception-FIRSTSEVEN '4' and the exception-GOD 'Jesus', which *are* quarterable and human respectively. Figs.4,5(e) say obvious specific negative things.

Definition assignments to non-deterministic variables (V-variables)

For other rules with the general premises (QUARTERABLE ?X) and (HUMAN ?X) of the kernel rules Fig.4(h) and Fig.5(h) we use V's (cf 12). Eg Fig.4(i) looks for a suffix of two zeros in the binary form of <X and Fig.5(i) attempts to prove that <X is a featherless biped; note that simple ANDs become implicit in fitter compositions. The rules Figs.4,5(j) are similar.

Definition assignments to demon variables (W-variables)

In addition we can type QUARTERABLE to fit integers only and HUMAN to fit mammals only, using the demon definitions Fig.4(a) and Fig.5(a) respectively (cf 14). Other types for QUARTERABLE/HUMAN are Figs.4,5(b), ensuring that the fittee is dividable by 2/shorter than 350cm, and Figs.4,5(c), ensuring similar things.

THE FITMENT VALUATION HIERARCHY

Although *the order in which definition assignments are made is immaterial*, in Figs.4-7 we use an order reflecting the fitment valuation hierarchy, which is the heart of the FIT valuator VAL:
First for each fitment *all* matching W-definitions are taken and their bodies are valuated for side-effects, where a single failF makes the whole fitment failF, as shown in Figs.6,7(a)-(c).
Then the most direct (specific) matching >-definition, whose valuation doesn't yield failU, is taken as in Figs.6,7(d)-(h). Eg the definition Fig.5(d), whose variable (HUMAN JESUS) contains constants only, is more direct than Fig.5(f), whose variable (HUMAN ?GOD) contains a typed variable ?GOD, which itself is more direct than Fig.5(h), whose variable (HUMAN ?X) contains the untyped variable ?X. Therefore the fitment (HUMAN JESUS) of Fig.7(d), for which there are the above three matching definitions, uses Fig.5(d), which is the most direct one of these.
Only when all >-definitions failU, *one* of the most direct matching V-definitions, not yielding failU, is taken as in Figs.6,7(i),(j). Eg if Adam as in Fig.7(i) is assumed not to meditate about himself, then we may use Fig.5(i) to verify his humanness via (FEATHERLESS (BIPED ADAM)) ⇒ (FEATHERLESS ADAM) ⇒ ADAM; note that we could have tried Fig.5(j) first, which is equally direct,but would have failUed if Adam is assumed to be motherless.
To understand Figs.6,7 completely note that the failU's indicate information we assume the date base to lack, the relevant definitions used by Figs.6,7 have a corresponding letter in Figs.4,5 (additionally Figs.4,5 (a)-(c) are used in the 'underground' of Figs.6,7(d)-(j)), and the atom F on valuation yields a failF.

REFERENCES

Bobrow, D. & Raphael, B. (1974) New programming languages for AI research. *Computing Surveys 6(3).*
Bobrow, D. & Winograd, T. (1977) An overview of KRL, a knowledge representation language. *Cogn. Sc. 1(1).*
Boley, H. (1976) A theory of representation. *IFI-HH-M-38, Fachbereich Informatik, Univ. Hamburg.*
Boley, H. (1977) Directed recursive labelnode hypergraphs: A new representation-language. *AI 9, 49-85.*
Boley, H. (1978) The FIT language for AI. *IFI-HH-M-55, Fachbereich Informatik, Univ. Hamburg.*
Bornat, R. & Wielinga, B. (1976) Does AI programming really have to be like knitting with spaghetti? *AISB76.*
Davis, R. & King, J. (1975) An overview of production systems. *STAN-CS-75-524, Stanford University.*
Hewitt, C. (1969) PLANNER: A language for manipulating models and proving theorems in a robot. *IJCAI69.*
Hewitt, C. (1975) How to use what you know. *IJCAI75.*
Hewitt, C. (1977) *Discussion contribution at the Symposium on AI and Programming Languages, Rochester.*
McQueen, D. (1978) Report on AI and Programming Languages Symposium Rochester, NY, Summer 77. *AISB 29.*
Salomaa, A. (1973) Formal languages. *Academic Press.*
Wilber, B. (1976) A QLISP reference manual. *Technical Note 118, AI Center, SRI.*
Winograd, T. (1975) Breaking the complexity barrier again. *ACM SIGPLAN Notices 10(1).*
Winston, P. (1977) Artificial Intelligence. *Addison-Wesley.*

DANGEROUS BEHAVIOUR

J.M. Brady, University of Essex, CO4 3SQ, U.K.

"A player should be cautioned about dangerous behaviour"*

Introduction

The ideas presented in this emasculated note derive from the author's experience on a project, extensively reported at the last AISB conference to develop a program capable of reading handprinted Fortran coding sheets (see Brady and Wielinga 1979 for a fuller account). The Fortran coding sheets project (FCSP) was originally based on three ideas: the active deployment of knowledge to guide the perception process, the exploitation of redundancy which is seemingly ever-present in real perception, and the "heterarchy conjecture".

In the years leading up to the project, a number of workers had emphasised the need for programs capable of highly flexible behaviour, seizing upon what was known, as and when it became available. Minsky and Papert conjectured (we call it the "heterarchy" conjecture) that such flexibility is best provided in computer science by process interactions, and that it is necessary to liberate such process interactions from the restrictions imposed by the LISP (or Algol-60) evaluator, where process records are allocated (and automatically deallocated) on a stack. This led to a number of experiments with control structures, such as pattern invocation, coprocessing and so on. The FCSP was thought of as being such an experiment, a view which prevailed until the Edinburgh conference.

This paper begins to isolate some of the traps which await the heterarchical programmer. It cautions against allowing control to dominate semantic issues, and suggests that the heterarchy conjecture leads to a perspective of computer vision as just another kind of problem solving. In fact, computer vision first needs to develop better computational models of visual phenomena such as occlusion, transparency, motion, as well as models of space and three-dimensional objects. We presuppose a detailed knowledge of the FCSP (Brady 1978, Brady and Wielinga 1979).

2. Dangerous behaviour (a recursion line)

This section consists of a list of the traps which await the unwary heterarchical programmer; most have ensnared the FCSP. We recall that the claims for heterarchy were based upon the observation that AI programs need to be capable of flexible behaviour, which is dynamically determined, and asserted that such flexibility is best afforded by control structures and process interactions. It should be clearly understood that the argument for flexibility is taken as read; it is the claims about process potency which are being examined here. We return to this point in section 3.

(1) Computer science has in fact a very limited stock of basic control structures other than those provided in LISP or Algol-60. Systems such as Conniver essentially additionally provide coprocesses and if added methods. The former are well suited to describing genuinely coexisting processes, while if added methods neatly model an ability to respond to infrequent events, such as slates falling off roofs while walking, and situation-action rules. A consequence of this limited stock is that it

* Advice given by the Footabll Association to British soccer referees.

is extremely difficult to construct the rich behaviours naturally called for after consideration of the domain. The FCSP junction computations (Brady 1979) are an example of this. In such cases, the power of process interactions is largely illusory.

(2) Sussman and McDermott (1972) have pointed out that the set of control and data structures offered in a programming language significantly influence problem solving based on that language. In particular, the programming language itself, rather than semantic considerations of the domain under investigation, can all too easily impose global organisation on a program. In this way, global organisation is essentially determined in advance by the choice of language and may well be wholly inappropriate, especially given the limited stock of basic control structures referred to above. Examples of this problem abound in the FCSP: its overall organisation as interacting character and Fortran "experts", and the use of a Conniver-like system in both the reasoner and an initial version of the character system.

(3) The difficulty of constructing rich semantically-appropriate behaviours by a program is directly reflected in the difficulty of specifying such behaviour. It is often not possible to state in advance precise behavioural goals for a program. The FCSP was careful never to do so! Generally, one has to rely on subjective judgement as to whether a program is behaving sensibly. It has been argued that a definition of a program's behaviour is embodied within the program; but a programmer usually has at any stage a clear idea of what he is trying to achieve (even if his ideas change over time). The point is that he is incapable of articulating them precisely other than by writing a program. This leads to the next point:

(4) In the absence of a useful formal semantics for goal-oriented non-sequential systems, it is impossible to analyse or predict the behaviour of such a program. Programs such as the FCSP typically work interestingly on a number of examples (the ones written up in papers!) but loop infinitely or collapse unceremoniously on others. At the Edinburgh conference, the project team gave a lecture which simulated coprocess behaviour of the entire system. The trials we conducted were initially very encouraging, and it was clear that under certain circumstances the different knowledge sources would help each other, and that the FCSP would converge upon an interpretation of a line that would not have been interpretable by any of the subsystems. To that extent, the FCSP completely fulfilled our original predictions. It was, however, extremely difficult to precisely characterise the circumstances under which the system would or would not converge to an overall interpretation. In the cases that it would not, it was hard to see why it would not. In the cases that it would, it was hard to see exactly what knowledge was crucial or how it was used. In short, it was very difficult to precisely analyse or predict the program's behaviour. The scope of a program is usually very far from clear. To the author's knowledge no proof of termination has been given for any heterarchical program for those examples within its scope. If Artificial Intelligence is to be taken seriously as a science, it should surely be concerned with such precise analysis.

(5) Placing such a strong emphasis on control structures inevitably seduces a programmer into believing that the way to repair the observed deficiencies of a program is by more of the same. In fact, such deficiencies usually point to inadequate theories, semantics or

representations. The segmenter and junction computations in the FOSP exemplify this problem, as does Adler's (1977) reliance on coprocesses to overcome a highly inadequate representation of depth.

(6) As control structures become more sophisticated, so the overall management of control becomes very difficult. One is inevitably lead to the familiar idea of scheduling process activity. Such a scheduler may be based on a simple domain-independent criterion, such as queue order or weight of numbers. The criticisms which can be levelled against such a scheme are essentially those that are levelled against knowledge-free searching. A scheduler may of course embody domain specific knowledge; this merely pushes the problem back one step and one soon has to think about scheduling the schedulers.

It should be clearly noted that we are not arguing that a concern with control structures is irrelevant to Artificial Intelligence. A program must be capable of flexible sensible behaviour. Fahlman's (1973) program stands out as a piece of AI, although his concentration on single block moves wherever possible leads to inefficient solutions to some problems. Any attempt to integrate techniques such as scaffolding more fully within the program would have serious consequences for control: a better representation is in fact what is needed.

Again, the idea of processes communicating indirectly via a data base neatly enables one to decouple the invocation of a procedure from assumptions about what is known. The point is that a LISP or Algol program's knowledge is implicit in its access and control environment at the time of its call. The point we wish to make here is that a concern for control issues all too easily dominates the semantic issues of what entities denote and what a piece of computation is for.

We also note that a number of premisses underlie the arguments raised above, including: the currently limited stock of control structures, the lack of a useful formal model of nondeterminism, and the lack of adequate theories and representations about vision. These are observations about the current situation, not facts, and may well change in the near future.

3. What's the alternative?

The purpose of this paper is to isolate some of the traps which await the heterarchical programmer, not preach "truth". However, we conclude by briefly examining some of the consequences of the above. A fuller account of the ideas in this section appear in Brady (1978). Firstly, if we take the point about formal analysis and prediction seriously, then, in the absence of a useful model of non sequential systems, representations must be static data structures. Processes then interact by updating a shared data structure. The points made about partial knowledge states in (Brady and Wielinga, 1979) amount to insisting that the representation admits of sufficient granularity, and defines behaviour as the sequence of intermediate representations. It seems that we are taking a declarativist position in the well-known debate (Winograd, 1977). A familiar objection is that one man's data structure is another man's procedure in the sense that any data structure is interpretable and so can be thought of as a procedure. The essential difference seems to be the extent to which one is committed in advance to a particular interpreter. We have argued elsewhere that it is possible to avoid many of the problems which are taken as indicative of of a need for rich process interaction by supplying sufficient power

of articulation in a representation. Often, one is forced to choose between hypotheses, which can be represented, each of which amounts to claiming more than is justified. If one cannot represent "P or Q", one has to choose to represent either P or Q individually with the consequent possibility of having to re-examine that choice later. A lot of difficult issues bear upon this point, specifically regarding uncertainty. Static representations are relatively uncommitted to sequential processing, and can be developed by constrained parallel algorithms. This is a major advantage in the case of computer vision. It may be argued that the sequence of intermediate partially formed representation corresponds directly to the sequence of intermediate states of a heterarchical program, so that the differences can be resolved by a suitable operational semantics. However, one examines a single step in the incremental development of a representation in domain specific terms, whereas the operational meaning of two coroutines (say) involves the entire meaning of the individual processes. It may not have escaped some readers that even if one accepts the need for flexibility, there are other ways of achieving it in existence in AI, for example production systems and constraint satisfaction.

References

M.R. Adler (1977): "Computer recognition of occluded curved line drawings", Ph.D. thesis, University of Edinburgh.

J.M. Brady (1978): "The development of a computer vision system", Psicologica Recherche.

J.M. Brady and B.J. Wielinga (1979): "Reading the writing on the wall", Computer Vision Systems, (ed. Hanson and Riseman) Academic.

S.E. Fahlman (1973): "A planning system for robot construction tasks", Artificial Intelligence 5, 1-48.

G.J. Sussman and D.V. McDermott (1972): "Why conniving is better than planning", A.I. memo 255, MIT.

EXPERIMENTAL LEARNING MODEL

Pavel Brazdil

Department of Artificial Intelligence
University of Edinburgh, Scotland

ABSTRACT: The aim of our work is to investigate how a relatively small set of rules can be transformed into a running program capable of solving a fairly wide variety of problems. We have written a program ELM (Experimental Learning Model) which modifies a given set of rules (clauses) as a result of its experience. The problems are chosen from the domain of school arithmetic/algebra and solved by one part of our system (with the help of the teacher). The solutions are analyzed and modifications to the existing system of rules are performed as a result.

CONTENTS:

1. Input Provided by the Teacher
2. Search for a Solution
3. Assimilation of Priority Orderings
4. Conflict Resolution
5. Results and Conclusions

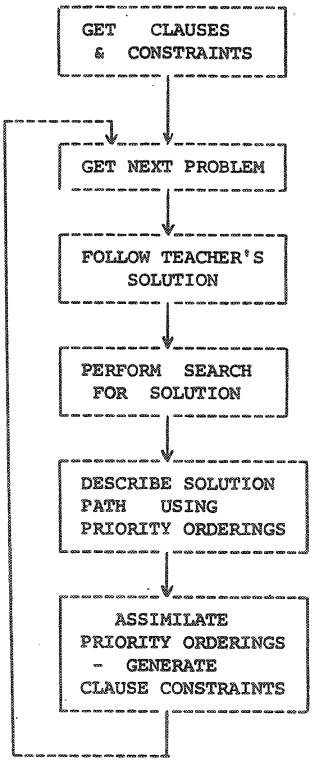


FIG.1 BASIC VIEW OF ELM

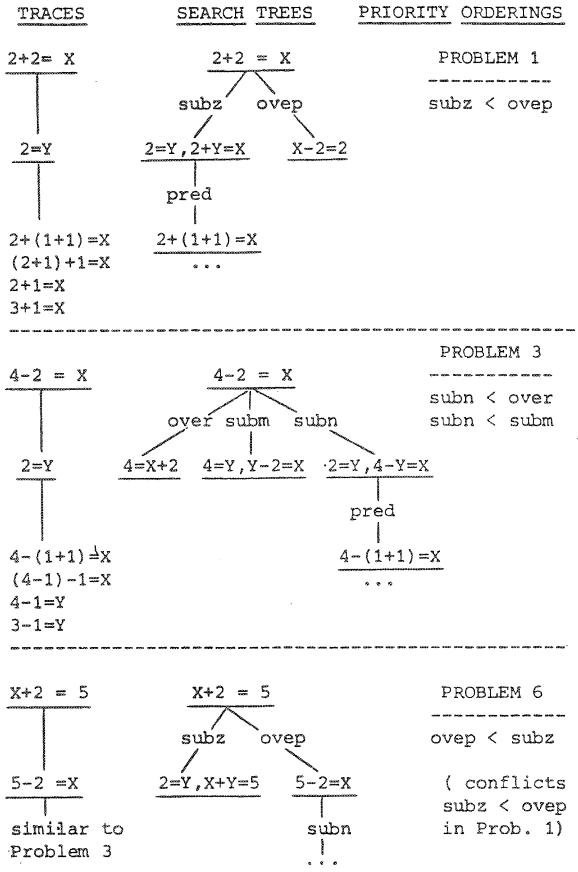
1. INPUT PROVIDED BY THE TEACHER

In this section we will concentrate on the description of the information the learning model is provided by the teacher. Later, we will show how problems are solved and what kinds of modifications our system can produce. Fig.1 shows the basic view of the system.

(1) At present ELM needs to be provided with a set of clauses which express various rules the teacher might consider useful for solving a certain class of problems. Fig.2 shows such a set of clauses intended for manipulating arithmetic and algebraic expressions. For example, the clause 'asoc' expresses associativity. The clauses contain little information about when they should be used. We will use our work to show that it is relatively easy to generate this information automatically.

(2) We provide the system with a set of predicates (constraints), which the system may insert among the existing predicates in the clause. Their function will be to constrain clause selection. In the implementation we have used the following:

predicate	var(X)	true if	X is a variable,
	int(X)	-"-	X is an integer,
	X:=U+V	-"-	X is of the form U+V.



Experience with Prob.3 facilitates the solution of Problem 6, since the subgoal 5-2=X is solved without search. Problem 1, on the other hand, eliminates temporarily the branch leading to the the solution of X+2=5. Thanks to trace provided the error is soon discovered and rectified.

FIG.4 EXAMPLES OF TRACES AND SEARCH TREES

```
asoc:  X1+(X2+X3)=X4 <- (X1+X2)+X3=X4.
ovep:  X1+X2=X3 <- X3-X2=X1.
over:  X1-X2=X3 <- X1=X3+X2.
subm:  X1-X2=X3 <- X1=X0 & X0-X2=X3.
subn:  X1-X2=X3 <- X2=X0 & X1-X0=X3.
subs:  X1+X2=X3 <- X1=X0 & X0+X2=X3.
subz:  X1+X2=X3 <- X2=X0 & X1+X0=X3.
pred:  1=1+0 <-.    2=1+1 <-.    etc.
suc:   0+1=1 <-.    1+1=2 <-.    etc.
```

FIG.2 EXAMPLE OF CLAUSES GIVEN TO ELM

(3) ELM needs to be given a sequence of problems to solve. Fig.3 shows an example of problems given to ELM. Each problem is solved by evaluating the arithmetic expressions and/or by finding the correct value for the variable in the expression.

(4) Generally, learning models may require some form of search guidance in their search for solution. ELM is shown a sequence of subgoals as they are to be solved - a trace. An example of a trace is shown in Fig.4. Traces are produced by a routine which in effect simulates the teacher.

2. SEARCH FOR A SOLUTION

ELM responds to each problem by initiating a search for a solution. For each subgoal a set of clauses is selected, provided they match. The ordering of clauses is also respected. Each clause, when applied replaces the current subgoal by its body (Fig.4). The application creates a new node in the search tree, which is expanded breadth-first. The trace provided is utilized to terminate a branch whenever the current subgoal differs from the one in the trace. The search terminates whenever there are no more subgoals to be solved,

```
1/  2+2 =X
2/  (1+1) +2 =X
3/  4-2 =X
4/  (5-1) -2 =X
5/  5-3 =X
6/  X+2 =5
7/  X+ (1+1) =5
8/  ((1+X) +2)+3 =8
9/  X+3=Y &  Y+1=9
```

FIG.3 A TYPICAL SEQUENCE OF PROBLEMS GIVEN

or whenever no clause can be found for solving a particular subgoal. The answer 'yes', or 'no' respectively, is given.

The search trees obtained by the problem solving routine are analyzed with the objective to eliminate search. Experience with one problem often facilitates solution of similar problems in the future, but difficulties can also be introduced. Illustrations of that are given in Fig.4. Thanks to the trace shown the errors are soon discovered and rectified by modification of clauses. The basic strategy of ELM is to restrict selection criteria for clauses that do not lie on the solution path Either priority orderings or clause constraints can be introduced.

(1) Priority orderings specify the order in which the clause selection should be performed. The priority ordering C<D, for example, indicates that the clause C should be selected preferentially before D. The selection of clause D is restricted as a result. The priority orderings are generated as as the solution path in the search tree is described *.

(2) Clause constraints can be used as an alternative means for restricting clause selection. Clause constraints are predicates whose truth/falsity is tested during clause selection. If they are not true the clause to be selected is ignored.

3. ASSIMILATION OF PRIORITY ORDERINGS

After the solution of each given problem has been reached the priority orderings describing the solution path are assimilated with the existing ones. They are added to the orderings previously remembered unless conflicts are detected, in which case new clauses are generated.

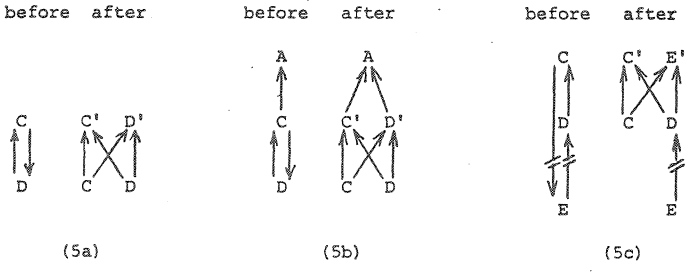


FIG. 5 PRIORITY ORDERINGS
(before and after conflict resolution)

For example, the conflict between C<D and D<C is resolved by generating two new clauses C' and D', constrained versions of C and D. They constrained versions are always given preference to the unconstrained ones (Fig.5a). We prefer to generate both clauses C' and D' although in principle we could have generated just one of them. However, we would have to make a choice, which we prefer to avoid.

---

* Sussman (1973) discusses the role of a 'critic' in his learning system HACKER. The description of the solution solution path can be viewed as 'criticism' of the present clause selection process.

The old clauses C and D are kept as a back-up and used whenever the new clauses do not get selected. Conflicts involving three or more clauses are resolved in the way outlined in Fig.5c. Some additional priority orderings are sometimes generated as a result of integration of the new clauses C' and D' among the existing clauses. Two such priority orderings, shown in Fig. 6b, ensure that the priority given to clause A is unaffected by conflict resolution.

4. CONFLICT RESOLUTION

The conflict resolution routine requires the contexts in which the clauses to be constrained were selected or rejected. The subgoal solved when clause C was selected (eg. when C<D was generated) is used as its selection context. The rejection context is defined similarly. Both contexts are stored together with the priority orderings generated. Had they not been stored, appropriate problem(s) would have to be found, the solutions re-created and the desired contexts retrieved. The conflict resolution routine takes each clause to be constrained and searches for its variables and corresponding terms in the selection and rejection contexts. Suppose that the routine is trying to resolve the conflict between subz<ovep and ovep<subz (Fig.4), and is just considering the following terms:

Clause head:	Selection context:	Rejection context:
X1+X2=X3	2+2=X	X+2=5
'	'	'

The conflict resolution routine takes each possible constraint (var(X), int(X) or X:=T) and applies it to the terms chosen. If the predicate is true when applied to the selection context and, at the same time, false when applied to the rejection context it is added as a disjunct to the expression being built. Since for example int(2) is true and int(X) false, the predicate int(X1) is added to the existing constraints. Further investigation yields var(X3), which is added as a disjunct to the constraint int(X1) found previously.

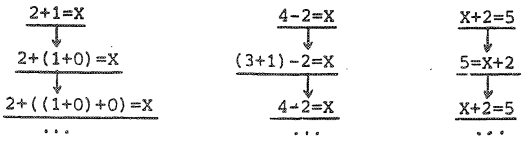
Disjunctions of constraints are preferred because arbitrary choice of a constraint does not need to be made. Also, we prefer more general constraints to special ones in order to avoid producing overspecialized and generally inapplicable clauses.

For example,	int(X) X:=X1+X2 int(X1) v var(X3)	is preferred to -"- -"-	X:=2, X:=2+1, int(X1).
Also,	C1 & (C1v C2) (C1v C2) & (C1v C3)	is replaced by -"-	C1, C1.

The last rule is used quite often by our program. The repeated occurence of the same predicate can be taken as evidence that it is relevant for clause selection while all the others are merely incidental.

5. RESULTS AND CONCLUSIONS

The learning model ELM discussed in this section has been implemented in PROLOG. In one of our experiments the program was given 20 clauses, but no priority orderings. Sequence of problems was also given (Fig.2,3). The search trees contained only one branch leading to the solution after one run through the learning sequence. Although the program knew nothing about 'loops' many were eliminated nevertheless, eg. the ones shown. In total twenty-two new clauses were generated, two for each conflict detected. Some of the clause new constraints are shown in Fig.6b, and corresponding priority orderings in Fig.6a. We are planning to extend the set of possible constraints to include others, which should enable us to deal with more complex equations in the future. Also, we intend to utilise to a greater extent the information in the trace, basically to generate clauses.




```
over.2:  X1-X2=X3 <- (X1:=3  v X2:=1) & int(X1)
over.1:  X1-X2=X3 <- int(X1)    ...
subm.1:  X1-X2=X3 <- X1:=X4-X5  ...
subh.1:  X1-X2=X3 <- X1:=4   v X2:=2     ...

ovep.1:  X1+X2=X3 <- var(X1) v int(X3)   ...
subs.1:  X1+X2=X3 <- X1:=X4+X5 v X2:=1   ...
subz.2:  X1+X2=X3 <- int(X1)    ...
subz.1:  X1+X2=X3 <- int(X1) v X2:=2     ...
```

FIG. 6b EXAMPLES OF CONSTRAINTS GENERATED
(Other predicates follow the constraints shown)

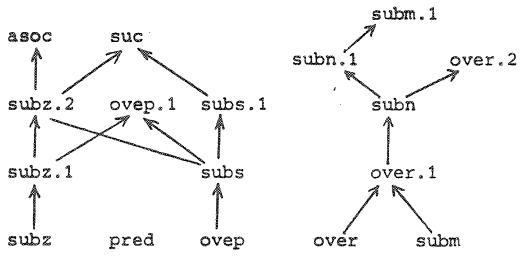


FIG. 6a SOME PRIORITY ORDERINGS GENERATED
(The arrows point to higher priority clauses)

The basic contribution of our work is, we believe, in that we show how analysis of problem solving activity can be used to produce systematic modifications to the existing program.

ACKNOWLEDGEMENTS AND REFERENCES

I wish to thank to Gordon Plotkin for many helpful discussions and comments on this paper. Also, thanks to IBM UK Ltd. for their help and support in the course of my stay in Edinburgh.

Sussman, G.J. (1973) A Computational Model of Skill Acquisition, MIT Tech. Report AI TR-297, MIT, Cambridge

Vere, S.A. (1977) Induction of Relational Productions in the Presence of Background Information, Proceedings of 5th IJCAI, 1977, MIT, Cambridge

Winston, P.H. (1970) Learning Structural Descriptions from Examples, MIT Tech. Report AI TR-231, MIT, Cambridge

Waterman, D.A. (1975) Adaptive Production Systems, Proceedings of 4th IJCAI, 1975, Tbilisi, USSR

UNDERSTANDING THE IMAGE FORMING PROCESS

M.J.BROOKS
DEPARTMENT OF COMPUTER SCIENCE,
UNIVERSITY OF ESSEX,
COLCHESTER, ENGLAND.

Abstract

Relatively little work has been done to try to understand the image forming process. It is contended here that this situation must change if vision programs are to attain any level of sophistication. Some difficult problems are posed and, as a first step towards answering them, an attempt is made to understand the effects of a light-source which is a 'sky', rather than the traditionally adopted point source.

Key words

Image forming process, analytic approach, point and sky light sources, reflectivity function.

Introduction

Despite the great effort that has gone into AI vision research in the last 15 years, very little work has been done to try and understand the physics of illumination and reflection. Our knowledge of the way surfaces and light interact to give different visual appearances is, at best, crude: for example, we have little idea of how we are able to distinguish between the myriad of different surface lustres which are common occurences in our visual world (eg. chrome, matt paint, varnished wood, etc.), or how shading is affected by shape and illumination. A consideration of human abilities, coupled with the lessons learned from early programming efforts (Horn 1970, Woodham 1977) suggests there is much to be gained from improving our knowledge in such areas. At the very least, we should assess the extent to which an understanding of the image forming process might help us before attempting to write programs which purport to show any level of 'intelligence'. See Horn, 1977 for a statement of this methodological stance.

Unfortunately, serious attempts to understand the image forming process seem destined to plunge headlong into complicated mathematics. Take, for instance, the seemingly innocent mutual illumination problem as attacked by Horn, 1975; despite several simplifying assumptions,

the problem has, so far, eluded analytic solution. This level of mathematical difficulty may be partly to blame for the small amount of work done in the field.

A common objection to the analytic approach is that complicated mathematical techniques are not really relevant to an AI, symbol processing, vision program. But this is to miss the point: few, indeed, would claim that differential equations belong -in any key sense- in a program simulating human perception; however, the objective of the above methodology is to improve our understanding of the physics of visual phenomena with a view to incorporating any useful results obtained into a vision program in a relatively simplistic form. As a parallel, some rather difficult mathematics contributed to the derivation of accurate edge-detecting operators (Brooks 1978a), though their final form essentially relies on simple addition and subtraction of light levels.

Some important problems

Let a scene configuration be uniquely characterised, for our purposes, by specifying values for the following parameters:

1. Surface shape: sphere OR cylinder OR plane

2. Surface lustre: matt OR shiny OR combinations of both

3. Lighting type: point OR finite plane OR sky (infinite)

4. Light-source emission: perfect scatterer OR lambertian

5. Geometry: light-source and viewer together and distant
OR many variations

In these terms, 'perception' of a scene will assign a set of consistent values to our 5 parameters. It is the author's intention, in the context of these scenes, to begin to answer the difficult and general questions outlined below.

Given a digitised image of a scene, how do we start to fill in the values of our parameters? It seems that before making inferences about surface shape we need to know something about the illumination, and vice-versa. If this is so, what initial assumptions are necessary? What assumptions do people make? (They probably assume surface continuity, for instance.) It would be useful to analytically generate photographs of scene configurations and gauge human response. Note that the programs of Horn, 1970 and Woodham, 1977 - which attempt to account for the orientation of each pixel in the image - do not, in our

sense, start, since they are fed with substantial pre-requisite information. As such, they go a little way towards answering the problem of how inferences are made at an intermediate stage. This will clearly require an appreciation of the inter-dependence of the 5 parameters. Intertwined with the above problems are the issues of what can and cannot be computed at a given stage, and to what level ambiguity is inherent in images. (And not just ambiguity of the Necker variety. Two different scene configurations could give rise to the same image, eg. a sphere under a point light-source and an ellipsoid under a sky.)

As an attempt to tackle these problems, we continue by investigating the effects of different light-source models.

Computing the appearance of Horn type surfaces beneath a lambertian sky.

To date, attempts to model the image forming process have only considered the distant point light-source - a very tractable choice. It is of interest here to gauge the effects, on the image formed, of a sky light-source.

First, we must consider the way in which our model of the sky is to emit light from each point. A perfect scatterer is not feasible since an infinite amount of energy will be received by any tilted surface beneath such a sky (Brooks 1978b). A lambertian emitter -one that emits light proportionately to the cosine of the emittance angle- is a more realistic model. We also need to consider the way surfaces are to reflect light. We choose here to adopt Horn's model of surface reflectivity. (See Horn, 1970 and Brooks, 1978c.)

Consider a point, on a small scale Y, with a local surface normal (a,b,c). Let the viewer position be (p,q,r), as displayed in figure 1.

The flux density at Y due to X, along the line of the incident ray $= \frac{F\cos(\phi)}{2\pi l^2}$

Luminance of Y viewed from (p,q,r) $= \frac{F\cos(\phi)}{2\pi l^2} \cdot \Phi$

where Φ is the reflectivity function representing the proportion of energy retained. Translating to cartesian coordinates and integrating over the whole of the light

source actually contributing to Y, we obtain the following integral, representing the viewed luminance of Y.

$$I = (1-s)\frac{F}{4}(1+c) + s.\frac{FD}{2\pi}\int_{-\infty}^{\infty}\int_{-\frac{b}{a}y-\frac{c}{a}b}^{\infty}\frac{(\lambda_1 x+\lambda_2 y+\lambda_3 D)^n}{(x^2+y^2+D^2)^{\frac{n+3}{2}}}\,dx.dy$$

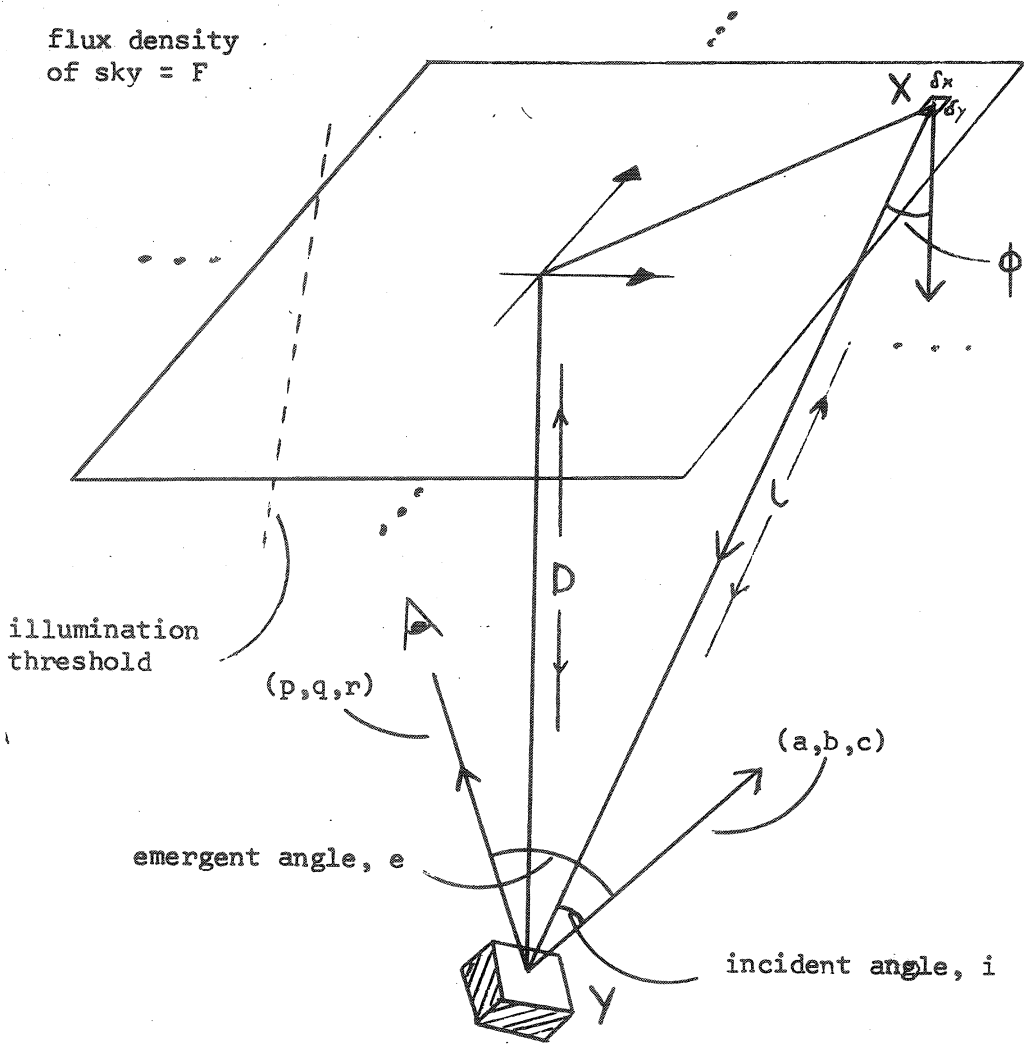


Figure 1

This is a non-trivial integral. Its cumbersome solution and fuller derivation are detailed in Brooks 1978b. Fortunately, when n, the degree of specularity, is odd, the result collapses to a relatively simple expression. The luminance of a horn type surface oriented at (a,b,c), viewed from (p,q,r) and beneath a lambertian sky is listed below for n=1,3,5. Note both vectors are assumed normalised.

$$L_1 = (1-s)\frac{F}{4}(1+c) + \frac{sF}{4}(\lambda_3 + \cos(e))$$

where $\lambda_3 = 2c\cos(e) - r$

$$L_3 = (1-s)\frac{F}{4}(1+c) + \frac{sF}{8}\left(3(\lambda_3+\cos(e)) - (\lambda_3^3 + \cos^3(e))\right)$$

$$L_5 = (1-s)\frac{F}{4}(1+c) + \frac{sF}{32}\left(15(\lambda_3+\cos(e)) - 10(\cos^3(e)+\lambda_3^3) + 3(\lambda_3^5+\cos^5(e))\right)$$

The author is currently using these results to generate photographs of various scene configurations.

Acknowledgements

My thanks go to Pat Hayes for suggesting this approach, to Anthony Holt for some very useful help with the integration, and to Chris Rowbury, Mike Brady and Pat for help with the draft. This work was funded by the Science Research Council.

References

Brooks, M.J (1978a) Rationalising Edge Detection, Computer Graphics and Image Processing, to appear.

Brooks, M.J (1978b) Investigating the effects of planar light sources, CSM 22, Dept. Computer Science, University of Essex.

Brooks, M.J (1978c) A critique of Horn's reflectivity function, CSM 23, Dept. Computer Science, University of Essex.

Horn, B.K.P, (1970) Shape from shading: a method for obtaining the shape of a smooth opaque object from one view, MAC TR-79, MIT.

Horn, B.K.P, (1975) Image intensity understanding, AIM 335, MIT.

Horn, B.K.P, (1977) Towards a science of image understanding, pg. 648, IJCAI, 1977.

Woodham, R.J (1977) A cooperative algorithm for determining object relief from one view, MIT, 1975.

A SEQUENT CALCULUS FOR MODAL QUANTIFICATIONAL LOGIC

F. M. BROWN
University of Edinburgh
Edinburgh, Scotland

Abstract

We give an axiomization of a modal quantificational logic which captures the notion of logical truth. This modal logic is stronger than S5. Next we describe a sequent calculus for this modal logic, and show that it is complete. Finally we describe some potential applications of Modal Logic to Artificial Intelligence.

1. Introduction

We first describe in section 2 a very strong modal logic which captures the notion of logical truth of the meaning of object language sentences. That is, for example, letting $\vdash$ be the modal symbol for logical truth, and M be our recursive meaning function (Brown [1977d]), then $\vdash$ (M S) states that the meaning of the object language sentence S is logically true. Thus, amazing as it may seem, we can construct a definition of logical truth without the use of any set-theoretic concepts.

Next in section 3 we give a complete set of rewrite rules for this modal logic which may be used to form a sequent calculus for modal quantificational logic. In particular these rules are intended to be added to a classical sequent calculus such as: Brown [1976], Brown [1977a], Brown [1977b], Brown [1977c].

Finally in section 4 we describe some applications of our modal logic to Artificial Intelligence.

2. Axiomization of Modal Logic

After explaining in section 2.1 the logical notation that we use, we give in section 2.2 the logical axioms of our modal logic. In section 2.3 we discuss what we call the Consistency Problem of modal logic, and explain why any theory formulated in modal logic should also include certain specific non-logical axioms about possibility.

2.1 Notation

We now explain our notation.

The symbols of classical logic are listed below with their English Translations:

$x \wedge y$	x and y
$x \vee y$	x or y
$x \rightarrow y$	if x then y
$x \leftrightarrow y$	x iff y
$\sim x$	not x
$\blacksquare$	true
$\square$	false
$\forall X\ \varphi X$	for all objects X, φX holds.
$\exists X\ \varphi X$	for some objects X, φX holds.
$\forall x\ \varphi x$	for all propositions x, φx holds.
$\exists x\ \varphi x$	for some propositions x, φx holds.

Capital letters such as X range over objects, whereas small letters such as x range over propositions.

The symbols of modal logic are:

$\vdash x$	x is logically true
$\vdash x\ y$	x entails y
$\Diamond x$	x is possible
(World x)	x is a World.

The last three modal symbols are defined in terms of the first one as follows:

$\vdash x\ y$	$=_{df}\ \vdash (x \rightarrow y)$
$\Diamond x$	$=_{df}\ \sim\vdash\sim x$
(World x)	$=_{df}\ (\Diamond x) \wedge \forall y\ (\vdash x\ y) \vee (\vdash x(\sim y))$

2.2 The Logical Axioms of Modal Logic

The axioms of our modal logic include any complete axiomization of classical quantificational logic, with propositional quantifiers plus the following inference rule and axioms about modality:

RO: from x infer $\vdash x$

A1: $\vdash x \rightarrow x$

A2: $\vdash (x \rightarrow y) \rightarrow (\vdash x \rightarrow \vdash y)$

A3: $\vdash x \vee \vdash \sim \vdash x)$

A4: $(\forall w((\text{World } w) \rightarrow \vdash w\ x)) \rightarrow \vdash x$

RO, A1, A2, and A3 are essentially the inference rule and axioms of S5 modal logic. Axiom A4 which we call Leibniz's postulate expresses his intuition that something is logically true if it is true in all possible worlds. A good introduction to modal logic is given in Hughes and Cresswell [1968].

2.3 The Consistency Problem

The consistency problem of modal logic is that from the logical axioms of modal logic we cannot prove certain elementary facts about the possibility of conjunctions of distinct possible negated atomic expressions consisting of non-logical symbols. For example, if we have a theory formulated in our modal logic which contains the non-logical atomic expression (ON A B) then since $\sim$(ON A B) is not logically true, it follows that (ON A B) must be possible. Yet $\Diamond$(ON A B) is not a theorem of our modal logic.

Thus, for any theory expressed in modal logic, a certain number of non-logical axioms dealing with possibility should also be added. For example, in the case of the propositional logic, or in the case of the quantificational logic over a finite domain since it reduces to propositional logic, one sufficient but inefficient axiomization would be to assert the possibility of all consistent disjunctions of conjunctions of literals as additional non-logical axioms:

$$\Diamond(\bigvee(\bigwedge \text{Literals}))$$

A more computationally efficient axiomization which is obtained by noting that the possibility of a disjunction of sentences is implied by the possibility of any one of those sentences:

$$\Diamond x \rightarrow \Diamond(x \vee y)$$

is to assert only the possibility of all consistent conjunctions of literals:

$$\Diamond(\wedge \text{literals})$$

Using our meaning function (Brown [1977d]) this may be done in a finite manner by adding the single axiom:

$$(\text{Conj } S) \wedge (\text{Consist } S) \rightarrow \Diamond(M\ S)$$

where Conj and Consist are recursive functions defined as follows:

$(\text{Conj } S) = \text{df } (\text{Lit } S) \vee \exists T\ \exists R\ (S=\ulcorner T \wedge R\urcorner \wedge (\text{Lit } T) \wedge (\text{Conj } R))$

$(\text{Lit } S) = \text{df } (\exists T\ S=\ulcorner \sim T\urcorner \wedge (\text{Atomicsent } T)) \vee (\text{Atomicsent } S)$

$(\text{Consist } \ulcorner\ \urcorner) = \text{df } \blacksquare$

$(\text{Consist } \ulcorner S, L\urcorner) = \text{df } (\text{Consist2 } S\ L) \wedge (\text{Consist } L)$

$(\text{Consist2 } S\ \ulcorner\ \urcorner) = \text{df } \blacksquare$

$(\text{Consist2 } S\ulcorner T, L\urcorner) = \text{df } \sim(\text{Opp } S\ T) \wedge (\text{Consist2 } S\ L)$

$(\text{Opp } S\ T) = \text{df } (\exists R\ S=\ulcorner \sim R\urcorner \wedge T=R) \vee (\exists R\ T=\ulcorner \sim R\urcorner \wedge S=R)$

The methods for representing object language expressions in our logic and for obtaining their meanings are defined in Brown [1977d]. For example $\ulcorner T \wedge R\urcorner$ is essentially a structural descriptive name of $(M\ T) \wedge (M\ R)$.

3. A Sequent Calculus for Modal Logic

We give in section 3.1 some theorems of our modal logic which when used as rewrite rules will form the basis of a method for translating every expression of our modal logic into the standard form described in section 3.2. In section 3.3 we show that these rewrite rules form the basis of a complete proof procedure for modal logic.

3.1 Rules of the Sequent Calculus

We list below thirteen theorems of our modal logic of the form $A \leftrightarrow B$ or $C \rightarrow (A \leftrightarrow B)$ which may be used to rewrite rules replacing A by B in any context in which C is a hypothesis. The symbols World* and $\vdash^*$ have the same meaning respectively as world and $\vdash$, but are never to be replaced by their definitions in a proof procedure using these rules. Furthermore any initial theorem given to such a proof procedure must not itself contain the world* or $\vdash^*$ symbols, although it could of course contain the World and $\vdash$

symbols.

$\vdash$: $(\vdash x) \leftrightarrow \forall w(\text{World}^* w) \rightarrow \vdash^* w\ x)$

$\vdash\wedge$: $(\text{World}^* w) \rightarrow ((\vdash^* w(x \wedge y)) \leftrightarrow (\vdash w\ x \wedge \vdash^* w\ y))$

$\vdash\vee$: $(\text{World}^* w) \rightarrow ((\vdash^* w(x \vee y)) \leftrightarrow (\vdash^* w\ x \vee \vdash^* w\ y))$

$\vdash\rightarrow$: $(\text{World}^* w) \rightarrow ((\vdash^* w(x \rightarrow y)) \leftrightarrow (\vdash^* w\ x \rightarrow \vdash^* w\ y))$

$\vdash\leftrightarrow$: $(\text{World}^* w) \rightarrow ((\vdash^* w(x \leftrightarrow y)) \leftrightarrow (\vdash^* w\ x \leftrightarrow \vdash^* w\ y))$

$\vdash\sim$: $(\text{World}^* w) \rightarrow ((\vdash^* w(\sim x)) \leftrightarrow \sim \vdash^* w\ x)$

$\vdash\blacksquare$: $(\text{World}^* w) \rightarrow ((\vdash^* w\ \blacksquare) \leftrightarrow \blacksquare)$

$\vdash\Box$: $(\text{World}^* w) \rightarrow ((\vdash^* w\ \Box) \leftrightarrow \Box)$

$\vdash\forall$: $(\text{World}^* w) \rightarrow ((\vdash^* w(\forall x\ \varphi x)) \leftrightarrow (\forall x \vdash^* w\ \varphi x))$

$\vdash\exists$: $(\text{World}^* w) \rightarrow ((\vdash^* w(\exists x\ \varphi x)) \leftrightarrow (\exists x \vdash^* w\ \varphi x))$

$\vdash a$: $(\text{World}^* w) \rightarrow ((\vdash^* w(\forall x\ \varphi x)) \leftrightarrow (\forall x \vdash^* w\ \varphi x))$

$\vdash e$: $(\text{World}^* w) \rightarrow ((\vdash^* w(\exists x\ \varphi x)) \leftrightarrow (\exists x \vdash^* w\ \varphi x))$

$\vdash\vdash$: $(\text{World}^* w) \rightarrow ((\vdash^* w(\vdash x)) \leftrightarrow \vdash x$

The $\vdash\forall$ and $\vdash\exists$ theorems pertain to quantifiers of object language variables whereas the $\vdash a$, $\vdash e$ theorems pertain to quantifiers for propositional variables. The $\vdash\forall$ and $\vdash\exists$ theorems are equivalent to the fact that something is an object iff it is logically true that it is an object; and the ta and te theorems are logically equivalent to the fact that something is a proposition iff it is logically true that it is a proposition. All these theorems hold regardless of whether propositions are objects or not.

In order to try to prove a theorem ψ with a proof procedure using these rules, it must actually try to prove $\vdash\psi$ instead. There is a deep and beautiful reason for this which is basically that this initial $\vdash$ inserted before ψ is a symbol of the metalanguage of this logic as are all the $\vdash^*$ and World* symbols. Essentially $\vdash\psi$ is the statement in the metatheory that ψ is logically true, and it is this rather than ψ itself which we are trying to prove.

Unlike sequent calculi for weaker modal logics (Schütte [1968]), our sequent calculus leads to a very efficient proof procedure, as can be seen from the fact that the $\vdash$ law is an explicit definition of $\vdash$ in terms of $\vdash^* w$, and that the remaining twelve laws are essentially a contextual definition which eliminates all occurrences

of the $\vdash^{*w}$ symbol which do not occur immediately before an atomic sentence. Thus in a proof procedure based on this sequent calculus it makes no essential difference as to which of the thirteen laws is first applied. Furthermore it makes no essential difference to which subformula of the theorem being proven a law is first applied. All possible strategies of applying these laws, so long as they are applied as many times as possible will lead to the standard form described in section 3.2.

3.2 The Standard Form

If the rewrite rules given in section 3.1 are supplemented by enough laws of classical logic which when used as rewrite rules suffice to put the sentences of classical logic into skolemized prenex conjunctive normal form, then every sentence of our modal logic will be rewritten to an equivalent sentence in the following standard form:

(1) First a sequence of universal quantifiers (ie. skolem functions) consisting of:
   (a) object variable quantifiers: $\forall X$
   (b) proposition variable quantifiers: $\forall x$ (except those introduced by the $\vdash$ and $\vdash\vdash$ rules)
   (c) Propositional variable quantifiers introduced by the $\vdash$ rule. These quantifiers are essentially treated as world quantifiers as the (World*w) hypothesis is always kept next to the quantifiers: $(\forall w(World^* w) \rightarrow \ldots)$.

(2) Second a sequence of existential quantifiers (ie. unification variables) consisting of:
   (a) object variable quantifiers: $\exists X$
   (b) proposition variable quantifiers: $\exists x$ (except those introduced by the $\vdash$ and $\vdash\vdash$ rules)
   (c) Propositional variable quantifiers introduced by the $\vdash$ rule. These quantifiers are essentially treated as World quantifiers as the (World*w) hypothesis is always kept next to the quantifiers: $(\exists w(World^* w) \wedge \ldots)$.

(3) And finally a matrix consisting of

(a) a conjunction: $\wedge$

(b) of a disjunction: $\vee$ (ie. of sequents)

(c) of negated: $\sim$ or unnegated

(d) atoms consisting of an entailment symbol $\vdash^{*}w$ whose first argument is a variable quantified by a quantifier of type (1c) or (2c), and whose second argument is either (i) a variable quantified by a quantifier of type (1b) or (2b), or (ii) a non-logical atomic sentence containing no variables of type (1c) or (2c).

Schematically this standard form can be represented as:

$$\forall X\ \forall x\ \forall w(\text{World}^{*}\ w) \rightarrow \quad \exists X\ \exists x\ \exists w(\text{World}^{*}\ W) \wedge \quad \text{Matrix}$$

where the matrix is of the form:

$$\wedge\ \vee \{\sim\} \vdash^{*}_{w} \begin{cases} x \\ (\varphi\ x\ X) \end{cases}$$

where φ is a non-logical symbol.

The fact that sorted quantifiers can be pulled out of the matrix and skolemized can be justified by the following theorems of classical logic:

$\rightarrow\forall w$: $(\forall w(\text{World}^{*}w) \rightarrow (s \wedge (t \vee \varphi w))) \leftrightarrow (s \wedge (t \vee (\forall w(\text{World}^{*}w) \rightarrow \varphi w)))$

$\rightarrow\exists w$: $(\exists w(\text{World}^{*}w) \wedge (s \wedge (t \vee \varphi w))) \leftrightarrow (s \wedge (t \vee (\exists w(\text{World}^{*}w) \wedge \varphi w)))$

Skolem: $(\exists x\ \Pi x \wedge (\forall a\ \Pi a \rightarrow \varphi x\ a)) \leftrightarrow (\forall a\ (\forall x\ \Pi x \rightarrow \Pi(a\ x)) \rightarrow (\exists x\ \Pi x \wedge \varphi x(a\ x)))$

The $\rightarrow\forall w$ and $\rightarrow\exists w$ theorems depend on the truth of the theorem: $\exists w(\text{World}\ w)$. The "a" in "(ax)" of the Skolem theorem represents a skolem function.

3.3 Completeness and Expressibility

We consider the fragment of our modal logic which does not contain any propositional variables and where propositions are not objects. Given the standard form described in section 3.2 we can prove the completeness of this fragment relative to classical state logic. Classical state logic is a classical quantificational logic containing two distinct sorts, namely objects and worlds such that each non-logical symbol contains exactly one argument position which contains a world variable. This completeness proof is carried out merely by forming an isomorphism between expressions of our modal

logic and expressions of state logic by translating each atom $\vdash^{*} w(\varphi^{n} X_1 \ldots X_n)$ of our modal logic containing an n-ary non-logical symbol φ into an (n+1)-ary atom of state logic: $(\varphi^{n+1}\ X_1 \ldots X_n\ w)$.

We now state the Completeness Theorem:

Completeness Theorem

For every sentence (a) of this fragment of our modal logic, there is an equivalent sentence (b) of our modal logic such that there exists a sentence (s) of classical state logic which is isomorphic to (b). Therefore (s) is provable iff (b) and (a) are provable.

Using this isomorphism we may also obtain an Expressibility Theorem for our modal logic:

Expressibility Theorem

For every sentence (t) of state logic there is an equivalent sentence (s) of state logic in skolemized prenex conjunctive normal form such that there exists a sentence (b) of our modal logic which is isomorphic to (s).

The expressibility theorem shows that everything expressible in state logic is expressible in our modal logic.

4. Applications of Modal Logic to Artificial Intelligence

There are a number of potential applications of modal logic to Artificial Intelligence. We shall mention only two. In section 4.1 we shall describe its applications to metatheoretic reasoning, and in section 4.2 we shall use it to define various Epistemological concepts.

4.1 Application of Metatheoretic Reasoning

Our Modal Logic is potentially useful in any scientific theory, which one intends to implement as a program, and which involves both the expressions of a language, and the meanings of those expressions. The reason for this is that our modal logic, because it captures the notion of logical truth, when supplemented by a meaning function (Brown [1977d]) is both adequate to represent reasoning about expressions and their meanings, and is computationally better than

any other known system which is capable of such reasoning. The reason our modal logic is computationally better than other systems of metatheoretic reasoning such as model theory, is due to its inherent simplicity and the existence of a sequent calculus for it. Model Theory, which depends on set theory, by comparison is extremely complex.

One particular area in which such metatheoretic reasoning seems especially important, is in constructing theories of natural language understanding. It is easy to see that such a theory must involve both natural language expressions and their meanings. An initial proposal for the use of metatheoretic reasoning in natural language understanding is given in Brown and Schwind [1978a] and Brown and Schwind [1978b].

4.2 Applications to Epistemology

Our modal logic may also be used to define intensional concepts such as believe, and know. If we let (beliefs E) represent the beliefs of a robot E expressed as a conjunction of axioms, then we can say that a robot E believes something iff it is entailed by his beliefs:

$$(\text{Believe } E\ x) = \text{df} \vdash (\text{Beliefs } E)x$$

Furthermore we can say that a robot E knows something iff E believes it and it is the case.

$$(\text{Know } E\ x) = \text{df } (\text{Believes } E\ x) \wedge x$$

From these definitions it follows that Believe possesses properties of a deontic S5 modal logic, and Know possesses the properties of an S4 modal logic.

5. Conclusion

We have given an axiomization of a modal quantificational logic which captures the notion of logical truth. We have then defined a complete sequent logic and finally have described some applications of it to Artificial Intelligence.

References

Brown, F.M. (1976) "A Deductive System for Elementary Arithmetic" 2nd AISB Conf.Proc. Edinburgh

Brown, F.M. (1977a) "Doing Arithmetic without Diagrams" Artificial Intelligence vol. 8 pp 175-200

Brown, F.M. (1977b) "A Theorem Prover for Elementary Set Theory" 5th IJCAI MIT.

Brown, F.M. (1977c) "Towards the Automation of Set Theory and its Logic" DAI Research Report 34

Brown, F.M. (1977d) "The Theory of Meaning" DAI Research Report 35

Brown, F.M. and Schwind, C. (1978a) "Analysing and Representing Natural Language in Logic" 3rd AISB/ GI Conf.Proc.Hamburgh

Brown, F.M. and Schwind, C. (1978b) "Outline of an Integrated Theory of Natural Language Understanding" to appear as a DAI Research Report

Hughes, G.E. and Cresswell, M.J. (1968) An Introduction to Modal Logic, Methuen and Co.Ltd. London

Schütte, K. (1968) Vollständige Systeme Modaler und Intuitionistischer Logik Erg. d. Mathem. und ihrer brenzgebiete, Band 42, Springer-Verlag, Berlin

ANALYSING AND REPRESENTING NATURAL LANGUAGE IN LOGIC

F M Brown
University of Edinburgh
Edinburgh, Scotland

C B Schwind
Technische Universität München
München, B.R.D.

Abstract

We describe the basic idea of a theory of natural language understanding in which the meaning of natural language text and the laws for natural language analysis are both represented in logic.

1. Introduction

The basic idea of our theory (Brown and Schwind, 1978) of natural language understanding is that laws for the description of the syntax of natural language, laws for the translation of natural language into meanings, and the representation of those meanings are all written in a single representation language.

This representation language is Brown's (1978) modal quantificational logic supplemented by the meaning function described in Brown (1977). The reason we use this theory as our representation language is that it is computationally efficient, and is capable of representing the metatheoretic reasoning about expressions and their meanings which is necessary for the disambiguation of natural language text. We believe that a theory of this nature has important scientific and methodological advantages over other types of theories of natural language understanding in that we will be able to precisely and concisely state laws involving all aspects of the process of natural language understanding.

2. Formal Orientation

The syntax of our theory of natural language includes logical connectives such as and: $\wedge$, or: $\vee$, if ... then: $\rightarrow$, iff: $\leftrightarrow$, not: $\sim$, true: $\blacksquare$, and false: $\square$. It also includes logical quantifiers such as For all: $\forall$ and there exists: $\exists$; Equality: =, and description operators such as iota: ι and the which obey the laws:

$$\Psi(\iota x\ \Phi x) \leftrightarrow (\exists y\ (\forall x\ \Phi x \leftrightarrow x=y) \wedge \Psi y) \vee (\sim\exists y\ (\forall x\ \Phi x \leftrightarrow x=y) \wedge \Psi_{Nil})$$

$$(\text{the } x\ \Phi x) \quad =_{df}\ \iota x(\Phi x \wedge \mathcal{A} x)$$

where $\mathcal{A}$ is a linguistic function which we leave undefined.

Our theory includes modal connectives such as logical truth: $\vdash$, entailment: binary $\vdash$, and possibility: $\Diamond$, (Brown, 1978) and tense operators such as Present, Past, and Future, (Schwind, 1977, 1978c)

It also includes LISP like operators to form structural descriptive names of expressions from names of symbols. Names of symbols are formed merely by prefixing an accent sign to that sign. Note however that such names are constant symbols of our language. A name of an expression is formed by representing it as a list of the names of the sub-expressions occurring in it. A list such as $[x_1 \ldots x_n]$ is an abbreviation for (Cons x_1 ... (Cons x_n Nil) ...), where Cons is an ordered pair and Nil is not an ordered pair. Finally we have a function

M which produces the meaning of (almost) any expression (Brown, 1977).

3. Parsing and Translation

Parsing and translation laws are represented in our theory by non-logical laws of the form:

PT $(A\ xo\text{-}xn\ z) \leftrightarrow \exists x1\ \exists x2 \ldots \exists xn (A1\ xo\text{-}x1\ z1) \wedge (A2\ x1\text{-}x2\ z2) \wedge \ldots \wedge (An\ xn\text{-}1\ -\ xn\ zn)$

The xi are lists which contain various parts of the sentence which is being parsed. x-y denotes the difference list of x and y, e.g. ['the 'boy 'is 'handsome] - ['is 'handsome] = ['the 'boy]. The axiom PT is to be read as: xo-xn is an A and is translated into z iff xo-x1 is an A1 and is translated into z1 and x1-x2 is an A2 and is translated into z2 and ... xn-1 - xn is an An and is translated into zn.

Example:

$(S\ xo\text{-}[\,][z\ zo]) \leftrightarrow \exists x1\ \exists x2 (PN\ xo\text{-}x1\ zo) \wedge (VBE\ x1\text{-}x2) \wedge (ADJ\ x2\text{-}[\,]\ z)$

xo-[] is an S (sentence) and is translated into [z z0] iff x0-x1 is a PN (Proper Name) and is translated into z0 and x1-x2 is a VP (verb, to be) and x2-[ ] is an ADJ (adjective) and is translated into z .

Using this axiom together with the following axioms:

$(PN\ [x.xo]\text{-}xo\ z) \leftrightarrow x='John \wedge z='JOHN27$

$(VBE\ [x.xo]\text{-}xo) \leftrightarrow x='is$

$(ADJ\ [x.xo]\ xo\ z) \leftrightarrow x='handsome \wedge z='HANDSOME$

we can parse and translate the following sentence:

(1) (S'(John is handsome)- [] [z zo])

(2) $\exists x1\ \exists x2$ ((PN '(John is handsome)-x1 zo) $\wedge$ (VBE x1-x2) $\wedge$ (ADJ x2-[] z))

(3) $\exists x2$ ((PN ['John.' (is handsome)] -'(is handsome) zo)
(VBE ' (is handsome)-x2) $\wedge$ (ADJ x2-[] z))

(4) ('John = 'John) $\wedge$ (zo = 'JOHN27) $\wedge$ (VBE ['is.'handsome] -'handsome)
(ADJ ['handsome.[]] -[] z)

(5) zo = 'JOHN27 $\wedge$ 'is='is $\wedge$ 'handsome='handsome $\wedge$ z='HANDSOME

(6) zo = 'JOHN27 $\wedge$ z='HANDSOME

Hence (S '(John is handsome)-[ ] ['HANDSOME 'JOHN27])

We give in figure 1 an example grammar expressed in our logical theory. This grammar is a small part of Schwind's (1978a, 1978b, 1978d) attributed feature grammar for English. A similar grammar for German is described in Schwind (1977).

Although these grammars translate natural language text into logical expressions as did Montague's (1974) natural language theory, our theory is superior in that it is not based on the linearity assumption that is inherent in Montague's theory which forces the use of higher types of functions as meanings of words. That is, we do not assume that most words have a meaning in and of themselves, and that the meaning of a text is obtained by a series of functional compositions of

the meaning of the words occurring in it, as did Montague. Rather we assume that many words have meaning only in certain syntactic contexts, as did Frege (Dummett, 1973) in his doctrine on incomplete expressions. This allows us to dispense with the whole edifice of Montague's higher order intensional logic with its computationally inefficient, if not impossible, artificial set theoretic constructions of word-meanings, and to use instead Brown's (1977, 1978) simple and computationally efficient first order modal logic and theory of meaning.

4. Interaction with World Knowledge

One of the main problems of syntactic analysis is the ambiguity of natural language. For example, there are two different syntactically correct parsings of each of the following sentences:

(1) The tree in the garden which has cones is pretty.

(2) The tree in the garden which has roses is pretty.

It should be clear that for semantic reasons we would like to parse the second sentence as parsing (B) and never as parsing (A) because it is false to claim that trees have roses. (See figure 2).

From figure 2 we see that the NG which is nearest to the root of the parsing tree is ambiguous; namely, in (A) NG is an NG3 whereas in (B) it is an NG2. Clearly then what we need to do is to modify the NG axiom of figure 1 so as to state that if both NG3 and NG2 parsings are possible then we want to reject one of them on semantic grounds. That is we want to say that the parsing accepted by the NG law should be the parsing produced by the NG2 law only if either there is no parsing produced by the NG3 law or if there is one and the meaning of the logical expression which is the translation of the natural language expression parsed by the NG2 law is consistent with our knowledge of the world and the second meaning is not. This would allow us to reject the parsing which claimed that a tree has roses since we would expect:

$\sim \exists x \exists y$ TREEx $\wedge$ ROSEy $\wedge$ HASxy to be deducable from general knowledge.

We modify the NG law as follws:

$$(\text{NG xo-xl vo-vl y z}) \leftrightarrow$$

$$((\text{NG1 xo-xl vo-vl y z})$$

$$\vee((\text{NG2 xo-xl vo-vl y z}) \wedge (\forall z1(\sim\exists y(\text{NG3 xo-xl vo-vl y z1})) \vee \Diamond(w \wedge \sim M[\text{'}\sim z]) \wedge \sim\Diamond(w \wedge \sim M[\text{'}\sim z1]\ )))$$

$$((\text{NG3 xo-xl vo-vl y z}) \wedge (\forall z1\ (\sim\exists y(\text{NG2 xo-xl vo-vl y z1})) \vee \Diamond(w \wedge \sim M[\text{'}\sim z]) \wedge \sim\Diamond(w \wedge \sim M[\text{'}\sim z1]\ ))))$$

The symbol w represents a conjunction of the nonlogical axioms expressing facts of the "real world". M is the meaning function described in Brown (1977), $\sim M[\text{'}\sim z]$ is the meaning of the existential closure of z.

REFERENCES

Brown, F. M. (1977) "The Theory of Meaning" D.A.I. Research Report 35.

Brown, F. M. (1978) "A Sequent Calculus for Modal Quantificational Logic" 3rd AISB/GI Conf. Proc.

Figure 1. The example Grammar

```
(SENT xo-x1 vo-v1 z) ↔ (SENT1 xo-x1 vo-v1 z) ∨ (SENT2 xo-x1 vo-v1 z)
                     ∨ (SENT3 xo-x1 vo-v1 z)
(SENT1 xo-x3 vo-v1 [zo(subst [z2 y] for * in z1)]) ↔
∃ x1 ∃x2((NP xo-x1 vo-v1 y z1) ∧ (VBE x1-x2 zo) ∧ (ADJ x2-x3 z2))
(SENT2 xo-x3 vo-v2
[(cadr zo) (subst(subst[(car zo)y1 y2] for * in z2) for * in z1)]) ↔
∃x1∃x2 v1((NP xo-x1 vo-v1 y1 y1 z1) ∧ (VT x1-x2 zo) ∧ (NP x2-x3 v1-v2 y2 z2))
(SENT3 xo-x4 vo-v2
[?[zo(subst(subst (car z3)y1 y2] for * in z2) for * in z1)]]. ↔
∃x1∃x2 x3 v1((VDO xo-x1 zo) ∧ (NP x1-x2 vo-v1 z1) ∧
             (VT x2-x3 z3) ∧ (NP x3-x4 v1-v2 y2 z2))
(CLAUSE xo-x3 vo-v1 y1 [(cadr zo) (subst[(car zo) y1 y2] for * in z1)]) ↔
∃x1∃x2((RELPRON xo-x1) ∧ (VT x1-x2 zo) ∧ (NP x2-x3 vo-v1 y2 z1))
(NP xo-x1 vo-v1 y z) ↔ ((NP1 xo-x1 vo-v1 y z) ∨ (NP2 xo-x1 vo-v1 y z) ∨
                        (NP3 xo-x1 vo-v1 y z) ∨ (NP4 xo1x1 vo-v1 y r))
(NP1 xo-x1 vo-v1 y [' y[z '∧ *]]) ↔ (NG xo-x1 vo-v1 y z)
(NP2 xo-x2 vo-v1 y [(car z1) y [z2 (cdr z1)*]]) ↔
∃x1((DETQ xo-x1 z1) ∧ (NG x1-x2 vo-v1 y z))
(NP3 xo-x2 vo-v1 ['the y z]*) ↔
∃x1((DETD xo-x1) ∧ (NG x1-x2 vo-v1 y z))
(NP4 xo-x1 [y.v1]-v1 y [z y *]) ↔ (PRON xo-x1 z)
(NG xo-x1 vo-v1 y z) ↔
((NG1 xo-x1 vo-v1 y z) ∨ (NG2 xo-x1 vo-v1 y z) ∨ (NG3 xo-x1 vo-v1 y z))
(NG1 xo-x1 [y.v1] -v1 y [z y]) ↔ (N xo-x1 z)
(NG2 xo-x2 vo-v2 y1 [z1 '∧ z2]) ↔
∃x1∃v1 ((NG xo-x1 vo-v1 y1 z1) ∧ (PP x1-x2 v1-v2 y1 y2 z2))
(NG3 xo-x2 vo-v2 y1 [z1 '∧ z2]) ↔
∃ x1∃v1 ((NG xo-x1 vo-v1 y1 z1) ∧ (CLAUSE x1-x2 v1-v2 y1 z2))
(PP xo-x2 vo-v1 yo y1(subst [zo yo y1] for * in z1))
∃x1((PREP xo-x1 zo)∧(NP x1-x2 vo-v1 y1 z1))
```

Lexicon

```
(DETD [x.xo]-xo) ↔ x='the
(DETQ [x.xo]-xo z) ↔ (((x='a ∨ x='some)∧ z=['∃ '∧ ]) ∨
                      ((x='all∨ x='every)∧ z=['∀ '→)))

(N[x.xo]-xo z) ↔
(((x='tree ∨ x='trees) ∧ z='TREE) ∨ ((x='garden ∨ x='gardens) ∧ z='GARDEN)
∨((x='rose ∨ x='roses) ∧ z='ROSE)∨((x='cone ∨ x='cones) ∧ z='CONE))
(ADJ[x.xo]-xo z) ↔ ((x='pretty ∧ z='PRETTY) ∨ (x='red ∧ z='RED))
(PREP[x.xo]-xo z) ↔ (x='in ∧ z='IN)
(VBE[x.xo]-xo z) ↔ ((x='is ∧ z='PRESENT) ∨ (x='was ∧ z='PAST))
((VDO x.xo -xo z) ↔ (((x='do ∨ x='does) ∧ z='PRESENT) ∨ (x='did ∧ z='PAST))
(VT[x.xo]-xo z) ↔
(((x='own ∨ x='owns) ∧ z=['OWN 'PRESENT]) ∨ (x='owned ∧ z=['OWN 'PAST])∨
 ((x='grow ∨ x='grows) ∧ z=['GROW 'PRESENT]) ∨ [(x='grew ∧ z=['GROW 'PAST]) ∨
 ((x='had ∨ x='has) ∧ z=['HAS 'PRESENT]) ∨ (x='had ∧ z=['HAS 'PAST]))
(RELPRON [x.xo]-xo) ↔ (x='which ∨ x='that ∨ x='who)
(PRON [x.xo]-xo z) ↔ ((x='somebody ∧ z='∃) ∨ (x='everybody ∧ z='∀))
```

Brown, F. M. & Schwind, C. (1978) "Outline of an Integrated Theory of Natural Language Understanding" to appear as a D.A.I. Research Report

Dummett, M. (1973) Frege, Philosophy of Language, Gerald Duckworth & Co., Ltd.

Montague, R. (1974) Formal Philosophy, Selected Papers of Richard Montague, ed. R.H. Thomason, New Haven & London, Yale University Press

Schwind, C. B. (1977) Ein Formalismus zur Beschreibung der Syntax und Bedeutung von Frage-Antwort-Systemen. TUM-INFO-7710, (Ph.D. Thesis).

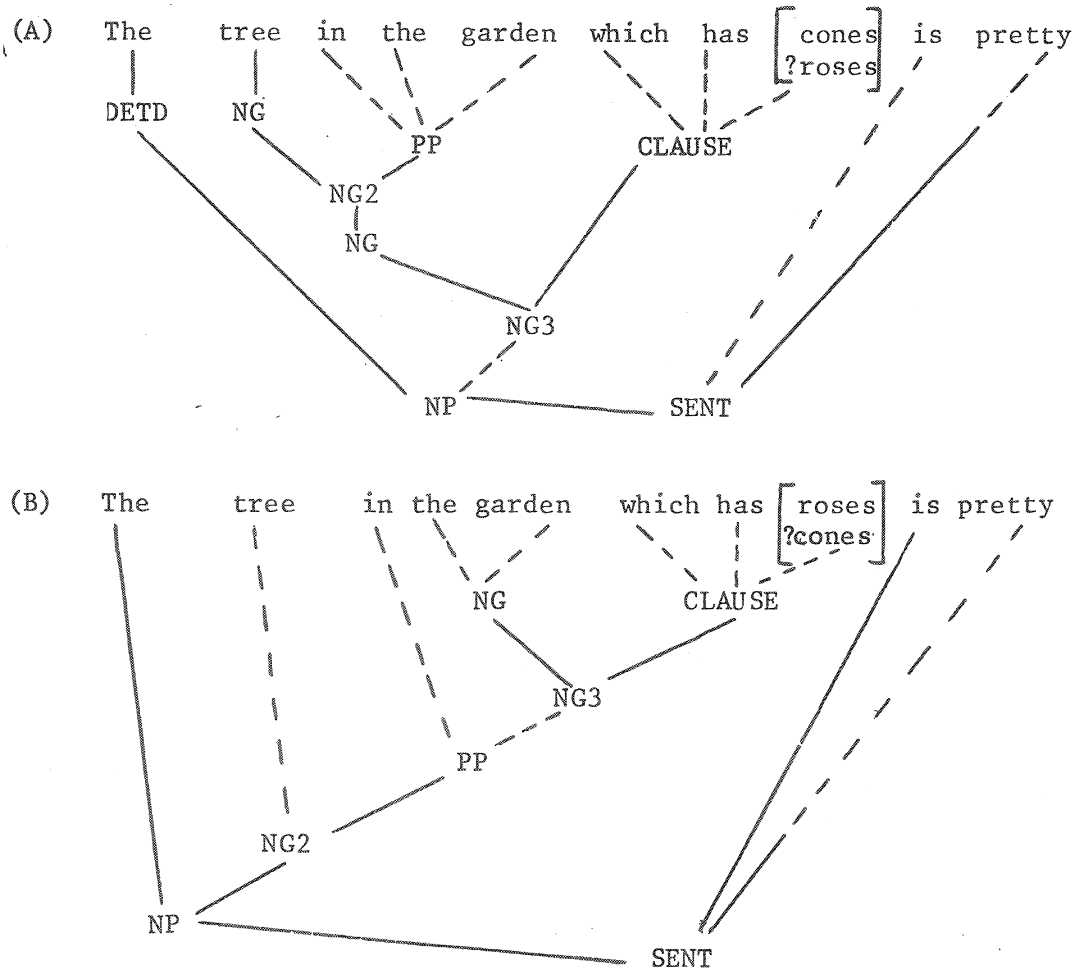
Schwind, C. B. (1978a) "A Formalism for the Description of Question Answering Systems", to appear in Natural Language Communication with Computer, Springer Verlag, Heidelberg.

Schwind, C. B. (1978b) "A Theory of Translation of Natural Language Texts into Logical Formulae", TUM-INFO-7806.

Schwind, C. B. (1978c) "Representing Actions by a Modal Tense Logic", 3rd AISB/GI Conf. Proc., Hamburg.

Schwind, C. B. (1978d) "Translation of Natural Language Texts into State Logic Formulae", to appear in 5th International Conference on Applied Linguistics, Conference Proceedings.

Figure 2. Syntactically ambiguous sentences.



KNOWLEDGE ABOUT KNOWLEDGE: MAKING DECISIONS IN MECHANICS PROBLEM SOLVING

Alan Bundy, George Luger,
Chris Mellish, Martha Palmer
Department of Artificial Intelligence,
University of Edinburgh,
Edinburgh, UK.

1. Abstract

In October 1975 we started a three year SRC supported project on "A Program to Solve Mechanics Problems Stated in English". This project is now beginning its third year and many of its initial objectives have been achieved. In this paper we briefly review the progress that has been made and discuss some of the more promising areas of research that we have uncovered.

The structure of the paper is as follows:

- Section 2 The current state of the project
- Section 3 An annotated example
- Section 4 Towards a computational logic for natural reasoning
- Section 5 Investigations of Syntax-semantics interaction

Conclusion and References

2. The current state of the project

The original objective of our project is summed up in the following quotation from the 1975 grant application:

> "Our objective then, is to write a program which can solve Mechanics problems, of the kind given as examples in Humphrey (1957) p1-90. We will concentrate on writing a program which can extract equations, given a surface level meaning representation obtained from the statement of the problem in English. If we make suitable progress on this, we would like to extend the program, so that it would accept problem statements in English."

This program, entitled MECHO, now solves a number of mechanics problems. The program centres around an algorithm for extracting equations based on Marples' (1974) study of Cambridge Engineering Students (for further details, see Bundy et al 1976a). This algorithm in turn calls on various inference rules to draw conclusions from the input meaning representation. The equations

extracted can be fed to an equation solving program written by R.K.Welham.

As might be expected, our current descriptive theory differs substantially from our ideas in 1975. The three stages of meaning representation (surface, deep and deep with quantities) have been merged into one, which is gradually added to in the course of problem solving. We now make use of far more high-level descriptions than envisaged in 1975 and have single predicates and associated schemas (Frames) for whole pulley systems (Luger and Bundy 1977b) and for particles in motion (Bundy 1977a).

The PROLOG programming language (Warren 1977) was chosen for all our programs and has proved quite successful. It has provided the important tools of an assertional database, pattern-directed invocation and a search facility as well as extra-logical evaluable predicates to modify the simple backtracking control structure. Our experiences with PROLOG are recorded in Bundy 1976b and Bundy and Welham 1977d.

Our original approach to problems of search control consisted in carefully designing and debugging inference rules within the basic PROLOG inference system. As more problems were tackled, generalisations were noted and some of the control information was incorporated in the inference system. This is leading to the design of a computational logic for natural reasoning based on our experience of mechanics problem solving (see section 4).

It was usually possible to rely on our own intuitions and experience of mechanics problem solving to design inference rules and descriptive terms. Occasionally, however, it was useful to investigate disputes or illuminate difficulties by examining protocols of expert problem solvers. The results of these protocol analyses are described in Luger 1977a.

We have progressed further than originally intended with the extension to allowing English input to the program. Since we were unable to find a consensus on the kind of meaning representation that might be output by a natural language "front end", we were forced to develop the natural language processing and problem solving programs in parallel. Our first programs took as input a syntactic parse from Soul's (1975) existing parser and produced a suitable meaning representation (see Stone 1976). Later efforts to produce a unified program have introduced interesting issues of syntax-semantics interaction (Palmer 1977 and Mellish 1977) and demonstrated the value of studying natural language processing and problem solving in the same system (see section 5).

The three major problem areas that have been concentrated on so far are those involving pulleys, the motion of particles on complex paths and the motion of particles moving under constant acceleration. Each of these is described below with an indication of the representational issues involved and an account of one of

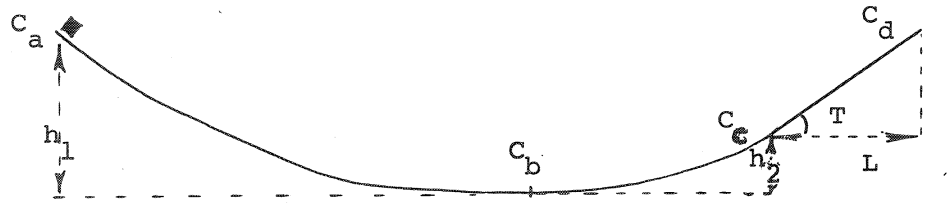
the problems solved by the MECHO system. Most of these problems have been translated by hand into Predicate Calculus assertions suitable for input to the program, although some have been processed by the natural language programs.

A. Pulleys (from Humphrey, p75)

"Two particles of masses 6 and 10lbs are connected by a light string passing over a smooth pulley. Find their common acceleration and the tension in the string."

Pulley system problems were used to develop the descriptive theory especially the use of schemas to describe high-level objects like pulley systems. Also the problem above was the main working example for the natural language processing programs.

B. Motion of Particles on Smooth Complex Paths (de Kleer, 1975)



The particle starts from rest at C_a; given h_1, h_2, L and T will it reach point C_d?.

This problem was used to explore the representation of motion and the technique of hypotheses making and testing. These "roller coaster" problems rely heavily on diagrams for their description, and no attempt was made to input either diagrammatic or English descriptions of them.

C. Constant acceleration problems (Palmer and Snell, p20)

"The distance between two stations is 2000 yds. An electric train starts from rest at one station with a uniform acceleration of a1 ft/secs^2. It comes to rest at another 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."

These problems were used as a vehicle for exploring the representation of time. This example motivated a package for converting all units into a compatible set (e.g. yards and minutes above into feet and seconds). It also provided some ammunition for the natural language processing, i.e. how to parse "starts from rest" etc. recognizing the referent to "It" and associating 2000 yards with the path of the train.

3. Annotated Example

To give a clearer picture of how the system acts as a whole and how the individual parts operate, we will now briefly describe

the steps it goes through for a concrete example. The example we have chosen is a simple pulley problem:

> Two particles of mass b & c are connected by a light string passing over a pulley. Find the acceleration of the particle of mass b.

The natural language analysis looks at each clause separately, interpreting it in two phases. As the clause is read, the syntactic structure is established and the objects referred to are identified. Then the meaning of the verb is consulted to determine how these objects are to be used to derive the meaning of the clause as a whole.

In the first clause, the word "two" starts the parsing of a noun phrase, the result being assigned the subject role because of its initial position in the sentence. Two new objects are created and predicated to be particles. The modifier "of mass b and c" is parsed, with a general-purpose routine distributing the masses - one to each particle. Then semantic routines scrutinise the particles to ensure their suitability to have the property of mass, creating new objects for the masses and ascribing them to the particles. The next major sentence constituent is the main verb group, which reveals the main verb "connect" as well as other information (for instance, that the sentence is in the passive voice). The preposition "by" initiates parsing of a prepositional phrase, and a new object, asserted to be a string, is created. This is asserted to have zero mass (it is "light") and takes the subject role in the reduced relative clause "passing over...". This second clause is now interpreted almost exactly as if it were the complete sentence "The string is passing over a smooth pulley", the main verb being "pass" and a pulley being created to fill the prepositional phrase slot. The full stop signals the end of all current clauses, which are now interpreted by semantic routines. These use syntactic roles like "logical subject" and "logical object" to access the appropriate participants in the relationships implied by the verb. The "pass" routine identifies its main participants as the string and the pulley and suggests contact between the midpoint of the string and the pulley (which is assumed to have no extent). Then the "connect" routine predicates contact between each particle and an appropriate contact point of the string, one particle being paired with each end. Notice that during this semantic analysis new objects, such as the ends of the string, have been created to satisfy the requirements of the meaning. Also the configuration is seen to satisfy sufficient constraints of the standard pulley system to cue in a pulley system schema.

The second sentence is seen to be an imperative when the first word is read, and the remaining work involves finding the logical object. The definite article in "the particle" reveals that it is referring to an object already known. Although there are two possible particles, there is only one known mass quantity

with measure b and only the first particle has this mass. The definite article in "the acceleration" cannot be interpreted in the same sense as this, since the values of functions of objects (like accelerations) can be referred to by definite phrases without there being previous specific knowledge of them. However, in this case the acceleration has already been introduced by the pulley system schema, and so it is not necessary to create a new object. Finally the second sentence is interpreted as a whole. The meaning of "find" in the imperative involves marking certain quantities as "sought" for the equation solver. With the reaching of the end of the problem, all other quantities that have been introduced and specified can be marked "given".

The following clauses are the result of the natural language analysis:

```
isa(particle, p1), isa(particle, p2), isa(string, s1)
isa(pulley, pull), end(s1, end1, right), end(s1, end2, left)
fixed_contact(end1, p1, period1), fixed_contact(end2, p2, period1),
midpt(s1, midpt1), fixed_contact(midpt1, pull, period1)
mass(p1, mass1, period1), mass(p2, mass2, period1), mass(s1, zero,
                                                          period1)
coeff(pull, zero), measure(mass1, b), measure(mass2, c),
accel(p1, a1, 270, period1), period(period1),
sought(a1), given(mass1), given(mass2),
```

in addition the following schema is cued:

```
pullsys_stan(sys, pull, s1, p1, p2, period1).
```

The backward reasoning Marples Algorithm then attempts to find an equation that expresses the sought quantity, a1, in terms of the given quantities b, c, and g (the gravitational constant). a1 is investigated and information is extracted which focusses the Marples Algorithm and narrows the range of equations to be considered. In this case the only equations considered are those obtained by resolving forces about the particle p1 and p2. Only one of these is needed at this stage, namely the resolution of forces about p1:

$$-b.g + tension1 = b.a1$$

This solves for a1 but introduces a new unknown tension1, the tension in the string.

This tension was not mentioned in the original problem statement, but was introduced by the pulley system schema in the same way as the accelerations of the particles. Other information introduced by the pulley system schema includes, for instance, that the string is divided into two straight segments with inclinations of 90° and 270° respectively. Thus the schemas are used to fill in "gaps" between what is given in the problem statement and what is needed to solve the problem.

The other "gap filling" mechanism is the backwards inference process initiated by equation extraction. In this example backwards inference is responsible for deducing that the tension of each of the string segments is tension1 and for working out the combined forces acting on the particles.

The Marples Algorithm now tries to find an equation which expresses tension1 in terms of a1, b, c, and g. Attention is focussed on resolving forces about p1, p2 or the pulley. Resolving about p1 is rejected because that equation has already been used. Resolving about p2 is preferred over resolving about the pulley because it introduces no new unknowns. The resulting equation is:

c.g - tension1 = c.a1

Before being solved the equations are checked to see that they are all in the same units. Solution in this case is very simple and merely involves eliminating tension1 to give

a1 = g.(c-b)/(c+b)

4. Towards a Computational Logic for Natural Reasoning

One of the original aims of the project was to see how domain specific knowledge could be used to control inference in a semantically rich domain. Inference rules were designed with care, to avoid superfluous search wherever possible. Program traces were examined to spot false trails and inference rules debugged to avoid them in the future. Eventually these rules were riddled with ad hoc control information.

It then became clear that much of this control information could be generalized and that it would then cease to be domain specific and become applicable to any inferential system. The two specific generalizations that we describe below are: (i) the exploitation of function properties (Bundy 1977c) and (ii) the use of similarity classes (Bundy 1978). We hope that this gradual process of generalization will lead to a computational logic for natural reasoning. That is, an inferential mechanism containing built in control primitives which have already proved their usefulness.

The first generalization is the exploitation of function properties. One of the properties of functions is that they give unique values. This uniqueness can be used both (i) to trap unachievable goals before they are called and (ii) to prune subsequent search after a first value has been found. For instance, if it is known that f(a) = b then it is not possible to prove that f(a) = c where $b \neq c$. In many cases it is possible to trap the unachievable goal f(a) = c before it is called. Similarly if the value of f(a) is requested and shown to be b it is silly to request alternative values of f(a) on backtracking and the possibility can be pruned from the search tree.

It might be felt that functions occur only in mathematical reasoning. In fact they are common in everyday reasoning but often go unnoticed. For instance:

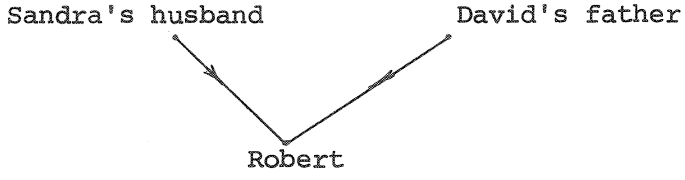
At is a function from objects and times to locations
Final is a function from periods of time to their final moments
Motherof is a function from mammals to their mothers.

In our inference system it is only necessary to specify that a particular relation is a function from some arguments to others for the appropriate control regime to be brought to bear. A relation can even be a function in more than one way e.g. the relation Timesys between a period and its initial and final moments can be a function in three ways.

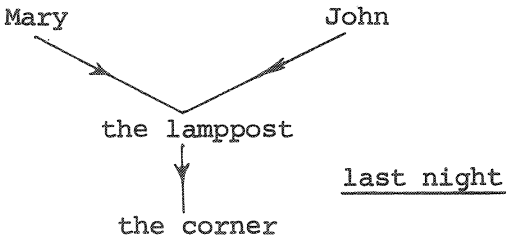
The second generalization is the use of similarity classes. This was a technique originally developed to control the equality relation. In fact, it is applicable to any relation with transitive, symmetric and reflexive properties and again this is a far wider class of relations than is generally appreciated. The idea is to put similar objects in classes with a distinguished element (the root), so that similarity between two objects can be tested by looking to see whether they are in the same class (i.e. point to the same root). This test is highly efficient compared to the infamous explosive search properties of the transitive and symmetric laws.

The standard way to represent similarity classes is by trees, with the root as distinguished element. Below are some sample trees for (i) equality, (ii) the sameplace relation (two objects are in the same place) and (iii) the factor relation (giving the conversion factor between two units of the same dimension).

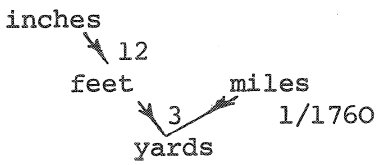
(i) equality



(ii) sameplace



(iii) factor



Slight modifications of the original techniques are needed for some relations. For instance, in the case of sameplace a different set of classes is needed for each time period or moment. For factor it is necessary to label the arcs with conversion factors. These need to be multiplied together as the arcs are traversed, to find, say, the conversion factor between inches and miles.

The MECHO program contains a generalization of the similarity class machinery and appropriate relations can be defined simply in terms of it.

5. Investigations of syntax-semantics interaction

In the area of natural language processing, one of our main aims is to study the process-control issues of efficient natural language parsing - the type of decisions that have to be made and the ways that syntactic and semantic processes can interact to best advantage. It is the process of reference evaluation that has concerned us mainly.

Winograd (1972) has shown that evaluating references early in the parsing can be useful in resolving syntactic ambiguity. Thus in the sentence:

Put the pyramid on the block in the box,

which is apparently ambiguous, only one reading is acceptable in a context where there are no pyramids on blocks. The absence of a referent to 'the pyramid on the block' is used to ensure that another interpretation is taken.

However, there is not always enough information in a noun phrase to determine the referent uniquely (for instance, if it is a pronoun, this is rarely so). Thus Winograd was forced into adopting discourse heuristics to make premature decisions (for most definite noun phrases) or carrying forward all possibilities in the hope that the rather crude semantic marker lists in the verb meaning will eliminate some (in the case of pronouns).

The system that we are investigating involves considering reference evaluation as an incremental process, with references becoming progressively more instantiated as the analysis proceeds. In this, pronouns are treated in the same way as other definite noun phrases, as the same mechanism is used to express both definitional information and 'semantic checks'. Both types of semantic constraints are interpreted as filters on possible referents and may be enforced at any time in the analysis, with the result that there can be a close interaction between syntactic and semantic processes. Reference evaluation still provides a check on syntactic hypotheses because a filtering process which has eliminated all the possibilities for a referent causes the abandonment of the current hypotheses. Moreover, in this framework, there is much less risk of carrying out extensive syn-

tactic manipulations on the basis of faulty reference evaluations.

An important feature of the system developed is that all semantic tests are in terms of actual referents and not in terms of the structure of referring phrases. Thus we are able to express a much wider range of constraints than can be captured by the use of traditional semantic markers of the Katz and Fodor (1964) type. Those of the simplest type involve single references and express ideas like 'an object cannot be a part of itself'. More complicated constraints impose dependencies between the possible values of several references. Thus the constraint that two objects be in contact causes interactions between the two evaluation processes - further information relevant to one may have repercussions for the value of the other. A great deal of knowledge of possible physical configurations cannot be expressed with a mechanism like semantic markers but requires something of this type. An investigation into how to make the best use of this powerful system will be an important further step in our work on mechanics problems.

As a simple example of how this kind of knowledge can help resolve ambiguity, consider the following problem:

> "AB is a uniform rod, of length 8a, which can turn freely about the end A, which is fixed;
> C is a smooth ring, whose weight is twice that of the rod, which can slide on the rod and..."

In this example, there is apparent ambiguity as to whether the final relative clause ("which can...") should be attached to "a smooth ring" or to "the rod". A human being who reads this sentence and attempts to visualise the scene has no difficulty with this problem. In order to resolve it mechanically, it suffices to notice the fact that an object cannot slide on itself. This information ensures that the phrase cannot be attached to the rod and thus must qualify the ring. However, the restriction needed (the irreflexivity of a particular sliding relation) cannot be expressed in terms of semantic markers.

There is a danger that the number and complexity of such semantic restrictions could be completely unmanageable, but several factors combine to ease this problem. Firstly, there is no reason why restrictions should have to be immediately verifiable - in general, deductions may be necessary to satisfy them. This means that similar restrictions can share pieces of deductive machinery; moreover in some cases complex restrictions can be pieced together from more primitive tests. Secondly, many restrictions express mathematical properties of particular relations and so are generalizable. These involve functional properties between relation arguments (c.f. section 4) and properties like symmetry and transitivity (e.g., the relation of 'support' is asymmetric and transitive). It is interesting that these are the kinds of properties that we are

exploiting in the quest for general mechanisms of search control. There seems to be a close similarity between controlling deduction by pruning inappropriate goals from the search tree and resolving ambiguity by avoiding consideration of semantically inappropriate hypotheses. Thus, in particular, the mechanisms we have developed to exploit function properties and similarity classes for search control may well be of direct use to the natural language programs. Conversely, mechanisms that have suggested themselves for the natural language tasks (such as the particular use of irreflexivity shown above) are likely to provide ideas for general search control techniques.

Conclusion

We hope that this paper has given an understandable account of the current state of the MECHO project and some of the more interesting issues that we are following up. Although our work covers a range of subjects that are not commonly studied in close conjunction, there is an underlying theme running through the whole project. This is the idea that only through carefully establishing exactly where and how individual pieces of knowledge should be used to best effect can one construct a system which performs an intellectual task of any complexity. Even in the relatively simple domain of mechanics problem solving, extremely sophisticated techniques have to be introduced in order to come close to the expert human's ability to make decisions and discard false trails and faulty hypotheses. It is our belief that a significant number of the ad-hoc solutions produced for specific problems can be generalized and that there are useful domain-independent principles of search and process control to be found. One of the most promising sources of relevant generalizations seems to involve the province of second-order relations like functionality, transitivity and irreflexivity and it is in this direction that our investigations are currently proceeding.

References

Bundy, A., Luger, G. and Stone, M., 1975. "A Program to Solve Mechanics Problems Stated in English", SRC Grant Application.

Bundy, A., Luger, G., Stone, M. and Welham, R., 1976a. "MECHO: Year One", Procs. of the 2nd AISB Conference, ed. Brady,M., Edinburgh, p94-103, also DAI Research Report No.22.

Bundy, A. 1976b. "My Experiences with PROLOG", DAI Working Paper No. 12.

Bundy, A. 1976c. "The Role of Inference in the Solving of Mechanics Problems", DAI Working Paper No. 16.

Bundy, A. 1977a. "Will it reach the top? Prediction in the Mechanics World", Artificial Intelligence Journal (forthcoming), also DAI Research Report No. 31.

Bundy, A. 1977b. "Can Domain Specific Knowledge be Generalized?", (Short Note), p496, Procs. of the 5th IJCAI, ed. Reddy,R., Cambridge, Mass.

Bundy, A. 1977c. "Exploiting the Properties of Functions to Control Search", DAI Research Report No. 45.

Bundy, A. and Welham, R.K. 1977d. "Utility Procedures in PROLOG", DAI Occasional Paper No. 9.

Bundy, A. 1978. "Similarity Classes", DAI Working Paper No.25.

de Kleer, J. 1975. Qualitative and quantitative knowledge in classical mechanics, MIT AI Lab. Report AI-TR-352, Cambridge, Mass.

Humphrey, D. 1957. "Intermediate Mechanics, Dynamics", Longman, Green & Co., London.

Katz, J.J. and Fodor, J.A. 1964. "The Structure of a Semantic Theory" in J.A. Fodor and J.J. Katz (Eds.) "The Structure of Language", Engelwood Cliffs, New Jersey: Prentice Hall.

Luger, G. 1977a. "The Use of Protocols and the Representation of Semantic Information in Pulley Problems", DAI Research Report No. 36.

Luger, G. and Bundy, A. 1977b. "Representing Semantic Information in Pulley Problems", p500 (Short Note), Procs. of the 5th IJCAI, ed. Reddy, R., Cambridge, Mass.

Marples, D. 1974. "Argument and technique in the solution of problems in mechanics and electricity, CUED/C - Educ/TRI, Dept. of Engineering Memo, University of Cambridge.

Mellish, C. 1977. "Preliminary Syntactic Analysis and Interpretation of Mechanics Problems Stated in English", DAI Working Paper.

Palmer, A.H.G. and Snell, K.S. 1956. "Mechanics", Univ. of London Press Ltd., London.

Palmer, M. 1977. "Where to Connect? - Solving Problems in Semantics", DAI Working Paper No. 22.

Soul, M. "Parsing and reference determination", unpublished Ph.D. Thesis, Essex.

Stone, M. 1976. "PAT - Pulleys and Things", DAI Working Paper No. 18.

Warren, D. 1977. "Implementing PROLOG - compiling predicate logic programs", Vol. 1 and Vol. 2, DAI Research Report No. 39 and 40, Edinburgh.

Winograd, T. 1972. "Understanding Natural Language", Academic Press.

A THEOREM PROVER DATING A SEMANTIC NETWORK

Dennis de Champeaux
University of Amsterdam
Amsterdam-Netherlands

0. Abstract

Ordinary predicate calculus (p.c.) is equiped with sorts, an extension of types, allowing a concise formulation of problems. A semantic network can easily provide the sort-part of the input of a theorem prover based on a sorted p.c. Experimental results are reported of such a theorem prover.

Keywords and Phrases:

Typed predicate calculus, sorts, theorem proving, N_1-resolution, semantic networks.

Contents:

0. Abstract
1. Motives
2. Sorts
   2.1. Definition of sorts
   2.2. Consequences for unification and resolution
3. Implementation of sorts in Cogito
4. Experimental results
   4.1. Comparison with other theorem provers
   4.2. Using characteristic predicates
5. Conclusions
6. References

1. Motives

Since 1965 a great amount of effort has been invested in theorem proving along the line of the resolution theory. The efforts aimed at refining the resolution rule by proposing constraints that could reduce the redundancies in the proof process, which all too often cluttered up computer memory before finding a solution. Although considerable savings were realized at each newly introduced constraint we have to face the fact that the current resolution based theorem provers are easily surpassed by the human deductive capabilities.
It is a reasonable conjecture that basically this situation will not be improved by adding new constraints. The bottle neck is that resolution theory is insensitive for particular properties of the search space which is associated with a collection of problems. A related topic is the difficulty to accumulate experience to be extracted from a derivation tree and to be used in subsequence proof efforts.

As a result some people threw away the general purpose approach as well as resolution theory and started writing ad-hoc deductive programs. It must be admitted that their performance is fine in a limited domain, but the necessity to have a great number of deductive specialists around, in order to approximate general purpose deductive power, created obscure interaction troubles when a specialist must be added, deleted or changed.

Automatic experience accumulation also isn't obvious.
Our position in this controversy is a rejection of the ad-hoc specialist approach. We strive to extend a general purpose techniques such that problem collection pecularities can be accounted for. Several non mutually exclusive options offer themselves, which we will formulate in the framework of resolution theory.
1. One can introduce a problem family sensitive heuristic function which imposes an order on the collection of resolutions which may be performed. When heuristic functions are only built up from some set of base functions, in say polynomial constructions, one may even imagine that experience accumulation lurks behind the corner.
2. One can add to a problem specification a model, a knowledge collection and an interpretation function (Gelernter)[2], with which literals to be resolved can be constrained to those belonging to clause pairs having opposite truth-false values in the model (set of support strategy with semantic decomposition).
3. One can use a sorted (to be specified later) predicate calculus with which problems can be more concisely formulated and with which resolutions can be blocked.

In this paper we report about our exploration of the third option.

What has this to do with semantic networks as is mentioned in the title? Although there is no consensus about what are semantic networks and what can be done with them it is reasonable to claim they are excellent for expressing is-a links between entities. The ordinary predicate calculus can accomodate this kind of information - clumsily - by offering one argument predicates with which the type of an argument can be expressed. However the sorts can takeover more efficiently the responsibility to capture this knowledge. At the same time a problem description in natural language needs less modification when sorts are available than when the formal language is the pure predicate calculus. A semantic network would play a crucial role in an automatic translation process by providing the sort-specifications.

In section 2 we will define the extension of predicate calculus with sorts. In section 3 we outline the refinement of resolution, based on Kowalski's connection graphs[5], which was implemented with the sorts in the Lisp program Cogito. In section 4 we report and discuss the experimental results we have obtained thusfar.

2. Sorts

2.1. Definition of sorts

Before defining the concept of sorts we give the intuitions behind it. The idea is simply to attach information to identifiers permitting e.g. to suppress preconditions, resulting in more concise predicate calculus formulas. A simpler example is the habit to introduce easy recognizable distinct sets of mnemonics for entities of different types in the mathematical literature. They facilitate the reasoning process by preventing naive substitutions. An other example is the necessity to declare identifiers in Algol 60 and Algol 68, which allows the compiler to perform type checking.
To say it loosely: semantics is partly transferred into syntax.

We start with a typed predicate calculus. We first provide some definitions.

Definition: Let there be an at most countable set of types T_o, then

$$T = \bigcup_{n=o}^{\infty} T_n \text{ with } T_{n+1} = T_n \cup \{t \mid t = (t_1, \ldots t_{n_t}) \text{ where } n_t \geq 1 \text{ and } t_i \in T_n\}$$

Usually the elements of T_o are interpreted as collections of entities which have mutually empty intersections. We will however in the sequel introduce inclusion relations for the sorts such that non empty intersection will be allowed. The types in T_o 'contain' elements and variables, while the types of $T-T_o$ 'contain' the predicates and the predicate variables.

Definition: ord is a function from T to the natural number IN with $\text{ord}(t) = \min\{i \mid t \in T_i\}$

Every element, element variable, predicate and predicate variable is assumed to have associated with a unique type and according to the definition of ord they have a unique order. The expressive power is enhanced by the introduction of types for functions, function variables, functionals, etc. in the next definition.

Definition: Let $F_o = \{f \mid f = ((t_1, \ldots, t_{d_f}), t_{r_f})$ while $d_f \geq 1$ and $t_i \in T\}$, then $F = \bigcup_{n=o}^{\infty} F_n$ with $F_{n+1} = F_n \cup \{f \mid f = ((f_1, \ldots f_{d_f}), f_{r_f})$ where $d_f \geq 1$ and $f_i \in F_n\}$.

Since an argument of a predicate can be the value of a function and since an argument with order greater than zero is also a predicate, predicates also may be composite.

Whether the predicate of an atomic expression is composite or not, it has a unique type. In a typed predicate calculus there must be a perfect correspondence between the types which are demanded by a predicate and the type(s) of its argument(s).
The same holds for functions and their arguments. With a formula of the predicate calculus we will assume to be given an explicit specification (declaration) of the types of all the identifiers in the formula. As announced we introduce relations which can be used in the declarations. Since in the sequel two different types can have non empty intersections we will call them sorts.

Definition: When s and $\{s_i\}$ are sorts of order 0 then $I(s, s_1, \ldots, s_n))$, with $n \geq 1$ holds when $\bigcup_{i=1}^{n} s_i \subset s$, while $s_i \cap s_j = \emptyset$ when $i \neq j$.

Definition: When s and $\{s_i\}$ are sorts of order 0 then $E(s, (s_1, \ldots, s_n))$ with $n \geq 2$ holds when $\bigcup_{i=1}^{n} s_i = s$, while $s_i \cap s_j = \emptyset$ when $i \neq j$.

We hurry to give an example. The obvious inclusion relations between Humans, Men, Women and Parents can be captured by:
E(Humans, (Men, Women))
I(Humans, (Parents)).

Further we assume to be given for each sort a characteristic predicate which has the obvious property that it evaluates to true if and only if its argument belongs to its domain.

We will relax the constraints as they hold in a typed predicate calculus between the types of arguments and the types of argument positions. For a function it will be allowed that an argument has a sort that is a part of the required sort on its position. For a predicate it will even be allowed that the sort of an argument encompasses the sort of the position it occupies. This allows e.g. writing:

(h) (c){Parentof (h, c)↔ Fatherof (h, c)∨ Motherof (h, c))}, where h is a variable of sort Humans, we have E(Humans, (Men, Women)), and the first argument position of the predicates Fatherof, Motherof is respectively Men and Women.

2.2. Consequence for unification and resolution

The unification algorithm is slightly changed as a consequence of the sorts. We follow the formulation of Chang & Lee[1]. Let W be a set of expressions:

Step 1: k:= 0; W_k:= W; σ_k:=ε.

Step 2: If W_k is a singleton then exit with σ_k, which is the most general unifier for W; otherwise D_k:= disagreement set of W_k.

Step 3: If there exist elements t_k and v_k in D_k such that v_k is a variable that does not occur in t_k and the sort of t_k is included in the sort of v_k, go to step 4. Otherwise stop, W is not unifiable.

Step 4: σ_{k+1}:=σ_k+{t_k/v_k}; W_{k+1}:=W_k{t_k/v_k}.

Step 5: k:=k+1; go to step 2.

A modification is made in step 3 of the algorithm. An additional constraint on the variable is requested to capture the specifications that are associated with the identifiers.

In the same spirit we modify the definition of a binary resolvent; but first we give a definition about significance of a literal.

Definition: A literal L is significant when for every argument holds:
a. its sort is included in the sort of the argument position, or
b. its sort encompasses the sort of the argument position, and the argument does not occur elswhere in an argument position in L such that the sorts of these two arguments positions are disjoint.

Example: (see section 2.1) let v be a variable with sort Humans, let P be a two place predicate of sort (Men, Women). The literal P(v, v) is then insignificant.

Definition: Let C_1 and C_2 be two clauses with no variables in common. Let L_1 and L_2 be two literals in C_1 and C_2 respectively. If L_1 and $\neg L_2$ have a most general unifier σ then C':= $(C_1\sigma - L_1\sigma)\cup(C_2\sigma - L_2\sigma)$. Remove from C' every insignificant positive literal. The resultant clause is a binary resolvent of C_1 and C_2. (In case C' has an insignificant negative literal or a positive characteristic predicate literal, whose argument fits, then C' is a tautology.)

In fig. 1α,β we give an example which is expressed respectively without and with sorts.
The short version is 'infinitely' superior in this example since the number of clauses obtained after translating in conjunctive normal form and the number of resolutions needed without sorts is 14 and 10, while when using the version with sorts one obtains only 6 clauses and one does not have to produce any resolvent. The reason is than the negated theorem is only one insignificant positive literal and thus gives immediately the empty clause !

axioms:	$(h)(Humans(h)\leftrightarrow(Men(h)\vee Women(h)))$, $(h)(Humans(h)\rightarrow(\neg(Men(h)\&Women(h))))$, $(f)(Fathers(f)\rightarrow Men(f))$, $(m)(Mothers(m)\rightarrow Women(m))$, $(f)(Fathers(f)\leftrightarrow(\exists c)(Fatherof(f,c)))$, $(m)Mothers(m)\leftrightarrow(\exists c)(Motherof(m,c)))$, $(h_1)(h_2)(Per(h_1,h_2)\leftrightarrow(\exists c)(Fatherof(h_1,c)\&Motherof(h_2,c)))$.
theorem:	$(h)(\neg Per(h,h))$.

Fig. 1α: Example in predicate calculus without sorts. Translation in conjunctive normal form gives 14 clauses. Derivation of the empty clauses takes a minimum of 10 resolutions.

sorts	: Humans, Men, Women, Fathers, Mothers.
relations	: E(Humans,(Men,Women)), I(Men,(Fathers)), I(Women,(Mothers)).
variables	: $(h,h_1,h_2,c$: Humans$)$,$(f$:Fathers$)$;$(m$:Mothers$)$.
predicates:	(Fatherof:(Father,Humans)), (Motherof:(Mothers,Humans)), (Per:(Fathers,Mothers)).
axioms	: $(f)(\exists c)Fatherof(f,c))$, $(m)(\exists c)(Motherof(m,c))$, $(h_1)(h_2)(Per(h_1,h_2)\leftrightarrow(\exists c)(Fatherof(h_1,c)\ \&\ Motherof(h_2,c)))$.
theorem	: $(h)(\neg\ Per(h,h))$.

Fig. 1β: Same example as in fig. 1α but now using sorts. Translation gives only 6 clauses and since the negated theorem is an insignificant literal no resolutions are necessarily to derive the empty clause. (The axioms are thus superfluous.).

We end the section with a more substantial example. It is the custom to represent the axioms of group theory not with a predicate GROUP, along the lines of:

$$(G)(GROUP(G)\leftrightarrow \ldots),$$

but to consider the group to be the 'universe of discourse' and to represent the axioms thus:

$$(x)(y)(z)\{x*(y*z)=(x*y)*z\},$$
$$(x)\{x*e=x\&e*x=x\}\text{and}$$
$$(x)\{x*x^{-1}=e\&x^{-1}*x=e\},\ \text{see e.g. (Minker, 8).}$$

In a pure predicate calculus a subgroup H can subsequently be introdoced by making H a 1-argument predicate and by stating:

$$(\exists x)\ H(x),$$
$$(x)(y)\{H(x)\&H(y)\rightarrow H(x*y)\} \text{ and}$$
$$(x)\{H(x)\rightarrow H(x^{-1})\}.$$

With the sorts a subgroup H can be introduced alternatively by declaring H to be a sub-universe of G with: E(G, (H, ¬H)) [¬H is the complement universe]; by specifying the element e to belong to H (and thus also to G); and by specifying h1, h2 and h3 to be variables over the sub-universe H and stating:

$$(h1)(h2)(\exists h3)\ h1*h2 = h3 \text{ and}$$
$$(h1)(\exists h3)\ h1^{-1} = h3\ .$$

We feel that the second representation is more natural. Moreover, Nevin's[7] coset theorem formulated in pure predicate calculus yields 30 clauses with 91 literals, while a formulation using the sorts yields only 19 clauses with 25 literals.

3. Implementation of sorts in Cogito

In order to test the usefulness of sorts we implemented them with a refinement of resolution in a Lisp program.
The size of the program (together with a predicate calculus - conjunctive normal form translator) is about 2000 lines and runs under Lisp4 with operating system Scope 3.4 on a CDC. The program often behaves like a file juggler and garbage generator because the amount of available core is small, there is no system-supported virtual memory and the virtual memory that is simulated by Lisp itself causes continuous dragging in of functions. (Moreover: for lack of space clauses are also represented as functions, without arguments, and have to be redefined and written on disk every time mutations in their bookkeeping are made.)

We used Kowalski's connection graphs[5] since they offer the advantage not having to search for unifiable literals. We supplemented them with N_1-resolution, i.e. factoring only of strict positive clauses, resolutions only when one of the parent clauses has only negative literals, and where the literal resolved upon is the 'smallest' (according to some literal ordering) in that clause.

A flag controls whether subsumption tests are performed with newly generated clauses. The same flag controls whether a 'brushup' operation will be tried on clauses. (A brushup of a clause is effective when deletion of one or more literals results in a clause that is still logical equivalent.

Example: $A(x) \vee A(y)$ can be replaced by $A(x)$. The example in fig. 2 shows that the brushup operation can prevent subsumption of blocking the derivation of the empty clause.
The brushup operation is however not sufficient. In a similar example with clauses: (1) A(x,y) B(x,y), (2) A(x,y) B(x,y), (3) B(z,z) Q, (4) Q immediate subsumption blocks derivation of the empty clause, while a brushup does not apply.

Link selection is governed by a heuristic function, which is momentarily problem-family independent. The heuristic function which assigns a numeric value to a link is:

$$wln.f_{wln} + wlp.f_{wlp} + wlvn.f_{wlvn} + wordr.f_{wordr},$$

where the constants and the base function are:

wln = 10, f_{wln} is the length of the negative literal list after merging;

wlp = 8, f_{wlp} is the length of the positive literal list before merging;

wlvn = 2, f_{wlvn} is the number of variables in the negative parent clause;

wordr = 5, f_{wordr} is the order of the clause, with the order of an input cluase is zero, and the order of a resolvent is one plus the sum of the order of the parent clauses.

We do not claim that the values of the constants are optimal They are only a wild guess and we did not tinker at them.

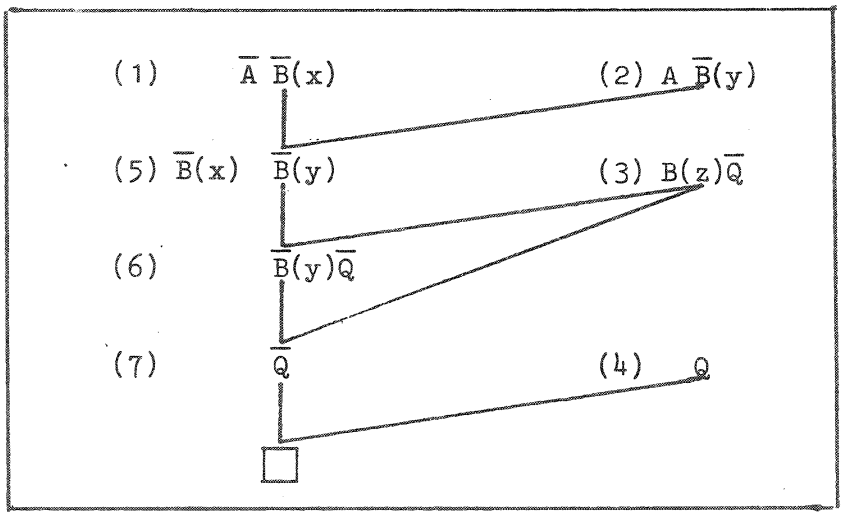


Fig. 2. Example of a set of clauses (1, 2, 3, 4) where the derivation of the empty clause is impossible when subsumption is allowed without a brushup operation; clause (6) is subsumed by clause (5).

4. Experimental Results

4.1. Comparison with other theorem provers

In order to test the program Cogito it was turned loose on some problems that are reported in the literature. Fig. 3 contains the data of three different problems.

No use was made of the sorts in these problems.

The behavior of Cogito is vastly superior in these problems. We ascribe this to the connection graph technique and more specific to the one-step-look-ahead that is the automatic consequence of this technique.

problem	program and/or author	input + generated clauses	clauses purged	clauses left	solution length	clauses in proof	r (1)	g (2)
monkey & banana	Green (3)	28	17 ?	11 ?	11	21	>1 ?	0.750
	Cogito+ subsum.	11+15 26	13	13	13	22	>1	0.846
sort	Green (3)	286	171 ?	115 ?	16	26	0.226	0.091
	Cogito+ subsum.	10+28 38	17	21	12	22	>1	0.579
wos3	P1+Q* Wil/Mink (4)(5)	17	4	13	?	7	0.538	0.412
	SOS+ΣLi Wil/Mink (4)(6)	32	7	25	?	7	0.280	0.219
	USOS Wos (4)(7)	30	5	25	?	8	0.320	0.267
	Cogito+ subsum.	5+5 10	1	9	3	7	0.778	0.700

Fig. 3. This table contains comparisons between Cogito and some other programs.

(1) r= #clauses in proof/# clauses retained.
For Cogito this number does not have much significance since as a consequence of purging clauses that get a pure literal (=having no links) the number of clauses in the proof is sometimes greater than the number of clauses retained.

(2) g= # clauses in proof/# clauses generated.

(3) the data for this row was taken from (3).

(4) " " " " " " " " (8).

(5) P1+Q*=P1 deduction with Q*-algorithm, see also (6).

(6) SOS+Σ*; Li=set of support with Σ*-algorithm, linear resolution, see also (4).

(7) USOS=unit preference with set of support, see also (9).

4.2. Using a characteristic predicate in the sorts

In order to test the effectiveness of employing a characteristic predicate we reformulated Green's sorting problem, that also was used in section 4.1. Fig. 4α contains the predicate calculus formulas to be supplemented with two versions of the declarations; fig. 4γ has clever declarations in which the one place predicate Sd is made the characteristic predicate of the sort Srlist, which is declared to be included in the sort List. (The rationale behind it: a sorted list, an entity in Srlist, is a special kind of list, an entity in List.) Fig. 4δ shows the performance of Cogito on the two versions which confirm the effectiveness of using a characteristic predicate.

[Treatment of the coset example of section 2.2. still awaits the extension of Cogito with paramodulation. A sub-problem recognizer that is already in operation however is able to decompose the pure predicate formulation into three sub-problems each having 14 clauses and with respectively 21, 21 and 18 literals and it is able to decompose the formulation with sorts also into three sub-problems each having 12 clauses with respectively 16, 14 and 14 literals.]

```
axioms:  (x)(y)(R(x,y)↔(Same(x,y)&Sd(y))),
         (x)(y)(Sd(y)→Sd(merge(x,y))),
         (x)(y)(u)((Sd(y)&Same(x,y)→
                    Same(cons(u,x),merge(u,y))),
         (x)(Equal(x,nill)→R(x,nill)),
         (x)(¬(Equal(x,nill))→Equal(x,cons(car(x),cdr(x)))),
         (x)(u)(v)((Equal(x,u)&Same(u,v))→Same (x,v)).
theorem: (x)(∃y)(Equal(x,nill)→R(x,y)) &
                  (((¬(Equal(x,nill)))&R(cdr(x),sort(cdr(x)))→
                   R(x,y))).
```

Fig. 4α Predicate calculus formulas of Green's sorting problem.

```
sorts      : List.
variables  : (x,y,u,v:List).
constants  : (nill: List).
predicates: (Equal,R,Same: (List,List)),(Sd:List)).
functions  : (car, cdr, sort: ((List),List)),
             (cons, merge: ((List,List), List)).
```

Fig. 4β Declarations for the sorting problem without characteristic predicate. Translation of the axioms and the negated theorem into conjunctive normal form gives 10 clauses.

```
sorts      : List, Srlist.
I          : (List,(Srlist)).
variables  : (x,u,v:List),(y:Srlist).
constants  : (nill:Srlist).
predicates: (Equal,Same:List,List)),
             (R:(List,Srlist)),
             (Sd:(Srlist)).
functions  : (car,cdr:((List),List)),
             (sort:((List),Srlist)),
             (merge:((List,Srlist),Srlist)),
             (cons:((List,List),List)).
characteristic
predicate:   (Sd:Srlist).
```

Fig. 4γ Declarations for the sorting problem with a characteristic predicate. Translation of the axioms and the negated theorem into conjunctive normal form with these declarations gives only 8 clauses.

COLUMN	I		II	
PROBLEM	GREEN WITH P.C.		GREEN WITH SORTED P.C.	
Subsumption?	NO	YES	NO	YES
Input + generated clauses	10+60 70	10+28 38	8+24 32	8+17 25
Clauses purged	19	17	8	10
Clauses left	51	21	24	15
Solution length	12	12	9	9
Clauses in proof	22	22	17	17
r	0.431	> 1	0.708	> 1
g	0.314	0.579	0.531	0.680
Time spent secs	67.87	37.55	14.25	11.99

Fig. 4 Since less clauses are generated and the solution length is shorter, usage of a characteristic predicate, whenever possible, is highly advisable.
Five clauses were purged as a consequence of subsumption in column I; while three clauses in column II. The gain which results from performing the expensive subsumption test - also reflected in the processing time - suggests that the N_1-resolution we used leaves still room for improving the prevention of generating useless clauses.

5. Conclusion

The experimental results that we reported in section 4 support our claim that a predicate calculus equiped with sorts leads to savings in a proof process. For lack of time we were unable to test our theorem prover more thoroughly on more examples. Since this paper is in fact a progress report we will try out more examples in the future.

Next extensions to be made to the program are respectively:
- incorporation of paramodulation;
- adding a resolution proof tree analyser which extracts and proposes lemmas to be used in subsequent proof efforts;
- adding the possibility to specify a model to be used by the program for clause simplification and/or deletion.

The speed of the program Cogito might certainly drastically increase by re-implementing it in a low level language and by then being bit-conscious, but we refrain from doing it since we do not consider the program Cogito, and theorem provers in general, mature enough to leave the experimenting stage.

Acknowledgement

This research originated from a grant of the Netherland Organisation for the Advancement of Pure Research (Z.W.O.), which enabled the author to visit J.A. Robinson at the University of Syracuse, N.Y.
The factor generating component of Cogito is due to H.J. Sint.
Discussions with her on an early draft were appreciated.
Careful typing was done by Pom van der Horst.

6. References

1. Chang, C. & Lee, R.C., Symbolic Logic and Mechanical Theorem Proving, Academic Press, N.Y.

2. Gelernter, H. et al., Empirical Explorations of the Geometry Theorem Proving Machine, reprinted in: Computers and Thought, Ed Feigenbaum, Feldman, McGraw-Hill, 1963, pp. 153-163.

3. Green, C., The Application of Theorem Proving to Question-answering Systems, AI Group, Technical Note 8, SRI, Stanford, 1969.

4. Kowalski, R., Search Strategies for Theorem Proving, in: Machine Intelligence, Vol. 5, Ed Meltzer & Michie, Elsevier, N.Y., 1970, pp. 181-212.

5. ———, A Proof Procedure Using Connection Graphs, in: Journal of the ACM, Vol. 22, October 1975, pp. 572-595.

6. Minker, J., et al., MRPPA an Interactive Refutation Proof Procedure System for Question Answering, J. Computers and Information Sciences, Vol. 3, June 1974, pp. 105-122.

7. Nevins, A.J., A Human Oriented Logic for Automatic Theorem-Proving, JACM 21, October 1974, pp. 606-621.

8. Wilson, G.A. & Minkler, J., Resolution, Refinements and Search Strategies; A Comparative Study, in: IEEE Transactions on Computers, Vol. C-25, No. 8, August 1976, pp. 782-801.

9. Wos, L.T., Argonne Nat. Lab., unpublished notes, 1965.

DETERMINING THE ORIENTATION OF SURFACES FROM OPTICAL FLOW

William F. Clocksin
Department of Artificial Intelligence
University of Edinburgh
Edinburgh, Scotland

Abstract

This paper presents a method for computing the orientation of physical surfaces relative to the moving observer. The method is theoretically derived; it is not 'ad hoc'. The method exploits the remarkable properties of the optical flow, a concept which figures prominently in the Gibsonian theory of visual perception. We first describe the dynamic nature of visual information, and formally derive the optically sensed velocity field transformation (optical flow) for a moving observer. Then we derive the inverse transformation to show how gradients of optical flow uniquely inform about the orientation and shape of physical surfaces.

Key Phrases: motion perception, motion contrast, optically sensed velocity fields, optical flow, physical constraints, surface orientation.

Introduction

We propose a computational theory of how the human visual system may obtain useful information about the physical environment directly from optically sensed velocity fields. We present here one result of the theory: a method for computing the orientation of physical surfaces relative to a moving observer.

It is important to first understand how perceived velocity fields arise from events in the environment. This paper begins by describing the character of optically sensed velocity fields. Next, we examine the two kinds of proximal velocity gradients, or motion contrasts, and how distal events determine them. Finally, we give details of the theory for the first type of motion contrasts, smooth motion contrasts, and how this type of information can be exploited by a visual system to determine the orientation of surfaces in the environment. We are currently implementing a computer program that uses the theory to determine the three dimensional layout of arbitrary objects with respect to a moving observer. The psychological and physiological implications of the theory are also currently under study.

Visual Information is Not Static

What visual information is available to an observer? We contend that a continuous transformation of stimuli over time is the natural

input to our visual systems. This view complements traditional artificial intelligence (AI) approaches to vision research. AI research has dealt with static scenes, such as pictures, drawings, or single frames of a television signal. This approach has occasionally fostered the belief that the human visual system, in turn, operates somewhat as a camera does. Indeed, one popular belief among AI researchers is that information about the three dimensional world must somehow be extracted solely from a flat static image (Bowden,1977:180). But, since humans are mobile creatures who consist of moving parts, it is the transformation of stimuli over time that is the usual stimulus. Non-transformations -- static images -- are rare and exceptional in nature. Not only in vision, but in other sensory modes such as touch and hearing, the natural information available is a continuously changing array of stimulation, and not a static one.

Is this approach relevant to computer vision? We argue that the flow of stimulation, the continuous transformation of stimuli over a sensitive surface, reveals information about the visual environment more naturally and compellingly than does the static image. Operations performed on one type of flow, the optical flow, are more suitable for revealing the activity of visual perception. A computational theory of the role of optical flow in visual perception would be useful not only for accounting for human visual perception in certain cases, but also as a fundament and itinerary for new methods in computer vision. By presenting a theory of how surface orientation can be recovered from optical flow information, we hope to initiate this line of research.

Optical Flow

The optically sensed velocity field we are most interested in is the optical flow. It is a kind of visual stimulation typically generated by movements of textured surfaces in the world or by a moving observer of textured surfaces. Gibson (1966) argues that this continuously changing flow of optical stimulation provides information about the layout of the environment and about the movements of the observer. Notwithstanding our usual conception that our visual field contains a distribution of brightnesses, the visual field also can be thought of as a distribution of velocities. This velocity field arises from events in the environment. In general, there will be a velocity vector associated with each point in the visual field where there is a brightness discontinuity. Thus, velocity vectors are associated with texture elements. The unfortunate observer of a world devoid of brightness dicontinuities would obtain no velocity field.

It is possible to formally determine the velocity distribution that is visually available to an observer. Knowing the velocity of the observer and the velocity of texture elements in the environment, we can calculate the perceived velocity vector for any texture element in the visual field. Although we could derive the flow field for the general case of a moving observer of an environment of moving objects, this full derivation would be too tedious for the purposes of this paper. We choose to derive the special case, where an observer is translating through a fixed environment. This simplified case is sufficient to convey the flavour of how the optical flow arises from

environmental events.

For most human activities that involve body movements, the pattern of optical flow will change many times over a duration of time. However, at any instant there will be one and only one optical flow pattern. This pattern is represented by the equations to be presented below. These equations only express the flow pattern in mathematical terms. We stress that the viewer does not compute these equations. When there are no moving objects other than the observer, the optical flow pattern will depend on the velocity of the observer, and the perceived portion of the optical flow will depend on the observer's direction of sight. In a later section we discuss how the observer recovers the orientation of surfaces from a perceived portion of the optical flow. For the moment we are concerned with the inverse problem: representing the optical flow in terms of the observer's velocity.

The optical flow pattern depends only on the observer's velocity, and not on the observer's direction of sight. The pattern is independent of the state of the observer's visual system. However, the observer is free to scan the pattern by altering the direction of sight to make accessible different portions of the pattern. The observer's velocity, on which the pattern depends, is the sum of all body movements including eye and head movements. Hence, the body movements necessary for scanning the visual environment will also affect the optical flow pattern. However, recall that at any instant there will be only one optical flow pattern, so this pattern will reflect the motions that were made to scan the environment.

The optical environment will be represented in terms of a spherical coordinate system centred on the observer's entrance pupil. The orientation of the coordinate system depends on the direction component of the observer's velocity. As the direction of movement instantaneously changes, the coordinate system instantaneously rotates relative to the outside world.

Refer to Figure 1. We adopt a spherical coordinate system centred at the observer's entrance pupil centre. This is used as a point of central optical projection onto a reference unit sphere. Any point in space P may be represented by spherical coordinates of declination ϕ, elevation θ, and distance from the origin r. Later we find it convenient to compute derivatives with respect to angular coordinates of meridian α and eccentricity β. The eccentricity along lines of meridian is available from the relation $\cos\beta = \cos\theta \cos\phi$. Refer to Figure 2. The unit vector $\hat{v}$ is the direction of the observer's movement; $\hat{p}$ is a unit vector in the direction of a line from the origin to point P; $\hat{n}$ is a unit vector normal to $\hat{v}$ and $\hat{p}$; $\hat{f}$ is a unit vector in the direction of the instantaneous angular velocity $\vec{\omega}$ (the flow) of P.

In this representation, the origin is always centred on the observer; the coordinate system moves. The X-axis always reflects the direction of movement.

Given the observer moving along $\hat{v}$ with speed S, the observer's optical flow is defined by the instantaneous angular velocity that each point P in space makes on the unit sphere. For some point P, the magnitude component of its angular velocity as seen by the observer centred at the origin is

$$\frac{d\beta}{dt} = |\vec{\omega}| = S \cdot \sin\beta / r \ .$$

The direction is along lines of meridian of the sphere. This direction is given by a unit vector normal to $\hat{p}$ and $\hat{n}$, which is $\hat{f}$. The value of $\hat{f}$ is $(\hat{n}\times\hat{p})$. Since $\hat{n} = (\hat{v}\times\hat{p})/\sin\beta$, then $\hat{f} = (\hat{v}\times\hat{p})\times\hat{p}/\sin\beta$.

Gradients of velocity will be computed with respect to lines of eccentricity and meridian. These are equivalent to longitude and latitude respectively, where the poles are defined as the two intersections of the X-axis with the reference sphere.

Physical Constraints

In this section we explore the basis of motion contrast by examining the physical constraints that give rise to it. We assume that the problem of vision begins with a field of velocity vectors such as the optical flow field. Two types of velocity fields can appear in such a field: gradual changes in velocity and stepwise changes. There is an important functional difference between gradual changes and stepwise ones. This difference is brought about by physical constraints that give rise to the stimulus. We contend that stepwise changes inform about discontinuities in depth, whilst gradual (smooth) changes inform about the orientation and shape of a surface in depth.

Refer to Figure 3. The discontinuities in depth that result from stepwise velocity gradients are caused both by the occlusion of one surface to another (Fig. 3a), and by different objective velocities of the surfaces (Fig. 3b). Within the boundary of a particular surface however, only gradual changes occur due to the shape or orientation of the surface (Fig. 3c). It is not difficult to imagine an object that gives rise to stepwise velocity gradients due to shape (Fig. 3d), but we would consider that object as constructed from individual surfaces. In this way, none of the constraints is violated, and the mapping from stimulus to percept is more manageable.

Marr and Poggio (1976) also recognised these constraints for the computation of stereo disparity. They expressed the constraint in the form of two rules, continuity and uniqueness. We can state analogous forms of these rules for our velocity field stimulus. The analogous rule of continuity states that surfaces vary smoothly -- only a fraction of a velocity distribution is composed of borders that correspond to discontinuities in depth. The analogous rule of uniqueness, states that a given point on a physical surface has a unique orientation in space.

Now with the knowledge that the distribution of velocities within a flow field are determined by physical constraints, we can set two tasks. The first task is to specify a computational model. The model would operate on a velocity distribution to produce a description of surface orientation and depth discontinuities. The determination of local surface orientation is the key to shape description (eg. planar vs. corrugated). Discontinuities of depth delimit the extent of surfaces. The model for smooth surfaces is given in the next section.

Second, we can experimentally find out what percepts are obtained by viewers of certain artificially produced displays of motion contrast. We would expect to verify the argument above -- that

stepwise changes yield percepts of depth discontinuities along the border, and that gradual changes yield percepts concerning orientation and shape. More importantly, we would hope to find relationships between objective motion gradients and perceived motion gradients. Are stepwise gradients subjectively enhanced? Is there a threshold below which smooth motion gradients are not perceived? How good is the analogy between motion contrast and brightness contrast? Answers to these questions would supply important factors for determining the computational problems that the visual system encounters during perception.

Smooth Gradients and Surface Orientation

Recall that we distinguish smooth gradients and stepwise ones, and that smooth gradients inform about local surface orientation. We now set aside stepwise gradients in order to construct a mapping from smooth velocity gradients onto local surface orientation.

We can use the spherical coordinate formalism described earlier. For the sake of simplicity, we will deal with an infinitesimal section of the sphere in Figure 1. The section is cut through the X-Y plane, at $\theta=0$. This two dimensional simplification is shown in Figure 4. A top view, looking down the Z-axis, gives Figure 5. The general case is derived shortly.

With our observer moving in the direction of $\hat{V}$ at speed S, the problem is to find a mapping from the given velocity field to a description of each point on the surface Σ. We will represent the velocity field as a function f of ϕ. As for Figure 1, $f(\phi) = S.\sin\phi/r$. The function f is 'given' to the observer as a consequence of her movement through the environment.

We will represent the orientation of a point on a surface as the normal of the surface at the point. The information useful to the observer is the orientation (normal) of the surface with respect to ϕ. ϕ, being the eccentricity with respect to the direction of travel, could just as well be the angle of gaze. Directing the gaze along ϕ would simply foveate P, although the calculations hold independently of the observer's regard. Now the angle σ between r and the normal of the surface at P is

$$\tan\sigma = \frac{1}{r}\frac{dr}{d\phi} \quad .$$

Angle σ specifies the orientation in an observer-oriented way. For a part of the surface facing the observer, σ is nil. As parts of the surface face away from the observer, σ approaches 90. At a (theoretical) depth discontinuity, σ is (theoretically) 90. Figure 7 shows several values of ϕ and σ for a surface.

This relationship is appropriate when r is known. In fact r is not given directly, but Nature gives us f, with which we can solve for r. For the simple 2D case, $r = S.\sin\phi / f(\phi)$. To find σ we need

$$\frac{dr}{d\phi} = \frac{d}{d\phi} S\cdot\sin\phi/f \quad .$$

The egocentric local orientation σ of a surface is then

$$\tan\sigma = \frac{1}{r}\frac{dr}{d\phi} = \cot\phi - \frac{1}{f}\frac{df}{d\phi} = \cot\phi - \frac{d}{d\phi}\log f \quad .$$

Thus the egocentric local orientation depends on the gradient of the velocity field, $df/d\phi$, as we suspected. σ is undefined at $\phi=0$. We should expect this, since $\phi=0$ defines what Gibson (1960:120) calls the 'focus of expansion'.

This result can be generalised to handle the real case, shown in Figure 1. The velocity field f is now a function of α and β. It is most convenient then to represent an egocentric local orientation of a surface as a pair σ,τ. Let the W-axis be the line through the origin and P. Then σ represents an orientation in the W-X plane, denoting a change in depth along the eccentricity. Similarly, τ represents a tilt in the W-Y plane, denoting a change in depth along a meridian. Figure 8 shows three simple orientations and how they are denoted. Together, σ and τ specify a convenient egocentric orientation in space.

$$\tan\sigma = \frac{1}{r}\frac{\partial r}{\partial \beta}, \quad \tan\tau = \frac{1}{r}\frac{\partial r}{\partial \alpha}; \quad \text{where} \quad r = S\cdot\sin\beta/f \qquad (1)$$

The gradient space formalism (Mackworth,1974) may be appropriate for an implementation of Eq.1. Instead of mapping gradients of velocity onto an egocentric orientation which depends on the location of the texture element with respect to the direction of locomotion, we would map all points of the same spatial orientation into the same point in gradient space G. This is done by mapping a point P at (α,β) into $G(\alpha-\tau,\beta-\sigma)$. The gradient space formalism is convenient because mapped orientations are invariant with repect to time and observer movement. Woodham (1977) also uses the gradient space to represent local orientation when given gradients of brightness from photographs.

Summary

We have presented a method for computing the local orientation of surfaces from the velocity information given by the naturally occurring optical flow. It is important to note that this method is neither a heuristic nor 'ad hoc'. Fixed textured surfaces in the environment give rise to (map onto) a unique distribution of velocities when the observer is translating through the environment. The method given here is nothing more than the inverse mapping.

We are currently implementing the method for the case of a simulated observer moving through a simulated environment; technical and bandwidth limitations render television methods unfit for detecting the optical flow.

We plan to investigate the psychological implications of the theory using human subjects. Physiological implications are also under study. This latter line of inquiry is similar in nature to David Marr's (1974) suggestion that retinal physiology is appropriate for computing the Retinex function (Land and McCann,1971).

Acknowledgements

I thank J.A.M. Howe, David Lee, David Marr, and Richard Young for conversations.

References

Bowden M,1977 Artificial Intelligence and Natural Man, Harvester P.

Gibson J,1960 Perception of the Visual World, Riverside P.

---------,1966 The Senses Considered as Perceptual Systems, Cornell P.

Land E & McCann J,1971 "Lightness and retinex theory", JOSA 61, 1-11

Mackworth A,1974 PhD Thesis, Sussex U.

Marr D,1974 "Essay on the primate retina", MIT AIM-296, and "The computation of lightness by the primate retina", Vision Research 14, 1377-1388

------- & Poggio T,1976 "Cooperative computation of stereo disparity", Science 194, 237-238

Woodham R,1977 "Cooperative algorithm for determining surface orientation from a single view", Proc. IJCAI 5

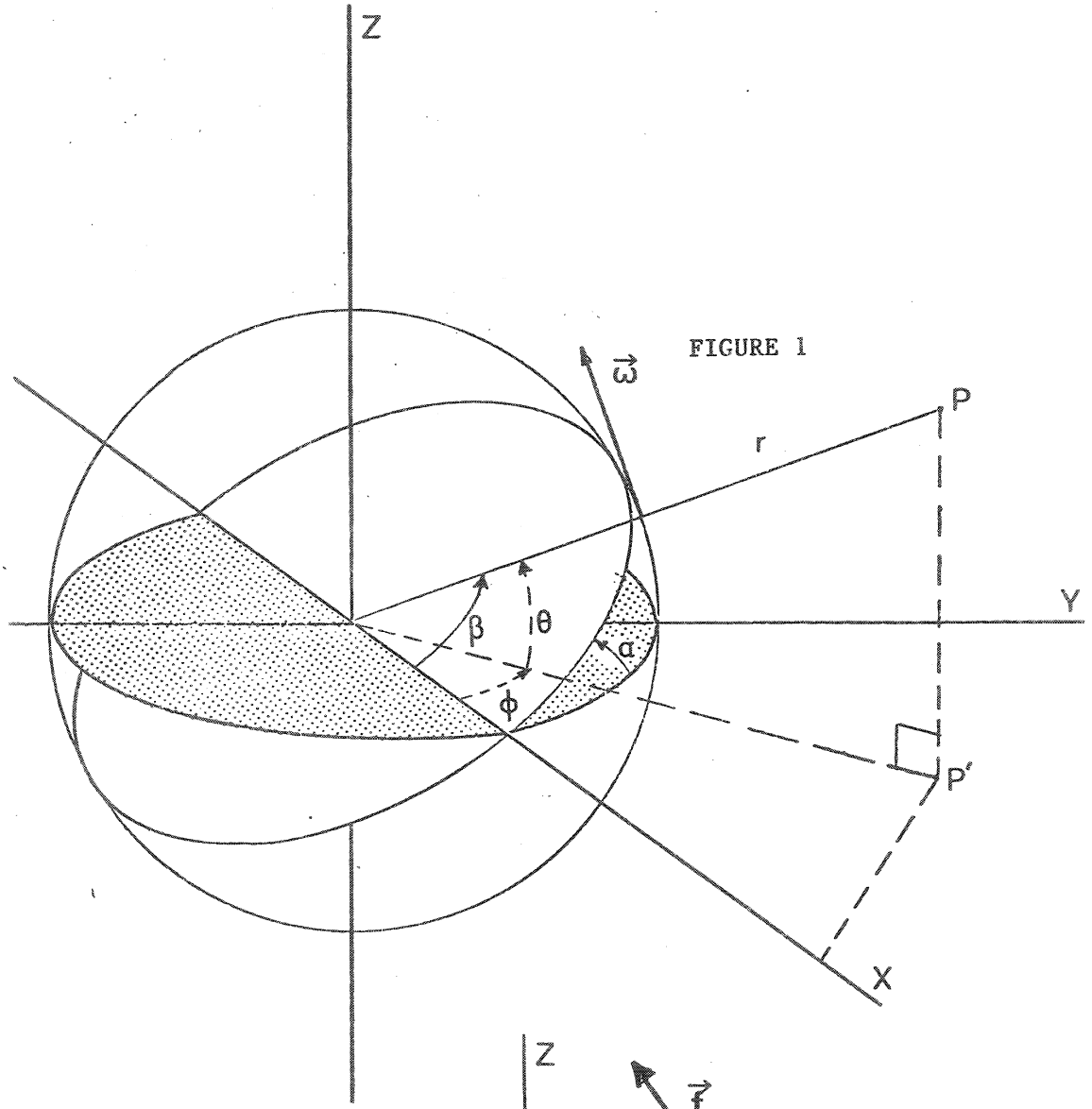


FIGURE 1

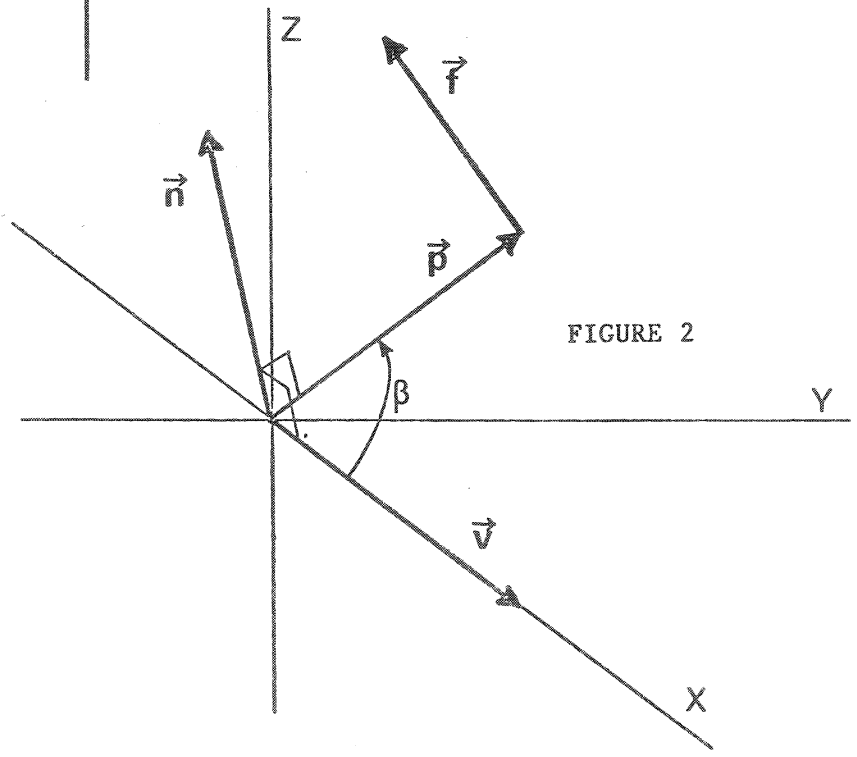


FIGURE 2

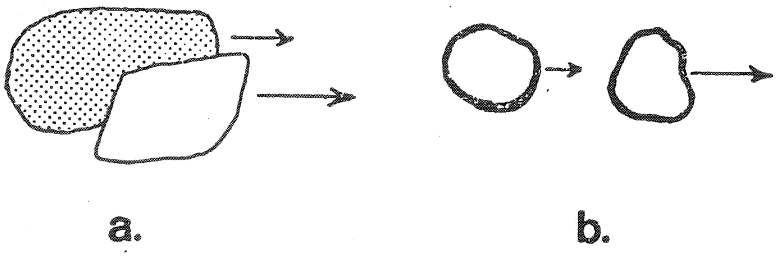
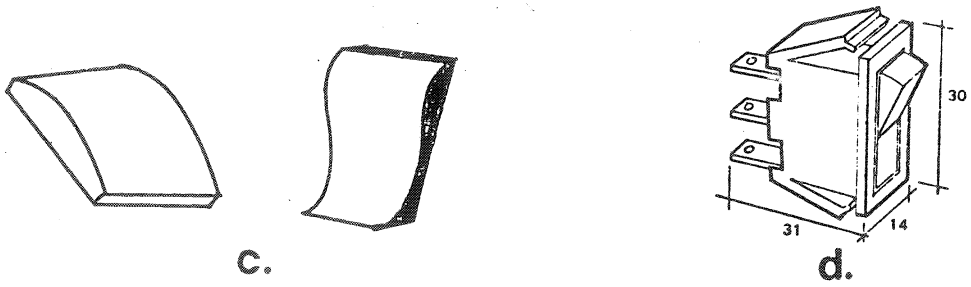


FIGURE 3



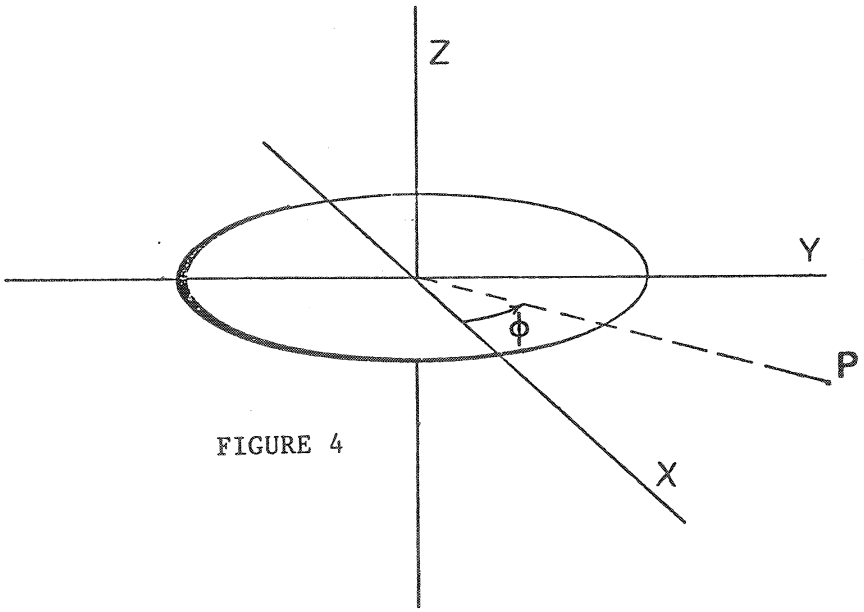


FIGURE 4

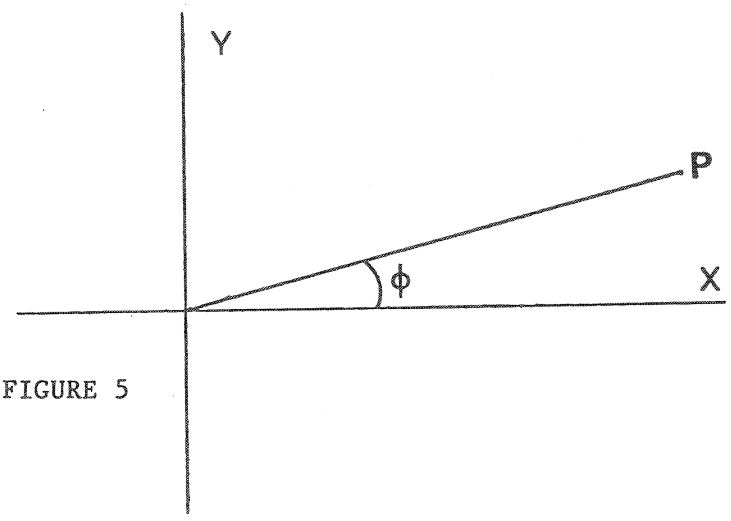


FIGURE 5

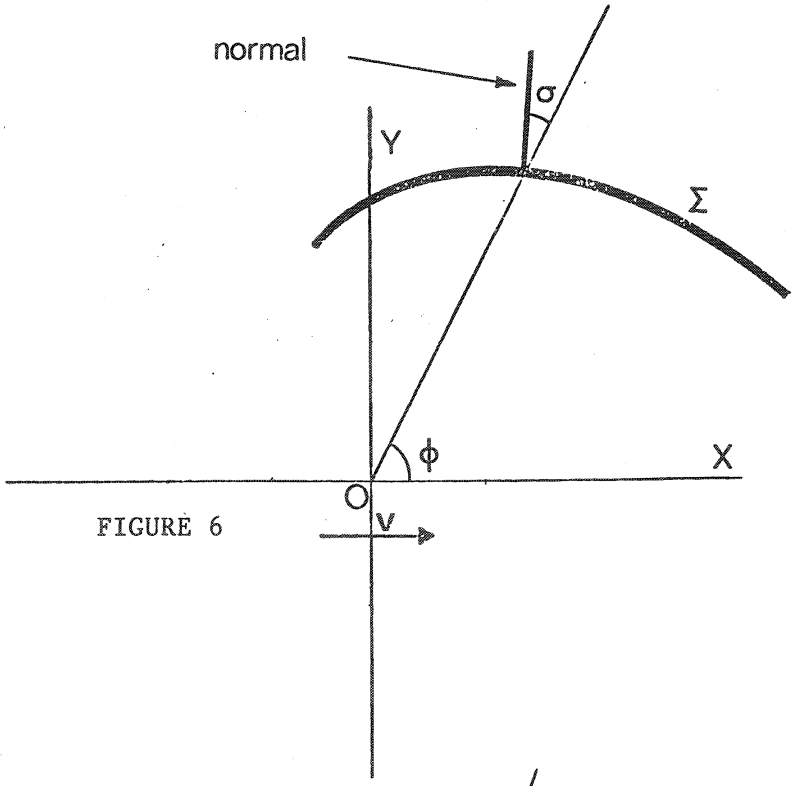


FIGURE 6

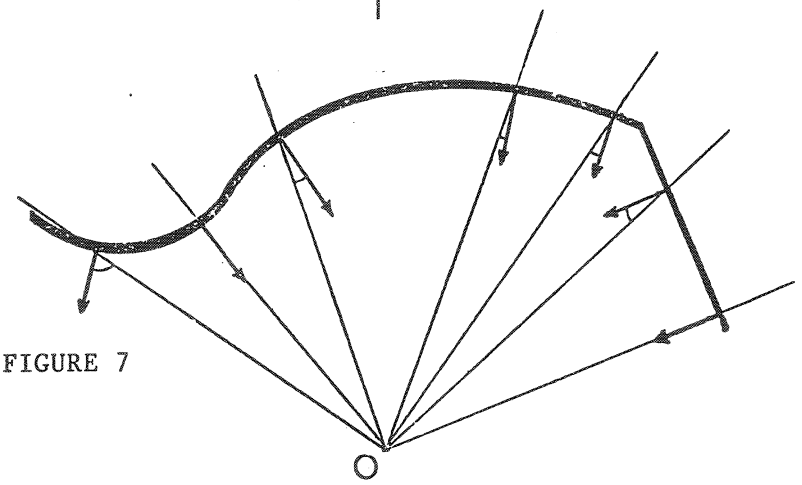


FIGURE 7

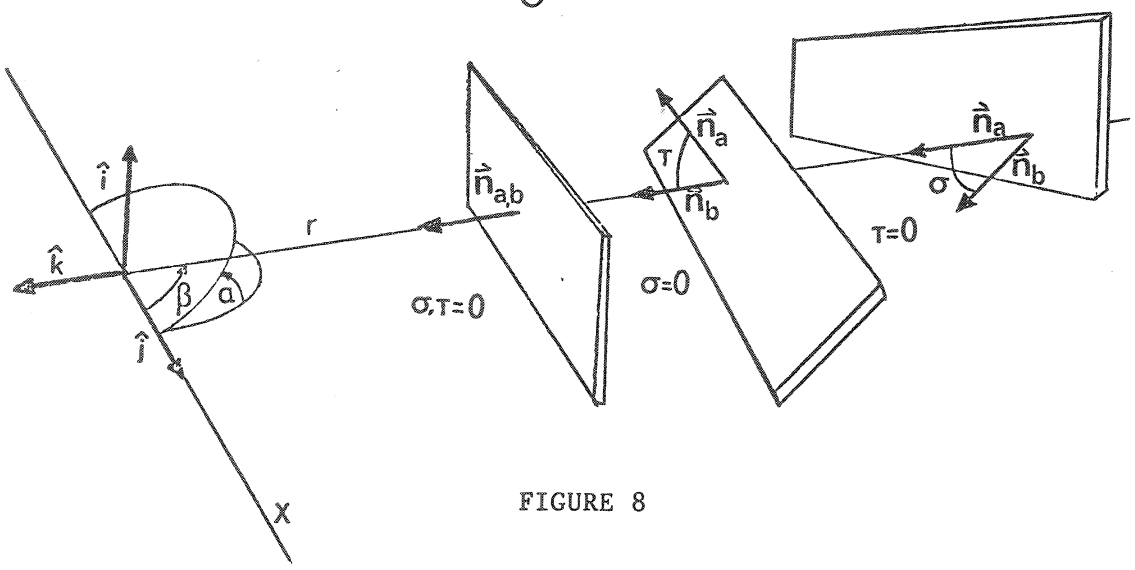


FIGURE 8

PROBLEM SOLVING BY HYPERPLANNING

Ch. EICK and P. RAULEFS

Institut für Informatik III
Universität Bonn
D-5300 Bonn 1, West Germany

Institut für Informatik I
Universität Karlsruhe
D-7500 Karlsruhe 1, West Germany

Abstract

Almost all current problem-solvers are unable to improve their behavior by analysing failures and alter their planning strategies. Only HACKER /1/ provides just a few error-analysis and error-correction mechanisms, but there is no systematic way of incorporating domain-specific knowledge.

A *hyperplan* consists of a sequence of operators and meta-operators both linked to various kinds of nodes for book-keeping. Operators perform state-transformations specified by *conditional add/delete sets*. *Meta-operators* are capable of transforming the hyperplan they are part of. Specific conflict situations *spontaneously* activate *conflict administrators* that use domain-specific criteria to select appropriate *conflict operators* for resolving conflicts. Both conflict and meta-operators can analyse a hyperplan and make appropriate corrections to avoid subsequent conflicts in similar situations.

Hyperplanning provides a uniform approach to problem-solving that
1. incorporates all earlier methods, and
2. allows the analysis of failures, the update and correction of decision criteria and planning strategies dynamically throughout the course of the formation/execution of a hyperplan.

Other innovations of the hyperplan approach are:
a. The *protection mechanism* considerably extends and improves e.g. that of HACKER and CSA /2/. Our protection technique (1) results in detecting conflicts much earlier, and (2) allows to dynamically change protection properties.
b. *Conflict resolution techniques* such as revising, renewing, unravelling, and postponing solve all conflicts resolved by conventional techniques, but do also work for many other conflicts where the other systems fail.
c. Hyperplan execution involves exetensive protocolling that allows to reconsider alternative strategies upon encountering conflicts.

The control structures set up by a hyperplanning problem-solver can rather directly be implemented in terms of *facetting CSSA-actors* /3,4/. The paper describes the problem-solver HYPOS. HYPOS provides a multitude of conflict resolution techniques, as well as the machinery for incorporating domain-specific knowledge. We illustrate HYPOS by carefully working through two detailed examples.

/1/ Sussman, G.J. A Computational Model of Skill Acquisition. Ph.D.-Thesis, M.I.T.-A.I. TR 297 (1973)

/2/ Rieger, C., P. London. Proc. 5th IJCAI-77 (Cambridge 1977)487-493.

/3/ Böhm, H.P., H.L. Fischer, P. Raulefs. Proc. ACM SIGART-SIGPLAN Symp. on AI & PL (Rochester 1977)100-108.

/4/ Böhm, H.P., H.L. Fischer, P. Raulefs. Dialogues in ACTOR nets. Tech. Rept. to be published (see also these Proceedings for an abstract). Inst. f. Informatik I/III, Bonn/Karlsruhe Universities (SEKI-Project), 1978.

PARAMODULATED CONNECTION GRAPHS

Norbert Eisinger, Jörg Siekmann, Graham Wrightson
Institut für Informatik I
Universität Karlsruhe
D-7500 Karlsruhe 1
West Germany

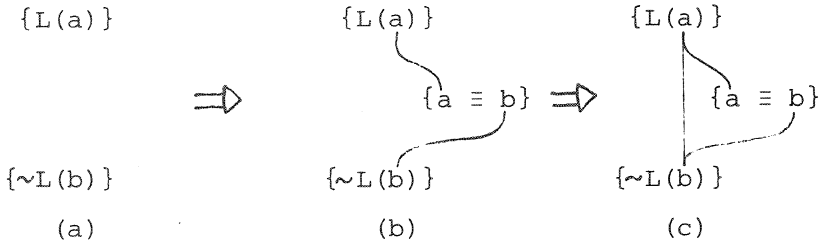
1. MOTIVATION

The axioms and the negated theorem (the input clauses) given to an automatic theorem prover are usually considered as a *set* of clauses. Only recently, [1], [3], [4] and [8] has it been suggested that it may be of considerable advantage to impose some graph structure upon the input clauses and to define particular proof procedures operating upon such graphs.

In this paper we extend the connection graph proof procedure of [1] to include paramodulation, a special rule of inference to handle equality, [2] and [7]. In particular we shall suggest the introduction of additional links connecting those literals which can be made complementary under paramodulation.

To gain a notion of the problem involved assume that two one-literal ground clauses {L(a)} and {~L(b)} form a connection graph (Example 1(a)).

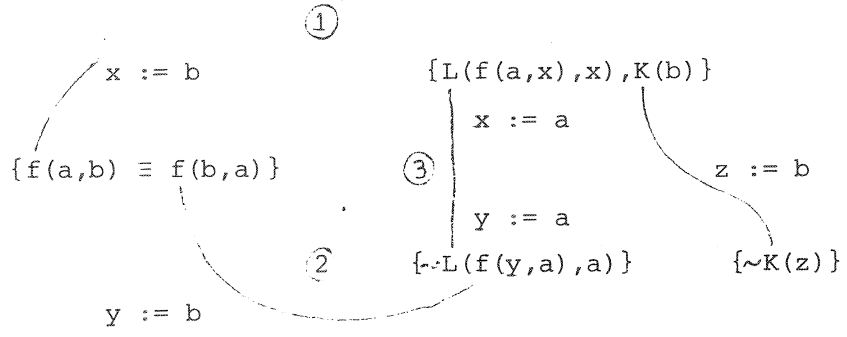
Example 1



As they stand these two clauses are not resolvable with each other. Now if it were known that a is equal to b and some way could be found of entering this information into the connection graph so as to be able to make a substitution into one of the literals then the two literals would become resolvable. The initial idea is to introduce this information in the form of an equality unit{a≡b}which is then connected to the

clauses by two special links indicating possible paramodulations, either of b into {L(a)} or a into {~L(b)} (Example 1(b)). (These links will be called paramodulation links or P-links in the sequel, as opposed to the normal links connecting unifiable literals, hereafter referred to as R-links).
Now, to complete the idea, a link is included to indicate the potential resolvability of {L(a)} and {~L(b)} (Example 1(c)). But just because these clauses are not unifiable as they stand the link cannot be an ordinary R-link. Rather the link indicates that the two literals connected can be made complementary under paramodulation: that is after certain paramodulation steps these two literals may be resolved upon. The information represented by such a link is absolutely essential, if the basic principle of the connection graph proof procedure is to be extended to the case of paramodulation. Unfortunately it is in general undecidable, whether such a link is to be set or not and hence the extension of the connection graph proof procedure by paramodulation appears to be impossible.
This fundamental problem will be discussed later and in the meantime two examples are given to demonstrate some of the advantages which would result if the connection graph proof procedure could be extended by paramodulation.
Example 2 illustrates the incompatability deletion for paramodulation links. The graph consists of the four clauses {L(f(a,x),x),K(b)}, {~K(z)}, {~L(f(y,a),a)} and {f(a,b) ≡ f(b,a)}. All possible P- and R-links are set and each link is labeled with it's corresponding substitution, e.g. x := b in P-link ①.

Example 2



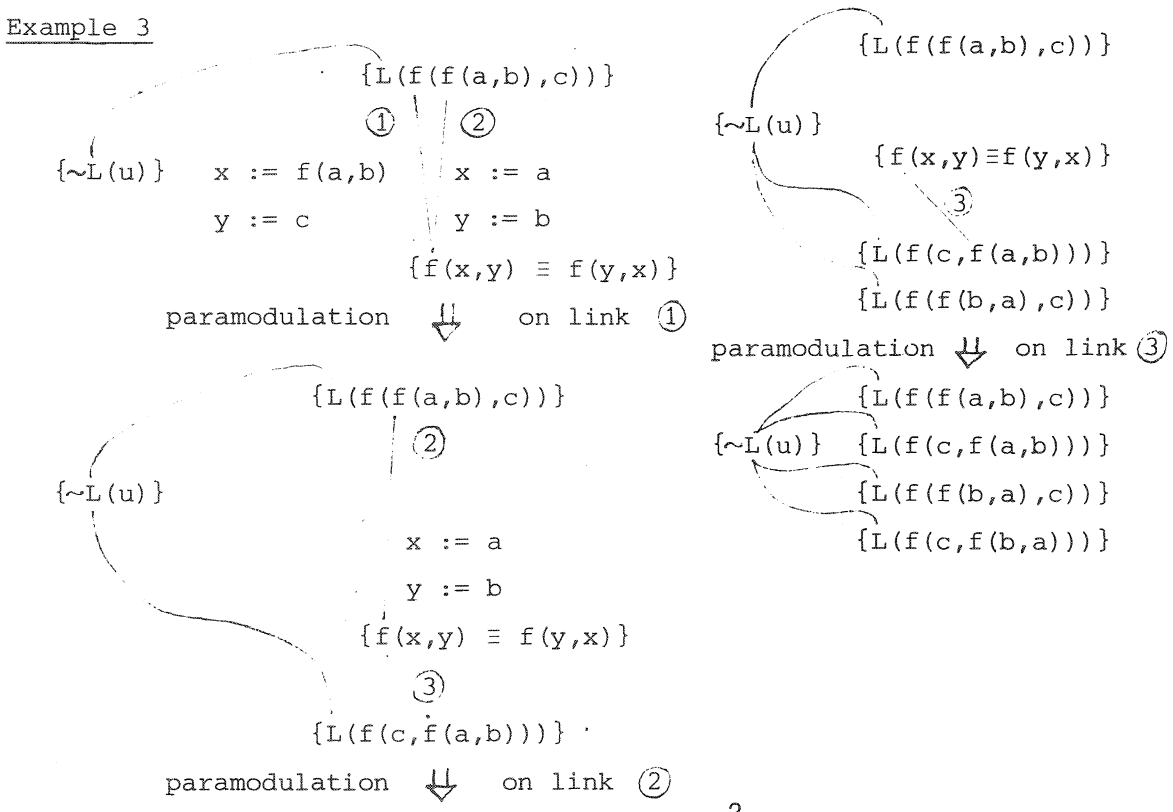
Paramodulation links ① and ② are incompatible with link ③ and may hence be deleted, since the respective paramodulants would contain pure literals. Note that in the case of incompatibility between

a paramodulation link and a resolution link always the paramodulation link is erased.

The point of demonstration is, that the equality unit becomes pure after the incompatibility check and hence the whole (equality) clause is erased. This may lead to a snowball-effect of other clauses being erased as a consequence. This familiar effect from the original proof procedure is demonstrated here to hold also for R-links and P-links. This amplifies the introductory remark that the Waltz effect is the more probable the more links are in the graph. Hence apart from the obvious motivation to provide for an equality rule in connection graphs, there is the additional chance that the graph enriched by P-links may reduce much more rapidly.

Example 3 shows that the exponential explosion of ordinary paramodulation can be avoided and that the control of this potential explosion is immediate in certain cases. Consider the (notorious) case of commutativity:

Example 3

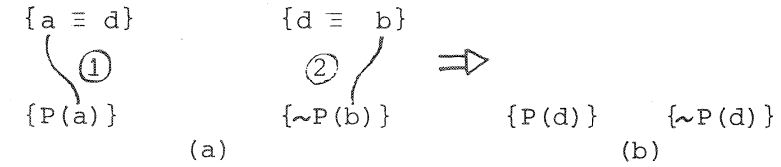


This example demonstrates how all possible ($2^2 = 4$) permutations in f(f(a,b),c) are generated and no additional paramodulants are derived as would be the case with ordinary paramodulation. Note that the order in which the links are selected is irrelevant.

2. THE PROBLEM

In order to achieve the effect mentioned in the previous paragraph, we would like to have a connection graph as dense as possible. In particular we would like to have the following links:
(i) all complementary literals in different clauses which are unifiable are connected with a link, called R-links;
(ii) all equations permitting a paramodulation into a particular literal are connected with a link to that literal, called P-links;
(iii) all literals which can be made complementary under paramodulation and unification are connected by a link.
The problem concerns the links of (iii): it is neither practically feasable nor (in general) theoretically possible to ever set all links of (iii). Consider a group G whose wordproblem is unsolvable. Let S be a set of clauses containing all the equations defining group G. Let $P(w_1)$ and $\sim P(w_2)$ also be in S, where w_1 and w_2 are words in G and it is now impossible to decide whether $P(w_1)$ and $\sim P(w_2)$ should be connected. But suppose we do not initially set all the links in (iii), then the following situation arises: Let S = {{a ≡ d},{b ≡ d},{P(a),~P(b)}} and the initial connection graph is given in example 4(a):

Example 4

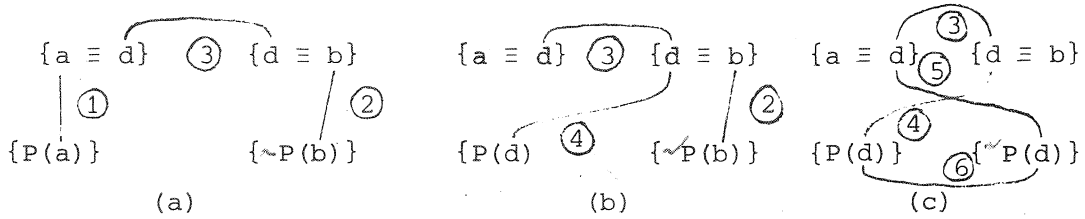


Now, after paramodulating on links (1) and (2), we obtain the connection graph <{P(d)},{~P(d)}> in example 4(b), but since P(a) and ~P(b) were not connected by a link, the paramodulants cannot inherit any links, i.e. P(d) and ~P(d) are *not connected* by a link and therefore can not be resolved upon. The obvious solution then may be, after each paramodulation step, to search through the whole graph and compare the paramodulant with every other literal for unifiability.
But to start searching for unifiable literals after each paramodulation step would destroy the whole purpose of the connection graph proof procedure, which is precisely to eliminate the unsuccessful search.

3. THE SOLUTION TO THE PROBLEM

In order to solve the problem of section 2 we demonstrate that P-links can have more than just the one function of recording possible paramo-

dulations. Their additional function is to store information.
Consider example 4(a) again in which we add another P-link connecting the d's of the two equality units i.e. the graph now becomes:



where (b) is obtained from (a) by paramodulation on link ① and (c) is obtained from (b) by paramodulation on link ②.
If links are inherited which eventually connect non-equality literals (link ⑥ above) a special process is initialized which checks whether the connected literals are resolvable and if so then the P-links is re-coloured into an R-link. Otherwise it is erased.
That is, potentially unifiable literals are discovered (without exhaustive search) and the appropriate R-links are set when needed.

In [6] a proof procedure incorporating these ideas is presented and soundness and completeness results are obtained. Also some additional refinements are discussed.

Literature:

1. Kowalski, R.: A Proof Procedure Using Connection Graphs. JACM 22 (1975).
2. Robinson, G.A. and Wos, L.: Paramodulation and Theorem Proving in First Order Theories with Equality. In: Machine Intelligence 4, (B.Meltzer + D. Michie,eds.), Amer.Elsevier, New York, 1969.
3. Shostak,R.E.:Refutation Graphs.Journ.of Art. Intelligence 7(1976).
4. Sickel,S.:Clause Interconnectivity Graphs,IEEE Trans.on Computing, C-25(1976).
5. Siekmann,J.+Stephan,W.:Completeness and Soundness of the Connection Graph Proof Procedure.Int.Ber.Nr.7/76,Universität Karlsruhe, 1976.
6. Siekmann,J.+Wrightson,G.: Paramodulated Connection Graphs.Int.Ber. Nr.5/77, Institut f. Inf. I,Universität Karlsruhe, 1977.
7. Wos,L.+Robinson,G.:Maximal Models and Refutation Completeness:Semi-decision Procedures in Autom.Theorem Proving.In:Wordproblems(W.W. Boone,F.B.Cannonito,R.C.Lyndon,eds.),North-Holland,1973.
8. Yates,R.A.,Raphael,B.,Hart,T.P.:Resolution Graphs.J.of Art.Int.1(1970).

A SYSTEM FOR VISUAL-LIKE UNDERSTANDING OF LISP PROGRAMS

Daniel GOOSSENS
Département d'Informatique
Université PARIS 8
Route de la Tourelle
75012 PARIS

ABSTRACT

Presented is an implemented system: CAN, which understands LISP programs. The system is a program debugging tool and is intended to use procedural knowledge about concrete list structures. The set of rules which implement this knowledge is represented as a unification process, called META-PATTERN-MATCHING, which handles "constrained segment variables".

Dealing with the "visual aspects" of data structures, the system CAN is useful as the programmers'assistant.

Keywords: Understanding, meta-interpretation, unification, meta-pattern-matching, propagation, generalisation.

INTRODUCTION

Considerable work is actually performed whose primary aim is to provide tools which help programmers to assure that the systems implemented fit their intended utilization.

Program verification (HOARE & LAUER 1973, IGARASHI LONDON & LUCKHAM 1975) can only verify properties that the programmer is expected to provide. A program verifier can do nothing, with a program on its own. Recent works on program understanding have demonstrated the need of domain dependant knowledge (GREEN BARSTOW 1975, RICH SCHROBE 1975) and pragmatic rules (WERTZ 1978).

Implemented systems in these areas are not equally efficient. That is, systems which depend on the use of domain dependant knowledge apply more easily to complex programs in the domain than to simple programs. Verification systems can be highly efficient for complex programs and extremely inefficient for simple ones.

Such systems lack a preliminary understanding process. Understanding a program is the ability to translate it in a special form, well suited to detect unsatisfied objectives.
That is, understanding is independant of the applications it allows: One can understand a program before even deciding to verify, correct, or improve it.

We propose the CAN system which understands statements by translating them in a canonical representation of their meaning.i:e the machine-state function they compute. This is different from denotational semantics (SCOTT STRACHEY 1971) since this understanding process deliberately applies to the "visual aspects" of programs. This restriction is supposed to prevent the understanding system from entering intricate problems when it is not relevant, and to pave the way for systems which would explicitly verify, correct, improve or reorganise programs.

THE NEED FOR A CANONICAL REPRESENTATION OF MEANINGS

Computer oriented approaches to semantic analysis of programs often consider the meaning of a program as it relates to other programs. For instance, (BOYER MOORE 1973, WALDINGER LEVITT 1973) would prove the following theorem:

(IF (EQUAL (APPEND L L) (APPEND M M)) (EQUAL L M) T)

as a step towards the meaning of an APPEND program.

- CAN translates (EQUAL (APPEND L L) (APPEND M M)) into its canonical meaning: "if L and M are lists, the value is T if L=M and NIL if L≠M". If APPEND is defined by a LISP program, CAN must understand it, that is, canonize it. Thanks to this representation of the meaning of APPEND, CAN will easily understand the following theorems:

- (EQUAL (APPEND L (APPEND M N)) (APPEND (APPEND L M) N))
- (EQUAL (CONS X (APPEND L M)) (APPEND (CONS X L) M))
- (IF (EQUAL L (APPEND L M)) (NULL M) M)

Instead, a theorem prover would have to prove them from the same LISP definition of APPEND, even if this definition is a bit awkward.

- Another point is the difficulty to fully express the meaning of a program by a set of relations, even if this meaning is "visual" (APPEND). CAN's understanding of APPEND is:

$$(\text{APPEND } (a_1\ a_2 \ldots a_n)\ (b_1\ b_2 \ldots b_m)) = (a_1\ a_2 \ldots a_n\ b_1 \ldots b_m)$$

THE CAN SYSTEM

CAN interprets programs in abstract environments. When given a LISP program, CAN symbolically executes its body in the most general environment (KING 1975, BOYER ELSPASS LEVIT 1975, WERTZ 1978, HEWITT 1975, YONEZAWA 1976).

To achieve this symbolic evaluation, CAN applies functions to classes of values. To evaluate the form (LENGTH L) in the abstract environment: $L = (a_1\ a_2 \ldots a_n\ \text{YES}\ b_1\ b_2 \ldots b_m)$ ① CAN builds the intersection pattern of ① and the pattern $(X_1\ X_2 \ldots X_y)$ which represents the class of inputs for which LENGTH is defined. This is done by a process called META-PATTERN-MATCHING (MPM) which unifies two patterns (see also STICKEL 1975, PATERSON 1976). In the present case, MPM would give:

$y = n + 1 + m$, $X_{n+1} = \text{YES}$, $X_1 \ldots X_n = a_1 \ldots a_n$, $X_{n+2} \ldots X_{n+1+m} = b_1 \ldots b_m$

A PROPAGATION module transmits this information to the entire current environment and the value y returned by LENGTH, which is now n+1+m.

This example illustrates the CAN language which describes sets of LISP elements by patterns, and functions over these sets by rules and sets of rules.

EXAMPLE

CAN's understanding of:

```
(IF (SETQ M (MEMQ X (APPEND L L)))
      (GT (LENGTH M) (LENGTH L))
    T)
```

in the abstract environment: X = !x L = !l

!x and !1 are variables whose domain is the set of LISP elements.
APPEND is defined as: (APPEND (?1) (?2)) = (?1 ?2).
?1 and ?2 are segment variables.

Here, APPEND is applied to !1 and !1. Unification of (!1 !1) and ((?1) (?2)) gives:

!1 = (?3) ?2 = ?3 ?1 = ?3

These results are propagated on (?1 ?2) which gives (?3 ?3). MEMQ is applied to !x and (?3 ?3). The CAN definition of MEMQ involves two cases:

$$(\text{MEMQ } !4\ ((\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?6)) = (!4\ ?6)$$

$$(\text{MEMQ } !4\ ((\mathbb{L}_i^n\ (\neg\ !4\ !5_i)))) = \text{NIL}$$

$(\mathbb{L}_i^n\ (\neg\ !4\ !5_i))$ represents the sequence $!5_1\ !5_2 \ldots !5_n$ where each $!5_i$ is different from the LISP element denoted by !4.

- In the first case, (!x (?3 ?3)) is unified with

$(!4\ ((\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?6))$, giving

$!x = !4,\ ?6 = ?7\ (\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?7,\ ?3 = (\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?7$

The value (!4 ?6) returned by MEMQ becomes

$$(!4\ ?7\ (\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?7)$$

- In the second case, M is assigned NIL and the whole form evaluates to T.

But in the first case, the form (GT (LENGTH M) (LENGTH L)) (1) must be evaluated in the abstract environment:

$L = ((\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?7),\ M = (!4\ ?7\ (\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?7),\ X = !4$

Clearly, the value of (LENGTH L) and (LENGTH M) will be respectively. n+1+a and 1+a+n+1+a where a is the length of (?7). CAN's definition of GT states that in all cases, (1) evaluates to T. CAN yields the final result:

$L = ((\mathbb{L}_i^n\ (\neg\ !4\ !5_i))),\ X = !4 \dashrightarrow T$

$L = ((\mathbb{L}_i^n\ (\neg\ !4\ !5_i))\ !4\ ?7),\ X = !4 \dashrightarrow T$

Union of the two rules gives the canonical meaning:

$L = (?9),\ X = !4 \dashrightarrow T$

GENERALISATION

This META-INTERPRETATION (MI) process may not terminate in the case of iterative or recursive programs. For example: (2)

```
(DE REVERSE (L M)
    (WHILE L
       (SETQ M (CONS (CAR   L) M))   } (3)
       (SETQ L (CDR L)))
    M)
```

CAN needs to symbolically evaluate (3) only once. This leads to the rule:

$L = (!1\ ?2),\ M = (?3)$ (4) $\dashrightarrow L = (?2),\ M = (!1\ ?3)$ (5)

MPM of these two environments represents the modified parts under the form of a "set of dynamic equations": ?3 = !1 ?3 | ?2 = !1 ?2 .

A GENERALISATION process permits to "solve" such equations. The result is the state of the environment after n iterations:

$L = (!1_1\ !2_2 \ldots !1_n\ ?2),\ M = (?3) \dashrightarrow L = (?2),\ M = (!1_n \ldots !1_1\ ?3)$ (6)

When the equation cannot be solved, one may use structural induction on the equations in order to prove theorems (BURSTALL 1969), since the equations explicitly represent the difference of complexity of data between two activations of the same process.
MI then proceeds. MPM seeks an "n" for which the process stops, by unifying the state of L after n iterations: L = (?2) with the state required to end the process: L = (). This gives (?2) = (). By PROPAGATION on (6), the final translation of REVERSE is:

$$(\text{REVERSE } ((\mathbb{L}_i^n\ !1_i)) (?3)) = ((\mathbb{L}_i^n\ !1_{n-i+1}) ?3)$$

VICIOUS CIRCLES

When do we stop MI and start to generalise or do induction ?
There is no reason to stop MI of (WHILE (CAR L) (SETQ L (LIST (CDR L)))) (7) though it contains a WHILE form, because it will stop on its own after two iterations at most. The solution is to stop at each detection of a "vicious circle". When MI of a LISP form F in an environment E leads to the MI of F in environment H, and E may be unified with H, a vicious circle is detected.
in example (2), F is (WHILE L ...), E is (4) and H is (5)
in example (7), E is L = ((¬ NIL !1) ?2), H is L = ((?2))
Now, ((¬ NIL !1) ?2) and ((?2)) may not be unified since ?2 should be both empty and non-empty. So, MI can proceed.

DISCUSSION

The property of canonicity of CAN statements is closely related to MPM's ability to unify patterns. Since we know of no general solution to the unification of patterns with segment variables, we cannot ascertain that every CAN statement is the only representation of the function it denotes. Canonicity is a goal towards which we must proceed by extending both language CAN's descriptive power and MPM's feasability.
That is, we do not think that the search for a general unifier is worthwhile since some unifications are not quite natural. An example is the unification of (?1 ?2) and (?2 ?1) whose result is the understanding of (EQUAL (APPEND L M) (APPEND M L)).

i:e: if L = $((\mathbb{L}_i^n\ ?3))$ and M = $((\mathbb{L}_i^m\ ?3))$ the value is T and NIL otherwise.

Rather, the motivation which underlies CAN is to understand in a straightforward manner the utmost of what would be considered by a programmer as "visually understandable".

CONCLUSION

What makes CAN an extremely useful program debugging tool is that the interpretation of CAN programs is a particular case of their meta-interpretation.
CAN has actually been run on large programs (like a pattern-matching algorithm) with some fifty CAN definitions of basic LISP functions at its disposal.
It is written in VLISP-10 (CHAILLOUX 1976, GREUSSAY 1976) and occupies about 10,000 cons-cells.
We intend to put CAN in the service of understanding systems which would be specialised in narrow areas.

REFERENCES

BOYER & MOORE (1973), "Proving theorems about LISP functions" Stanford university, Stanford CA. PP. 486-493.

BOYER, ELSPASS & LEVITT (1975) "SELECT--A formal system for testing and debugging programs by symbolic execution" Int. Conf. on Reliable Software, April 1975, pp. 234-245.

BURSTALL (1969) "Proving properties of programs by structural induction" Computer journal Fev. 1969 (12) (1) pp. 41-48.

CHAILLOUX (1976) "VLISP-10 manuel de reference" Département d'informatique, Université Paris 8, RT 17-76.

GREEN & BARSTOW (1975) "A hypothetical dialogue exhibiting a knowledge base for a program understanding system" Stanford A.I. Lab. Memo-aim 258, Computer sciece dept., Report n° stancs-75-476.

GREUSSAY (1977) "Contribution à la définition interprétative et à l'implémentation des lambda-langages", Thèse, Université Paris 8.

HEWITT & SMITH (1975) "Towards a programming apprentice" IEEE Transaction on Software Engineering, Vol.SE-1. pp. 26-45.

HOARE & LAUER (1973) "Consistent and complementary formal theories of the semantics of programming languages", Ibid 3, pp. 145-182.

IGARASHI, LONDON & LUCKHAM (1975) "Automatic program verification I: A logical basis and its implementation" Acta Informatica, Vol.4, pp. 145-182.

KING (1975) "A new approach to program testing" Int. Conf. on reliable software, April 1975, pp. 228-233.

PATERSON & WEGMAN (1976) "Linear unification", 8th annual SIGPLAN, MAY 76.

RICH & SCHROBE (1975) "Understanding LISP programs: Towards a programming apprentice", Master's Thesis, EECS M.I.T.

SCOTT & STRACHEY (1971) "Towards a mathematical semantics for computer languages", Proc. of Symp. on Computers and Automata, Polytechnic Institute of Brooklin, pp. 19-46.

STICKEL (1975) "A complete unification algorithm for associative-commutative functions", Fourth Int. Joint Conf. on Artificial intelligence (Tbilisi, Georgia, U.S.S.R.; Sept. 75).

WALDINGER & LEVITT (1973) "Reasonning about programs" A.I. center Technical note 86 SRI Project 2245.

WERTZ (1977) "Understanding LISP programs is improving LISP programs" Informatik-fachberichte n° 5, ed E.J. Neuhold Springer Verlag pp. 427-441.

YONEZAWA & HEWITT (1976) "Symbolic evaluation using conceptual representations for programs with side-effects" M.I.T. Artificial Intelligence Laboratory, AI-Memo 399.

A METHOD FOR COMPUTING HEURISTICS IN PROBLEM-SOLVING

Giovanni Guida - Marco Somalvico

Milan Polytechnic Artificial Intelligence Project
Milan-Italy

Abstract

The purpose of this paper is to propose a formal framework for structuring and embedding the heuristic information, in order to allow an algorithmic computation, in quite general cases, of the evaluation function $\hat{h}(n)$ of the classical Hart-Nilsson-Raphael algorithm. The notion of semantic graph is first introduced, in which the atomic notion of node is expanded by associating to it an internal structure where the heuristic information is inserted. It is proved that $\hat{h}(n)$ can be computed by solving an auxiliary problem, obtained from the original one, by adding new arcs, and of smaller complexity than that one. A class of algorithms for the computation of $\hat{h}(n)$ is then defined. The validity of the model proposed is discussed in detail.

Introduction

The problem of finding an optimal path through a graph in an efficient way is still a central one in many domains of scientific and engineering importance (Hart et al. 1968, Pohl 1970, Nilsson 1971). Heuristic search strategies, which aim to combine the two often contrasting requirements of generality and efficiency, have been proposed by several authors in the past (Hart,et al. 1968, Ernst 1969). These methods are all based on the assumption that the graph model can not embed all the relevant information for an efficient solution of the problem (Pohl 1970, Montanari 1970). Therefore, a supplementary heuristic information must be drawn from the semantic domain of the problem to be solved. This new information is then utilized to guide the search of the path through the graph. Depending on the formal way in which the heuristic information is expressed and on the manner in which it is used, different algorithms were proposed. We are interested in this paper in examining more closely the Hart-Nilsson-Raphael algorithm (Hart,et al., 1968) which still represents a central result within heuristic search. In this algorithm, the heuristic information is embedded in a function $\hat{f}(n)$, called evaluation function, which maps each node n of the graph in an integer. Nothing is still said about the manner in which the heuristic information can be explicitated in order to compute the evaluation function (more precisely, the $\hat{h}(n)$ part of $\hat{f}(n)$)in a general case).

The purpose of this paper is to propose a formal framework for structuring and embedding the heuristic information in order to allow an algorithmic computation, in a general case, of the function $\hat{h}(n)$ of the Hart-Nilsson-Raphael algorithm.

The central idea is to expand the atomic notion of node of a graph by associating to it an internal structure in which the heuristic information is inserted. Moreover, the notion of arc is expressed in a new logical formalism, more suitable than the algebraic one, for the processing required in the computation of $\hat{h}(n)$. It is then proved that $\hat{h}(n)$ can be computed, for each n, by solving an auxiliary problem, easily obtainable from the original one and of smaller complexity than that one. A class of algorithms is also proposed for the computation of $\hat{h}(n)$.

The implications of this new algorithms on the Hart-Nilsson-Raphael algorithm and the validity of the whole model are discussed in detail. Hints are given as well for a critical comparison and possibly for an abstract evaluation, between the computational complexity of the new algorithm proposed and the classical Dijkstra-Dantzig algorithm (Johnson 1973, Martelli 1977).

The results here presented constitute a part of the research effort carried on by the authors in developing a theory of problem solving (Sangiovanni Vincentelli,et al. 1973, Guida,et al. 1975 and 1977, Guida and Somalvico 1976).

Basic definitions

In this section we present the basic concepts and definitions of our theory. Let us first recall some fundamental definitions about graphs.

A (directed) graph is a couple $G = (N,E)$, where :

- N is a nonempty set of nodes;
- $E \subseteq N \times N$ is the set of the arcs of G.

A path of a graph $G = (N,E)$ from a node n_i to a node n_j is a string of nodes $\alpha = n_0 n_1 n_2 \ldots n_k$, s.t.:

$n_0 = n_i$

$(n_t, n_{t+1}) \in E$, for $t = 0,1,\ldots,k-1$

$n_k = n_j$

The length $|\alpha|$ of a path $\alpha = n_0 n_1 \ldots n_k$ is the integer k, i.e., the number of arcs in it.

On the basis of the formal structures above presented, we introduce now our first definition of problem, which strictly reflects the classical models proposed within the state-space approach (Nilsson 1971, Sangiovanni Vincentelli and Somalvico 1973). We denote this definition as syntactic description. In fact, it is only structural, or syntactic, in the sense that the internal nature of nodes and arcs is not taken into account and these elements are just considered as atomic concepts. This simple model will constitute the skeleton of the new representation that we shall introduce later on and that we shall denote as semantic description.

A problem is a fourtuple $P = (N,E,i,F)$, where (N,E) is a graph and :

- $i \in N$ is the initial node;
- $F \subseteq N$ is the set of the final nodes.

A solution x of a problem $P = (N,E,i,F)$ is a path of (N,E) from i to a node $f \in F$.

A solution $\bar{x}$ of a problem P is minimal iff :

$$\bar{x} = \min\{|x| \mid x \text{ is a solution of } P\}.$$

The semantic description of the state-space approach to problem solving we are now going to introduce is just a particular case of the more general concept of problem task presented in (Guida and Somalvico 1976). This particular case has been choosen in order to show in detail how the introduction of semantic elements into old syntactic models can allow the definition of more efficient search algorithms. In fact, the semantic description is just intended for enabling the computer to process and utilize in the correct way, by means of a fully automatic procedure, some heuristic informations, supplied to it within the problem representation itself. In particular, the central purpose of the semantic model we propose is to constitute a formal basis for structuring and embedding the heuristic information in order to allow an algorithmic computation, in a general case, of the evaluation function of the Hart-Nilsson-Raphael algorithm. The richer information, which is proper of the semantic description, is embedded in the state-space representation by expanding the notion of state, and by associating to each node a structure in which the new information is inserted (Sangiovanni Vincentelli and Somalvico 1973, Guida and Somalvico 1976).

Before presenting the definition of semantic problem, let us enrich, from a formal point of view, the graph structure above presented.

Let:

- $A = \langle a_1, a_2, \ldots, a_k \rangle$ a k-tuple of attributes;
- for $i = 1,2,\ldots,k$, V_i a finite nonempty set of values, associated to the attribute $a_i \in A$ (where the symbol $\in$ denotes the "membership" in the k-tuple), called the domain of a_i;
- $V = \langle V_1, V_2, \ldots, V_k \rangle$ the K-tuple of domains.

An attribute-value couple for an attribute $a_i \in A$ is a couple $c_i = (a_i, v_i)$, where $v_i \in V_i$.

The attribute-value couple set associated to an attribute $a_i \in A$ is the set:

$$C_i = \{ c_i \mid c_i = (a_i, v_i) \wedge v_i \in V_i \}$$

A semantic node (S-node) $\bar{n}$ is an element of the set :

$$\bar{N} = \bigtimes_{a_i \in A} C_i$$

which is called the universal S-nodes set defined on the couple (A,V).

A (directed) semantic graph (S-graph) is a fourtuple $\bar{G} = (A, V, \Pi, \Lambda)$ where A and V are, as in the above definition, and :

- Π is a finite set of predicates of one argument on the set $\bar{N}$, called properties;
- Λ is a finite set of predicates of two arguments on the set $\bar{N}$, called legal conditions.

The predicates $\pi \in \Pi$ and $\lambda \in \Lambda$, which are expressed in some logical calculus, are not unique; i.e., they can have several equivalent expressions (Nilsson 1971, Sangiovanni Vincentelli and Somalvico 1973).

We can therefore assume that they must be set in some normal form.

The logical expressions of π and λ are formulated by utilizing the attributes and the values.

To each S-graph it is possible to associate exactly one underlying graph, which constitutes the more elementary syntactic skeleton of the S-graph. More precisely, the underlying graph of the S-graph $G = (A,V,\Pi,\Lambda)$ is the graph $G = (N,E)$, s.t.:

$$N = \left\{ \bar{n} \in \bar{N} \mid (\forall \pi)\ ((\pi \in \Pi) \supset \pi(\bar{n})) \right\}$$
$$E = \left\{ (\bar{n}_1,\bar{n}_2) \mid (\forall \lambda)\ \ ((\lambda \in \Lambda) \supset \lambda(\bar{n}_1,\bar{n}_2)) \right\}$$

Note that the concepts of graph and S-graph reflect two different generation strategies, implicitely contained in their definitions. In fact, the definition of a syntactic graph is obtained by specifying (by means of the set E) which nodes of an originally fully separate graph (i.e., a graph having only the nodes in N and no arcs) have to be connected by an arc. On the other hand, the S-graph definition is obtained by specifying (by means of the sets Π and Λ) which nodes, within a set of possible ones, belong to the graph and which arcs have to be pruned from an originally complete graph.

The concepts of path and length are still valid and maintain their meaning for S-graphs by referring them to the underlying graphs.

Let us present now the definition of semantic problem.

A semantic problem (S-problem) is a sixtuple:
$P = (A,V,\Pi,\Lambda, i,F)$, where (A,V,Π,Λ) is an S-graph and :

- $i \in N$ is the initial S-node;
- $F \subseteq N$ is the set of the final S-nodes;

if (N,E) is the graph underlying (A, V,Π,Λ).
The concepts of solution and minimal solution above presented are still valid for S-problems.

Let us illustrate this definition by means of a simple example. We will consider the classical well know Eight puzzle and we refer for its description to the literature (Nilsson 1971).

In this problem, we identify, according to the previous definitions, the following 9-tuple A of attributes :

$$A = \langle a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8 \rangle$$

where a_0 is the empty tile and a_i, for $i = 1,2,\ldots,8$, is the i-th tile. For $i = 0,1,\ldots,8$, the set of values V_i associated to the tile a_i is the set of all the possible positions of the i-th tile.

Recalling the description of the puzzle, it is trivial to observe that the sets of values associated to every tile are all equal, namely:

$$V_i = \bar{3}\times\bar{3} \quad \text{for } i = 0,1,2,\ldots,8$$

where $\bar{3}$ is the set of the first three natural numbers, i.e.:

$$\bar{3} = \{1,2,3\}$$

For each $v_i = (x,y) \in V_i$, the first element x denotes the row occupied by the tile a_i, while the second element y denotes the column occupied by the tile a_i.

In order to complete the semantic description of the puzzle, the sets of the properties and of the legal conditions Π and Λ have to be individuated. In this puzzle the only condition to be fulfilled is that two tiles are not allowed to occupy the same position. Therefore we introduce the following property π_1, defined as a logical predicate of one argument on $\bar{N}$ (recall that : $\bar{n} = ((a_0,v_0),(a_1,v_1),\ldots,(a_8,v_8))$):

$$\pi_1(\bar{n}) \equiv (\text{for } i, j = 0,1,\ldots,8 \mid (i \neq j) \supset (v_i \neq v_j))$$

According to the description, we have to formalize that only two tiles can be moved in the same time. Therefore we define the legal condition λ_1:

$$\lambda_1(\bar{n}', \bar{n}'') \equiv ((\exists !i)(\exists !j) \quad ((v'_i \neq v''_i) \wedge (v'_j \neq v''_j)))$$

(where $(\exists !i)$ means: it exists exactly one i such that).

The legal condition λ_2 is related to the fact that one of the moved tiles has to be the empty tile :

$$\lambda_2(\bar{n}', \bar{n}'') \equiv (v'_0 = v''_0)$$

Another fact, which is defined by the legal condition λ_3, expresses that the exchange can be done if the two tiles are on the same row or on the same column:

$$\lambda_3(\bar{n}', \bar{n}'') \equiv ((v'_i \neq v''_i) \supset ((x'_i = x''_i) \vee (y'_i = y''_i)))$$

The last legal condition λ_4 is related to the fact that the two tiles have to be adjacent to each other :

$$\lambda_4(\bar{n}', \bar{n}'') \equiv ((v'_i \neq v''_i) \supset (((x'_i - x''_i) \in \{-1,0,1\}) \wedge ((y'_i - y''_i) \in \{-1,0,1\})))$$

Let us outline that the new S-problem model introduced is adequate for the purpose we proposed it for. In fact, the S-graph structure on which it is based can embed more information than the graph model. Moreover, it represents a valid formalism for representing problems within the state-space approach. We shall show, in the next section, how the supplementary information contained in the S-problem model can be effectively used for the precise definition of the heuristically guided search algorithm.

Computing the evaluation function

This section is devoted to present a formal framework which allows the automatic computation of the function $\hat{h}(n)$ on the basis of the supplementary information embedded in the problem representation prepared. An algorithm which performs the proposed technique is provided.

In the following, we assume the knowledge of the Hart-Hilsson-Raphael algorithm (Hart et al. 1968).

Let us first introduce a preliminary definition.

Given an S-problem $\bar{P} = (A,V,\bar{\Pi},\Lambda,i,F)$, an auxiliary S-problem for $\bar{P}$ is an S-problem $\bar{P}'=(A',V',\Pi',\Lambda',i',F')$, s.t.:

$A'=A$; $V'=V$; $\bar{\Pi}'=\bar{\Pi}$; $\Lambda'\subseteq\Lambda$; $i'=i$; $F'=F$.

$\bar{P}'\sqsubseteq\bar{P}$ denotes that $\bar{P}'$ is an auxiliary S-problem of $\bar{P}$.

Theorem

Let $\bar{P}'\sqsubseteq\bar{P}$ and $\bar{x}'$ a minimal solution of $\bar{P}'$; then $|\bar{x}'|\leq h(i)$.

Proof: The formal proof of the theorem is quite easy to be obtained,but also rather heavy to express.Let us only recall,in an informal way,its main steps. Because of the definition of auxiliary S-problem, the graph underlying $\bar{P}'$ (more precisely, $\bar{G}'=(A',V',\bar{\Pi}',\Lambda')$) is more connex, i.e., it has more arcs, than the graph underlying $\bar{P}$(more precisely, $\bar{G}=(A,V,\bar{\Pi}\ \Lambda)$). In other words, the set of the arcs of $\bar{P}$ is exactly a subset of the set of the arcs of $\bar{P}'$.
Moreover, h(i) is the actual length of a minimal solution $\bar{x}$ of $\bar{P}$. Therefore the thesis follows.

This theorem constitutes the basis for a modified version of the classical Hart-Nilsson-Raphael algorithm, that we are going to present.

Algorithm G

1. Given the S-problem $\bar{P}=(A,V,\bar{\Pi},\Lambda,i,F)$,put i in the OPEN list, compute all the successors of i, and put them in OPEN.
2. Construct, for all the k successors $\bar{n}$ of i an auxiliary S-problem $\bar{P}'_{\bar{n}}\sqsubseteq\bar{P}_{\bar{n}}$, where $\bar{P}_{\bar{n}}=(A,V,\bar{\Pi},\Lambda,\bar{n},F)$. All the k auxiliary S-problem $\bar{P}'_{\bar{n}}$ must share the same subset Λ' of Λ.
3. For each auxiliary S-problem $\bar{P}'_{\bar{n}}$, find the minimal solution. The k solutions are obtained in parallel thus constructing within the graph G' underlying the auxiliary S-problem $\bar{P}'$ a solution forest $T'\subseteq G'$ the leaves of which are all the initial nodes of the k auxiliary S-problems $\bar{P}'_{\bar{n}}$, the roots of which belong to the set F of final states.
4. Compute the evaluation function of the nodes in OPEN on the base of the costs of the solutions obtained in 3.
5. Apply the Hart-Nilsson-Raphael algorithm until new nodes are inserted in OPEN.
6. For the new nodes inserted in OPEN find in G' the minimal distance from T'. Assume such distance as a subminimal solution of the auxiliary S-problems associated to the new successors and go back to 3.

Conclusions

The algorithm proposed doesn't specify how to construct the auxiliary S-problem $\bar{P}'_{\bar{n}}$ (i.e., how to choose $\Lambda'\subseteq\Lambda$). The choice of Λ' implies the precise evaluation of the relations existing between the following three entities: extent of Λ' in comparison to Λ , goodness of the estimate $\hat{h}(\bar{n})$ compared to h(n), complexity of the resulting auxiliary S-problem $\bar{P}'_{\bar{n}}$. Clearly,both $\hat{h}(\bar{n})$ and the complexity of $\bar{P}'_{\bar{n}}$ increase as Λ' approaches Λ. In particular, if $\Lambda'=\Lambda$ we obtain the optimal estimate $\hat{h}(\bar{n})=h(\bar{n})$, but the solution of $\bar{P}'_{\bar{n}}$ (possibly by means of the Dijkstra-Dantzig algorithm) requires too much computational effort. A more precise insight into this question can be obtained by

introducing the definition the power of each legal condition $\lambda \in \Lambda$. The power $\| \lambda \|$ of a legal condition $\lambda \in \Lambda$ is the integer

$$\| \lambda \| = \# N' - \# N$$

(where # A denotes the cardinality of the set A) where :

-N' is the set of the arcs of the graph underlying the S-graph having $\Lambda' = \Lambda - \{\lambda\}$ as legal condition set;

-N is the set of the arcs of the graph underlying the S-graph having Λ as legal condition set.

The power of a legal condition λ gives an estimate of the greater connexness of the graph underlying the auxiliary S-problem P'_n obtained by setting $\Lambda' = \Lambda - \{\lambda\}$.

Clearly, the computation of the power of each legal condition requires an effort (run time and memory storage) which must also be carefully taken into account.

We are now developing a theoretical investigation on evaluating the trade off on the complexity of the different techniques to adapt for choosing the set Λ' (Johnson 1973, Martelli 1977). An experimental activity for the actual testing of the method is also planned.

References

Chang C.L., Slagle J.R. - "An Admissible and Optimal Algorithm for Searching AND/OR Graphs" - Art. Int. 2 (1971), pp. 117-128.

Ernst G. - "Sufficient Conditions for the Success of GPS"-J. ACM, Vol. 16, pp. 517-533, October 1969.

Guida G., Mandrioli D., Somalvico M. - "An Integrated Theory of Problems as an Algebraic Base for Complexity Understanding and Automatic Problem Solving" - Proc. 7th IFIP Conf. on Optimization Techniques, Nice, September 1975, Springer Verlag, Heidelberg.

Guida G., Mandrioli D., Somalvico M. - "An Integrated Model of Problem Solvers" - Information Sciences 13, pp.11-33,1977

Guida G., Somalvico M. - "Semantics in Problem Representation and Search"- Inf. Proc. Letters, 5,5 - pp. 141-145, 1976.

Hart P.E., Nilsson N.J., Raphael B. - "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"-IEEE Trans. Sys. Sci. Cybernetics, Vol. SSC-14, no. 2, pp.100-107, July 1968.

Johnson D.B. - "A Note on Dijkstra's Shortest Path Algorithm"-J. ACM Vol. 20, pp.385-388, July 1973.

Martelli A. - "On the Complexity of Admissible Search Algorithms"- Art. Int. 8(1977), pp. 1-13.

Montanari U. - "Heuristically Guided Search and Chromosome Matching"- Art. Int. 1(1970), pp.227-245.

Nilsson N.J. - "Problem-Solving Methods in Artificial Intelligence"- Mc Graw-Hill, New York, 1971.

Sangiovanni Vincentelli A., Somalvico M. - "State-Space Approach to Problem-Solving Optimization - Proc. 5th IFIP Conf. on Optimization Techniques, Roma, May 1973, Springer Verlag, Heidelberg.

HAM-RPM: NATURAL DIALOGUES WITH AN ARTIFICIAL PARTNER

W. v. Hahn, W. Hoeppner, A. Jameson, W. Wahlster
Germanisches Seminar, Universität Hamburg, 2000 Hamburg 13, West Germany

ABSTRACT: This paper introduces the understanding system HAM-RPM, which simulates a dialogue partner conversing about a real-life domain. After outlining the system's overall structure, we discuss three of its distinguishing features: The first is its organization of spatial data in a redundant multiple data base, inspired by certain aspects characteristic of visual search in humans. A new algorithm for noun-phrase generation is then sketched which is sensitive to the conversational state and uses a 'worst-case-first' strategy. Finally, we describe in some detail a specific operationalisation of the notion of the communicative relevance of objects. The paper concludes with a summary of the objectives of this research.

DESCRIPTIVE TERMS: natural language processing, dialogue simulation, focus of attention, reference semantics, visual search, FUZZY

1. SYSTEM OVERVIEW

The AI system HAM-RPM [11] can be classified as a parsed, content-motivated natural language system [14]. It simulates a dialogue partner conversing in German [12]. At present we are testing the system on two different domains: the interior of a living room and a natural traffic scene. The former is a typical AI toy world, the latter a real-life domain with many challenging aspects [1],[2]. The conversational setting is as follows: the system is observing the scene through an artificial eye; the user, who cannot see the scene but who is familiar with the setting or has a photograph of it, asks the simulated dialogue partner about its current state.

The system is implemented in FUZZY, a LISP-embedded PLANNER-type language which provides a number of facilities for representing and manipulating fuzzy knowledge [8].

The heterogeneous knowledge base integrates various procedural and declarative representation languages, including production rules, semantic nets,

pattern-operation rules and FUZZY antecedent theorems. Large parts of the knowledge are coded into a conceptual and a referential network, both of which are composed around a set of primitive structural and assertional links [16]. In the conceptual net facts are represented which are true in all possible states of the micro-world, whereas the referential net contains facts true in the current state. So that a user unaquainted with the program can alter the knowledge base, the basic system retrieves all of its specific knowledge from external files (Fig. 1) written in a self-explanatory notation.

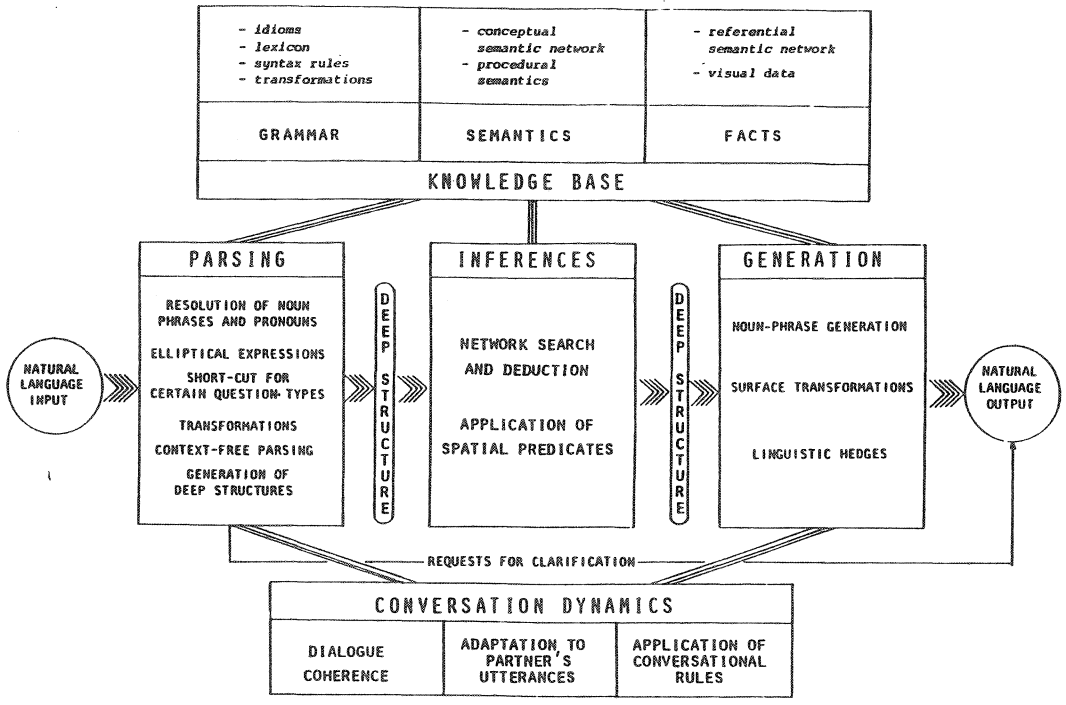


Fig. 1

During the processing of an input sentence interacting semantic, pragmatic and syntactic procedures are activated. However certain types of input, e.g. simple 'which'-questions, are processed without complete parsing by specialized and therefore very efficient inference procedures. Definite noun phrases are replaced by referential equivalents. The referents of pronouns are determined by consulting a special CONTEXT which contains a record of previously mentioned objects. The criteria applied are gender, number and recency of mention. The system can ask the user for any further information it requires at various points during the dialogue.

HAM-RPM handles elliptical expressions; this is of great importance in

natural conversations, where the sentences used are by no means always complete. If the system's attempts to parse an input as a complete sentence fail, it backs up and starts an ellipsis-recognition process: the analysed elliptical expression is matched against the shallow structure of the previous sentence and expanded to a complete sentence. An improvement over the ellipsis facility of LIFER [7] is that for the construction of the deep structure of the expanded sentence full parsing is not performed; instead, particular constituents are projected onto the deep structure of the previous input.

As our emphasis, like that of the projects described in [3],[5], is on natural dialogues, the handling of fuzzy knowledge, including the generation of linguistic hedges, is of major importance for our system. A certain degree of communicative competence is embodied in the generation component of the system. Thus the form of the system output is determined in part by the current focus of attention [4].

2. THE HANDLING OF SPATIAL RELATIONS

Spatial relationships are often mentioned in the dialogue situation described above, on the one hand because they form the topic of certain questions, e.g. *How many people are standing to the right of the old woman?*, and on the other hand because they are frequently used to help identify objects (see below, section 3).

Although the locations of our objects are represented simply by single points in a 2-D coordinate system, we have found it worthwhile to develop a relatively complex structuring of the visual data which makes it possible to model certain aspects of the visual search process in humans: While searching within a scene for an object which satisfies a given description, a human being can use the information available during a given fixation not only to examine an object located in the center of the visual field, but also to direct his eye movements toward potentially relevant peripheral objects [13]. This capacity, which in humans is based on low-level parallel processing, can be approximated through the use of a redundant multiple data base[1] [9]: For each point $\dot{P}$ in our coordinate system a separate

[1] The efficiency of this realization depends on the way in which multiple data bases are implemented; FUZZY's implementation is favorable in this respect.

CONTEXT, including P's coordinates in its name (e.g. 'C14-5') is established, which contains at any moment more or less the same information as would be available to a human being fixating the corresponding point in a photograph of the scene, i.e.

- detailed information about the object (if there is one) located exactly at point P; as well as
- a few facts about the objects located near P: their location and salience, plus other simple properties such as color, size, and object type.

Thus each object in the scene has detailed information on it entered in exactly one CONTEXT and less detailed data in a number of adjoining ones. The number of these depends on the presumed prominence of the object within the scene, and is computed by demons responsible for maintaining the correct distribution of such information among the CONTEXTs.

An example will illustrate the kind of search procedure which can take advantage of this way of structuring the visual data: To determine the referent of the noun phrase *the woman standing to the left of the traffic light*, assuming that the location of the traffic light has already been determined to be P, the system computes a list of the names of several of the CONTEXTs which lie in the area to the left of P, the CONTEXTs being close enough to each other so that it is likely that any object within this area will be mentioned in at least one of them. It then enters these CONTEXTs one at a time, within each looking for an entry indicating that a particular object is a person. When it finds one, it retrieves the object's location and switches to the corresponding CONTEXT to obtain more detailed information on it e.g. whether it is a woman (see below, section 4 for a more detailed example). Our reasons for adopting this structure of the visual data, despite its high cost in terms of memory, are three:

- It provides a framework within which hypotheses concerning certain types of visual search in humans can be formulated precisely. One of us is presently comparing such models with experimental data.
- It will permit us to explore the human-engineering hypothesis that man-machine communication can be improved through the assimilation of the internal processes of the machine to those of the man (see below, section 3).
- It permits tasks such as the one described above, which otherwise often involve long linear searches through sets of objects or points in space, to be performed by highly selective algorithms: the individual CONTEXTs are small and independently indexed, and only the relevant ones are searched. The resulting speed is vital for an on-line dialogue system.

3. NOUN PHRASE AND OBJECT

Two major problems of reference semantics confronting builders of understanding systems are noun-phrase (NP) resolution, i.e. the determination of the referent of a definite NP, and NP generation, i.e. the construction of NPs to identify objects uniquely. HAM-RPM's NP interpreter works on the shallow structure of the input sentence, which is generated through the replacement of multiple-word phrases and idioms by canonical expressions [10], a dictionary look-up and a simple morphological analysis. In a way reminiscent of SHRDLU [15], semantic/pragmatic processes like NP resolution are activated in HAM-RPM as soon as possible; this saves a great deal of unnecessary effort, since when existential presuppositions or selectional restrictions fail there is no further parsing, but rather feed-back to the conversational partner. The method we have developed for the inverse process, NP generation, is distinguished from earlier approaches (for an overview see [6]) by two aspects: The first is its use of a 'worst-case-first' strategy. Here is a sketch of the algorithm as applied to an object X:

1) If X has a proper name, this is used.
2) If X has no cohyponyms i.e. it has no siblings in the ISA-hierarchy the identifying NP is formed using the definite article and the type-node of the token-node.
3) If cohyponyms of X are found, it is first determined whether there are any properties of X which should not be mentioned in its description because they have already been presupposed in the question,e.g. *red* in the question *Which car is red?*; any such properties are excluded from further consideration. The algorithm then checks for the worst case, i.e. the existence of cohyponyms for which exactly the same properties are stored in the referential net as for X. If such an object Y is found, the system tries to distinguish the two in terms of spatial relations, in one of two ways: if X and Y are within a certain radius of each other, X is distinguished using an expression such as *the right hand ...* or *the second ... from the left*. Otherwise, the position of X relative to the objects in its vicinity is described. This process is recursive, because the system has to identify the objects which serve as spatial reference points for X (e.g. *the red car which is parked behind the tree which is to the left of the large building*). The system must then check whether the next-worst case obtains as well:
4) If the characterizing properties of X are a subset of those of a cohypo-

nym Y, X is characterized in terms of the properties it lacks (e.g. *the red, but not old car*).

5) The system then looks among the properties of X for one which distinguishes it from its cohyponyms.
6) If there is no such property, all properties of X in the referential net are used, with no further attempt to find a minimal characterizing set.

The generator's second novel aspect is the kind of selectivity which is necessary in a world with a large number of objects, where

- usually there are many possible ways of uniquely identifying an object, and
- it becomes important to choose one which makes it easy for a human listener looking at the scene to locate the object.

Our generator in its present form does justice to these considerations in the spatial parts of its descriptions, since the system's spatial search processes are biased in favor of objects as reference points which are visually easy to find starting from a given point, or which have recently been attended to (see below, section 4). By contrast, the decision as to whether to mention spatial relations or other properties in cases where both are possible (e.g. *the car in front of the big tree* vs. *the small brown parked car*) is still made without consideration of the ease of interpreting the resulting characterization. This is because the algorithm has not yet been adapted to provide for the retrieval of information on the object's non-spatial properties from the multiple data base described in section 2 above, in which the visual accessibility of a given property of a particular object is reflected in the number of CONTEXTs in which it appears.

4. THE SEARCH FOR RELEVANCE

The specific criteria used to determine an object's relevance depend on the question and the context in which it is asked. One such set of criteria, appropriate within our dialogue situation to questions of the form *What's in front of the big tree?*, is embodied in the FUZZY deduce procedure NEIGHBOURS, which we shall describe in order to illustrate three general principles governing the selection of relevant objects and the way in which their implementation can be facilitated through the use of certain characteristic mechanisms of the FUZZY language.

NEIGHBOURS (see Fig. 2) is a generator in the CONNIVER sense i.e. when it is called with an argument like (IN-FRONT-OF OBJECT13) it returns exactly one neighbour of OBJECT13, but it can be restarted to produce more.

```
1  (PROC NAME: NEIGHBOURS DEMON: TIMES-DEMON (?RELATION ?OBJECT1)
2        (GOAL (LOCATION !OBJECT1 ?LOCATION1))
3        (FOR ?CONTEXT &(ADJOINING-CONTEXTS !RELATION !LOCATION1)
4             (CONTEXT !CONTEXT)
5             (FOR FETCH: (SALIENT ?NEIGHBOUR)
6                  (GOAL (!RELATION !NEIGHBOUR !OBJECT1))
7                  (GOAL (NEAR !NEIGHBOUR !OBJECT1))
8                  (Z + 1 (RECENTLY-MENTIONED !NEIGHBOUR))
9                  (SUCCEED? !NEIGHBOUR ZACCUM))))
```

Fig. 2

Principle 1 The search for relevant objects should be so designed that the most relevant ones will tend to be found early. This is accomplished in NEIGHBOURS in the two iteration loops in lines 3 through 9. Lines 3 and 4 provide for the CONTEXTs 'in front of' OBJECT13 to be searched starting with the nearest and moving away (the LISP procedure ADJOINING-CONTEXTS returns an appropriately ordered list of CONTEXTs). A further, more general technique for ensuring that the most relevant objects will be found early is applied in line 5, in which a basic feature of the FUZZY language is exploited: Each entry in the associative data base has associated with it a 'z-value', e.g. ((SALIENT OBJECT1) .4). The system automatically keeps the entries ordered according to z-value, so that an iterated FETCH generates entries in monotonic order of z-value. Here, this means that, within each CONTEXT, the objects are retrieved for examination in decreasing order of salience.

Principle 2 The relevance of an object is often a function of several factors which can be combined numerically. The four factors in our example appear in lines 5 through 8: the visual salience of the neighbour; the degree to which it is truly in front of OBJECT13 (i.e. as opposed to being merely 'more or less in front of' it); and the extent to which it has been mentioned recently.[1] The FUZZY consequent theorems called in lines 6 and 7 (by pattern) and in line 8 (by name) return, like the FETCH of line 5, both a skeleton and a z-value representing how high the neighbour rates according to the criterion in question. These z-values needn't be explicitly manipulated, as they are automatically multiplied together by the 'procedure demon' called TIMES-DEMON specified in line 1. Their product, the value of ZACCUM, which can be interpreted as the total relevance of the neighbour in question, is returned in line 9 as the z-value associated with it.

[1] All references to objects during the dialogue are recorded in a separate CONTEXT 'references', which is also consulted during pronoun resolution.

The function Z in line 8 permits the z-value to be changed before it is passed to the demon; in this case it is increased by 1 so that objects not recently mentioned will not be judged entirely irrelvant.

Principle 3 There must be a rule for determining how exhaustive the search for relevant objects should be and how many of them should be mentioned. In our example, the rule has the form 'Find six objects with a relevance above the threshold of .2, and mention the three most relevant of these', where the numbers 3, 6 and .2 are values of parameters which depend on the course of the dialogue, for example the number of times the human questioner has expressed a desire for a detailed answer.

The FUZZY iteration in Fig. 3 causes those objects generated by NEIGHBOURS with a z-value above .2 to be stored temporarily in a seperate CONTEXT called

```
(FOR TRY: (NEIGHBOURS) (IN-FRONT-OF OBJECT13) ZVAL: .2
          (CONTEXT (QUOTE STM))
          (ADD &(VAL) (ZVAL))
                      ...)
```

Fig. 3

STM; when this CONTEXT has collected six elements, three FETCHes will automatically retrieve the three most relevant, because of the ordering in terms of z-value.

5. CURRENT STATUS OF IMPLEMENTATION

A non-compiled version of HAM-RPM, implemented in FUZZY, is currently running on the PDP-10 of the Fachbereich Informatik in Hamburg under the TOPS10 operating system. Incorporating all of the features described in this paper, it occupies 55k of core and requires from 1 to 15 seconds of CPU time to generate a response. The program is user engineered, robust, and debugged enough so that it can be run by naive users.

6. GOALS AND METHODOLOGICAL PRINCIPLES

HAM-RPM's character as a system for the investigation of dialogue structures and the associated cognitive processes implies certain methodological principles:

1. the principle of consistent dialogue simulation

Communication with the system will consist exclusively of utterances appro-

priate to a natural dialogue between two human partners, even when it is meta-communicative in nature.

2. the principle of the pragmatic/semantic foundation

The emphasis will be on the pragmatic/semantic aspects of non-technical conversations. By this we mean not so much the superficial linguistic features as the type of speech-act sequences which are involved in this kind of dialogue.

3. the principle of complete implementation

A complete running version of the system will be maintained at all times, so that changes and extensions can be evaluated in relation to the model as a whole.

4. the principle of domain-independence

The system's domain-independence will be ensured by a clear distinction between the basic program and all representations of the knowledge specific to a particular type of scene.

Two goals on which we shall concentrate in the near future are:

- the improvement of the meta-communicative capacity of the system - while adhering to the methodological principle of consistent dialogue simulation - with a view to the extension of its repertoire of communicative strategies.
- the development of a systematic representation of time, so that e.g. the movements of objects in a sequence of pictures can be described.

ACKNOWLEDGEMENT

This research is currently being supported by the Deutsche Forschungsgemeinschaft.

REFERENCES

[1] Badler, N. I. (1975): Temporal Scene Analysis: Conceptual Descriptions of Object Movements. Technical Report No. 80, Dept. of Computer Science, University of Toronto

[2] Bajcsy, R. / Joshi, A. K. (1977): Partially Ordered World Model and Natural Outdoor Scenes. Report UP-MS-CIS-77-63 (presented at Workshop on Computer Vision, Univ. of Mass., Amherst)

[3] Bobrow, D.G. / Kaplan, R.M. / Kay, M. / Norman, D.A./ Thompson, H./ Winograd, T. (1977): GUS: A Frame-Driven Dialog System. *Artificial Intelligence, 2, April 1977, 155-173*

[4] Grozs, B.J. (1977): The Representation and Use of Focus in a System for Understanding Dialogs. *Proceedings of the 5th IJCAI, Cambridge, Mass., 67-76*

[5] Hayes, P.J. / Rosner, M.A. (1976): ULLY: A Program for Handling Conversations. *Proceedings of the AISB Summer Conference, Edinburgh, 137-147*

[6] Heidorn, G.E. (1977): Generating Noun Phrases to Identify Nodes in a Semantic Network. *Proceedings of the 5th IJCAI, Cambridge, Mass., 143*

[7] Hendrix, G.G. (1977): The LIFER Manual. A Guide to Building Practical Natural Language Interfaces. Technical Note 138, Stanford Research Institute.

[8] LeFaivre, R.A. (1974): Fuzzy Problem Solving. Technical Report No. 37, Univ. of Wisconsin, Madison

[9] McDermott, D.V. (1974): The CONNIVER Reference Manual. MIT-AI-Memo No. 259a

[10] Parkison, R.C. / Colby, K.M. / Faught, W.S. (1977): Conversational Language Comprehension Using Integrated Pattern-Matching and Parsing. *Artificial Intelligence, 9, 111-134*

[11] Wahlster, W. / v. Hahn, W. (1976): Einige Erweiterungen des natürlichsprachlichen AI-Systems HAM-RPM. *Laubsch, J.H. / Schneider, H.-J. (eds.): Dialoge in natürlicher Sprache und Darstellung von Wissen. Freudenstadt, 204-225*

[12] Waltz, D.L. (ed.) (1977): Natural Language Interfaces. *Sigart Newsletter, No. 61, Febr. 1977, 16-65*

[13] Williams, L.G. (1967): The Effects of Target Specification on Objects Fixated During Visual Search. *Sanders, A.F. (ed.): Attention and Performance. Amsterdam: North-Holland, 355-365*

[14] Wilks, Y. (1974): Natural Language Understanding Systems within the AI Paradigm: A Survey and Some Comparisons. Memo Nr. 237, AI-Lab., Stanford Univ.

[15] Winograd, T. (1972): Understanding Natural Language. N.Y.: Academic

[16] Woods, W.A. (1975): What's in a Link? Foundations for Semantic Networks. *Bobrow, D.G. / Collins, A. (eds.): Representation and Understanding. N.Y.: Academic, 35-82*

A MULTIPROCESSOR SYSTEM FOR KNOWLEDGE REPRESENTATION AND MANIPULATION

F.K. HANNA and W.T. HARWOOD
University of Kent, Canterbury, Kent, U.K.

Descriptive Terms

Multiprocessor system, knowledge representation, parallelism, LISP, microprocessors.

Abstract

Knowledge manipulation tasks are computationally intensive : one way to provide sufficient computing power is to use a loosely coupled network of microprocessors. For this approach to be efficient it is necessary that the overall task be divisible into parts which can be run in parallel and which are, to a degree, self-contained. It is argued that contemporary approaches to knowledge manipulation systems fulfil these criteria. The paper describes the design of such a system, currently under construction. It is programmed in a LISP-like language which incorporates parallelism, and allows programs to be run in remote environments. Aspects of this language, its implementation and the underlying hardware are described.

Introduction

Out of the last decade's work on systems for manipulating "real-world" knowledge (e.g. language understanding systems) perhaps one of the few universally accepted results to emerge has been the observation that, as a system's universe of discourse is widened, so the amount of processing power it requires rises combinatorially. Typically such systems are run on large, fast, dedicated machines and their problem-solving capabilities are bounded by the amount of processor time available. Now it is generally agreed that already contemporary computer architecture is approaching the point of diminishing returns : technological advances seem unlikely to result in many more than a couple of orders of magnitude increase in processing speed. In order to realise knowledge manipulating systems we are therefore motivated to investigate alternative structures which have the potential for providing greater amounts of processing power. Obvious candidates at the present time are systems consisting of very large numbers of interconnected microprocessors. Such systems are attractive (in hardware terms) because it is becoming possible to replicate individual processors, memories and interfacing components very cheaply. Indeed, within the next decade one can foresee systems consisting of upwards of 10^4 such elements being economically viable. It is the purpose of this paper to discuss some aspects of a system of this nature which has been designed and is presently under construction.

§1 Overall Organisation

We start then with the stated aim of implementing a knowledge manipulation (henceforth KM) system on a loosely coupled network of small processors. The problem domain we are working within is natural language understanding, such as is well exemplified by systems for interrogating

large, complex data bases (e.g. Woods et al.'72). At the outset it is apparent that there are many possible approaches to this problem : we note in passing a very different one (essentially a production-based system using a common intercommunications area) has been documented in the HEARSAY II project (Fennell and Lesser).

In order to effectively utilise the potentially very large processing power such an approach offers, we need to satisfy two criteria:

(i) A large amount of processor concurrency must be realised. We can achieve this by identifying and exploiting parallelism at both the high (or KM) level and at lower levels. So far as parallelism at the KM level is concerned, it is interesting to note the progressive trend towards the view that KM systems operate most naturally in a multiprocess environment. This is well exemplified by, for instance, HEARSAY, SPEECHLIST (Rovner et al '75), GUS (Bobrow et al '76), etc.

(ii) The relative amount of data flow between processors (vis à vis that within them) must be small. This requirement (we term it modularity) arises because of the inherent difficulty in contriving hardware mechanisms for interconnecting large numbers of elements and providing high data rates with low propagation times. This restriction plays a fundamental role in shaping the overall system design and it has led us towards the system described below.

We have chosen to make each (hardware) processor realise a high-level abstract machine. This machine (actually a multi-task LISP-like interpreter, described in detail in the following section) has characteristics and primitive operations especially matched to the KM problem domain. These include:

* the conventional list-manipulation primitives for building complex structures,
* a forking statement for programming explicit parallelism,
* primitives for explicitly allocating processing resources (either by rate or by quota) to facilitate open-ended breadth-first searches,
* a mechanism for saving an environment, and for running a program in a specified environment in a remote processor.

By adopting this approach, we have confined nearly all the data accesses by the underlying hardware processors to be local (i.e. fast) ones, that is, they take place between a particular processor and its local memory. Essentially, non-local (i.e. slow) accesses by a particular processor are only required when, at the abstract machine level, access is requested to a remote abstract machine. As a proportion of all the local (i.e. internal) operations involved in emulating the abstract machine itself, such access are relatively very infrequent.

We will now return to the topic of concurrency, and will identify some of the areas in KM where there is much scope for explicitly programmed parallelism. It is not appropriate in this paper to deal with these in any depth; instead we refer the reader to Bobrow and Winograd '76, and to Hanna '75 for extended discussions. We consider first data access. KM systems are characterised by very large data sets embodying complex interrelationships. In seeking to establish a particular hypothesis, or to retrieve a particular fact, or to disambiguate a referent, etc, an unstructured search through all the data is not

practicable : instead it is desirable to have preferential access to information that is more likely to be relevant. This can be achieved by a variety of methods such as foregrounding recently accessed data by using spreading activation semantic nets (Quillian '68), etc. In any of these approaches there is a substantial amount of work involved building up (and later deleting) these accessing structures and this work can well proceed in parallel with the main task in hand.

At perhaps a higher level, there is the organisation or factoring of knowledge into clusters (this may be identified with concept-formation). This, although a highly desirable attribute of a KM system, is such a computationally intensive task that it is not presently practicable to implement it. This sort of activity is, however, an ideal process to have running in a background mode on a multiprocessor system, "mopping-up" any under-utilised resources.

Yet another example of concurrency is provided by explicit user-scheduling of resource allocation. This may be used for focussing attention, by slowing down or killing-off lines of investigation which are proving unproductive (as for instance in a breadth-first search of a tree-organised structure).

§2 Design of the Abstract Machine

In the previous section we saw that each processor in the system was used to emulate a high-level abstract machine : we now describe the design of that machine. The two factors which most governed its structure were that it should allow as much scope as possible for parallel execution of programs, and that it should have as "clean" and compact a structure as could be achieved (so as to restrain the size of the emulator, which has to be implemented on every processor) commensurate with being able to work effectively on KM problems. This latter requirement suggested (as always!) a dialect of LISP as being an appropriate starting point, its uniform representation of both program and data being especially useful.

The design of the LISP machine is conventional in many respects : it executes programs written in an ALGOL-60 like procedural style. Unlike ALGOL however, the language is typeless: for instance, the value bound to a particular variable can be treated indifferently as either data or program. Where, however, it differs sharply from other LISP systems is in the way it exploits parallelism. This parallelism may be present either implicitly or explicitly in a program. As an example of the former, consider a function, for example $(F\ \boxed{\alpha}\ \boxed{\beta}\ldots\boxed{\omega})$. Conventionally, a LISP system would first evaluate argument α, then β, and so on. Only when all the arguments had been sequentially evaluated, would evaluation of the function then occur. In this system however all arguments are evaluated concurrently. Now obviously this gains nothing if the evaluation of each argument takes place entirely within that particular processor but, if some or all of the arguments are evaluated remotely in other processors (as described below), then there is a corresponding increase in execution speed.

Parallelism may also be scheduled explicitly by the programmer by using a conventional $(FORK\ \pi_1\ \pi_2\ \ldots\ \pi_n)$ construction, which will

cause the execution of each of the procedures π_1 to π_n to proceed in parallel. Control returns to the succeeding statement only after each of the procedures π_1 to π_n have run to completion: a primitive operator, (FRATRICIDE), is however provided to enable premature termination of siblings if desired. We note that the ability for parallel execution of programs (whether scheduled implicitly or explicitly) implies that the data environment associated with each abstract machine is modelled by a tree rather than just by a simple stack. Thus, at any one time we may have, in general, a whole set of simultaneously active environments. In order to be able to "save" these for subsequent use or to be able to access them remotely, there is a NAME statement by means of which a (globally accessible) name may be attached to any environment.

We now come to what is perhaps the central feature of the abstract machine, the mechanism whereby it can cause programs to be run remotely, that is, in other machines. This is effected by the RUN statement which has the following structure:

(RUN env proc arg-list resources control)

The purpose of a RUN statement is to cause the procedure specified by 'proc' to be run in the remote (previously NAMEd) environment 'env', having first had the value of the expressions specified in 'arg-list' bound to it. The arguments of RUN are first evaluated in the local environment and they are then transmitted to the remote environment. The value of 'proc' is then evaluated again, in this remote environment. This double evaluation is to allow the user freedom in locating the actual procedure body where he wishes. For instance, he can specify a procedure already defined in the remote environment, or a procedure defined in the local environment, or a procedure may be dynamically constructed and used remotely. Each of these mechanisms is useful in particular situations.

The last argument of the RUN statement, "control", may take only the values WAIT or CONTINUE. If its value is CONTINUE then the machine simply transmits the information necessary to establish the new process in the remote environment and carries straight on with the execution of the local process and there will be no further contact with the remote process. If alternatively, the value of the last argument is WAIT, then that process within the local machine establishes the new process in the remote environment and waits until this remote process executes a RETURN statement. We thus see that RUN provides very compactly a wide variety of facilities for interacting with environments in remote machines, for synchronising remote processes and for establishing programmes in an initially empty system.

§3 Implementation

In the above section we described the abstract machines as they appear to the user : we will now move down a level and describe some aspects of their implementation. As noted above they are all identical and are LISP based, and as such, are conventional in many respects. For the first version of this system we have chosen to use an entirely interpretive approach, with variables and their values being kept on association lists. Perhaps the most interesting aspect of the system is the very simple and yet powerful way in which parallelism is handled. This approach may be contrasted with the Spaghetti-stack system (Bobrow

and Wegbreit '73), and with Parallel Euler (Chirica et al).

Each interpreter has associated with it an activation-record space in which are held activation-records (hereafter, AR's). Each of these is a vector containing about a dozen items, mainly pointers to s-expressions and to others AR's. The organisation and function of these AR's may best be illustrated by means of a simple example. Suppose at some point in the execution of a program, the following functions needs to be evaluated:

(ADD (MINUS 12) (HD (P X)) (RUN ... WAIT))

When this function is encountered at run-time, the first operation to occur is the allocation of an AR with the following fields (amongst others) filled:

i. The name of the function, ADD, (in this case a primitive operation),

ii. A semaphore, initially set to zero,

iii. A pointer to the environment (an association-list) in which this function is to be evaluated,

iv. A pointer indicating where the result of the ADD operation should eventually be returned to,

v. A pointer to the parent AR. This AR represents a process which is held up pending this same result.

Next, the body of the ADD function, is entered (ADD being a primitive function, this body is a machine-code program). The first task of this is to cause the arguments (of which, for the ADD function, there may be an indefinite number) to be evaluated. In order to do this a set of AR's is created, one for each argument. These are each initialised in a similar way to that described above. In addition, the semaphore of ADD is set to 3. This semaphore will be decremented each time one of the processes associated with the three new AR's has run to completion. The scheduler, (which is responsible for causing the processes associated with AR's to run), only processes AR's whose semaphore field is zero. The net result of this is to cause all the arguments of a function to be scheduled for parallel evaluation, and then to suspend all further operations by that function until the results of evaluating all its arguments are available. We note that, in some cases, as in the above example, an AR may be awaiting the completion of execution of a program in a remote environment or processor.

In summary then, we have not the conventional pushdown stack of subroutine linkages, but a tree of AR's. At any point in the computation, the processors represented by the terminal AR's of the tree are all concurrently active (control passing from one to another, in indeterminate order, as each propagates further AR's) and those processes represented by non-terminals are suspended. Some branches of the tree spill over from one processor to another.

A specialised multi-microprocessor network on which this system is being implemented is currently nearing completion. It consists of a loosely-coupled set of Motorola M6800 8-bit microprocessors under overall control of a PDP-11/UNIX system. Each microprocessor has 24k bytes of memory, of which about one third is taken up by the multi-process LISP interpreter and low-level intercommunication procedures,

and the remainder is available for holding LISP cells and AR's. The system is described fully in Coghill '78.

§4 Conclusion

In this paper we have argued that typical KM processing tasks can be so structured that they may be efficiently implemented on a large loosely-coupled network of high-level, microprocessor-based machines, and the design of such a system has been outlined. Work is currently in progress to gather quantitative data to support the validity of this approach. A representative example of a KM system, a digital logic design aid with a natural-language user interface, is being programmed to run on the multi-processor system and preliminary results will be reported shortly.

References

BOBROW, D.G., KAPLAN, R.M., KAY, M. et al "GUS, A Frame-Driven Dialog System", Artificial Intelligence, Vol.8, 1977.

BOBROW, D.G., WEGBREIT, B., "A Model and Stack Implementation of Multiple Environments", Comms. ACM, Vol.16, No.10, October 1973.

BOBROW, D.G., WINOGRAD, T., "An Overview of KRL, a Knowledge Representation Language", J. Cognitive Science, December 1976.

CHIRICA, L.M., DREISBACH, T.A., MARTIN, D.F., PEETZ, J.G., "Two Parallel Euler Run Time Models", Conference Record of the Third ACM Symposium on Principles of Programming Languages.

COGHILL, G., "A Multi-microprocessor System for Knowledge Manipulation", Internal Report, Electronics Laboratory, University of Kent, 1978.

HANNA, F.K., "An Active Distributed System for Representing and Manipulating Real-World Knowledge", Internal Report, Electronics Laboratory, University of Kent, 1975.

QUILLIAN, M.R., "Semantic Memory". In M. Minsky (ed) "Semantic Information Processing", Cambridge, Mass., 1968.

ROVNER, P., NASH-WEBBER, B., WOODS, W.A., "Control Concepts in a Speech Understanding System", IEEE Trans. Acoustics, Speech, Sig. Processing, Vol.ASSP-23, No.1, February 1975.

WOODS, W.A., KAPLAN, R.M., NASH-WEBBER, B., "The Lunar Sciences Natural Language System : Final Report", B.B.N. Inc., Cambridge, Mass., 1972.

PROGRAM SYNTHESIS IN PREDICATE LOGIC

C. J. Hogger
Imperial College, London S.W.7, England.

Abstract

Research in automatic theorem proving has led to an interpreta-tion of predicate logic as a high-level procedural programming language. Logic 'programs' can be executed by interpreters based on the resolution principle. Logic offers a single formalism in which programs can be specified, derived and finally expressed using the same tool (theorem proving) as is used for executing them. This paper describes the derivation of logic programs from specifications written in standard logic.

Introduction

In its application to computer programming, predicate logic has been traditionally viewed as a specification language suitable for defining the relations computed by conventional programs. However, Kowalski (1974) has shown that logic can also be treated as a source programming language by giving a procedural interpretation to senten-ces in clausal form. Logic programs consisting of such sentences can be executed with reasonable efficiency by interpreters based on Robinson's (1965) resolution principle; the first practical logic interpreter ('Prolog') was developed by Colmerauer and others (1972). Recent developments in logic programming methodology and the implem-entation of a new interpreter at Imperial College are described in a report by Clark and Kowalski (1977).

A logic program describes relations but not the way they are investigated during execution. The resolution principle and a logic program determine a search space of resolution derivations (computat-ions) whose run-time exploration is governed by the interpreter's search strategy. In Kowalski's (1976) representation of an algorithm a logic program and a search strategy respectively determine the logic and control components of the algorithm. The logic component is problem-dependent and defines the algorithm's meaning in terms of the relations it computes. The control component is problem-independent and determines the algorithm's efficiency but not its meaning. A logic component can be viewed as a program in that it constitutes a complete and executable representation of the problem to be solved.

This formalism offers, in principle, alternative choices of logic and control to comprise a desired algorithm; these choices represent alternative programming styles. However, the limited strategies of existing interpreters generally require somewhat sophisticated logic in order to secure acceptable behaviour. As a result, non-trivial programs may not be self-evidently correct and so require reliable techniques for their verification. Techniques for verifying logic programs have been formulated by Clark and Tarnlund (1977) using first order deduction with induction principles

characterizing the program's data structures. Independent studies by Hogger (1977) and by Clark and Sickel (1977) have since shown that logic programs can be derived from specifications written in standard logic; such derivations may be pursued either as syntheses or as veri--fications according to their motivation. Clark and Sickel adapt the inferences developed by Burstall and Darlington (1977) for transform--ing recursion equations, but these are not the predominant features of non-trivial derivations; there the chief difficulties arise in the manipulations of arbitrary sentences of standard logic in order to admit occasional substitutions analogous to Darlington's 'fold' and 'unfold' rules. The present paper illustrates some inferences which impose a little uniformity upon these intermediate manipulations.

Logic Programs

A logic program is a set of Horn clauses of which one clause is a goal and the remainder are procedures. Execution attempts to show that the procedures imply the existence of solutions to the goal; the completeness of resolution determines that if solutions exist then they must be computable by a resolution interpreter. For example, the goal of the program below is to show that a minimum member u exists in the term $1:2:4:3:\emptyset$:-

goal: $\leftarrow min(u,1:2:4:3:\emptyset)$

procedures: $min(u,u:\emptyset)\leftarrow$
$min(u,v:w:x) \leftarrow v<w,\ min(u,v:x)$
$min(u,w:v:x) \leftarrow v<w,\ min(u,v:x)$.

A resolution interpreter computes the solution u := 1 to prove that these procedures imply $(\exists u)min(u,1:2:4:3:\emptyset)$. In this notation the symbol $\leftarrow$ means 'if', a comma means 'and' and all variables in the procedures are implicitly universally quantified.

Logic Specifications

Clausal form logic is generally unsuitable for expressing concise and complete definitions of relations and is therefore not a satisfactory specification language. However, relations can normally be defined quite lucidly using sentences of standard logic. A logic specification for a relation R is a set of such sentences which defines R and is intended to be self-evidently correct. In general these sentences take the form $R \leftrightarrow D_1, \ldots, D_n$ where the conjuncts in the definiens of R are arbitrary formulas. For example, the following sentences define the min relation computed by the program above :-

$min(u,x) \leftrightarrow m(u,x),\ (\forall v)(u \leqslant v \leftarrow m(v,x))$
$m(u,\emptyset) \leftrightarrow false$
$m(u,w:z) \leftrightarrow u=w \vee m(u,z)$.

Logic Program Synthesis

The objective of a synthesis for a relation R is a set of Horn clause procedures sufficient to solve a goal $\leftarrow R$. The initial resources consist of a set S of sentences comprising a specification

for R, and another set K (possibly empty) of sentences providing additional knowledge about the problem domain. A particular synth--esis consists of one or more derivations, each of which derives a procedure P for R satisfying $S,K \vdash P$. At present it is expected that both the choice of K and the ensuing derivations must be governed by the programmer's intelligence. However, the prospects for partial mechanization of the derivations (but not the choice of K) are quite good and may be envisaged in a context similar to that provided by Darlington's interactive implementation, but suitably extended to deal with our untidier syntax and inference rules. Nevertheless we advocate the plausibility of logic as a language in which programmers can specify, derive and express computer programs correctly even with--out automated aids.

Each logic procedure derivation presented here is organised as a sequence $(P_1, \ldots, P_r)$ of procedure-like sentences, where P_1 and P_r take the form :-

$$P_1 : \quad R \leftarrow D_1, \ldots, D_n \quad (D_1, \ldots, D_n \text{ arbitrary})$$
$$P_r : \quad R \leftarrow A_1, \ldots, A_m \quad (A_1, \ldots, A_m \text{ atomic}).$$

P_1 is trivially implied by the definition $R \leftrightarrow D_1, \ldots, D_n$ assumed in S and so the derivation certainly begins correctly; P_r is a derived Horn clause contributing to the eventual program. Sentences between P_1 and P_r (if any) are intermediate (in syntax) between standard logic and Horn clauses, and (in purpose) between declarative specifications and executable procedures.

Each derived sentence P_i is obtained from P_{i-1} by either of two distinct kinds of inference. The first kind (simplification) simplifies one or more conjuncts in P_{i-1} by, for instance, distribut--ing connectives or quantifiers, or by deleting trivially satisfiable conjuncts. Such inferences determine $P_{i-1} \vdash P_i$ independently of S and K. The second kind (substitution) combines P_{i-1} with some sentence implied by S,K such that $S,K,P_{i-1} \vdash P_i$. Suppose that P_i has the form $R \leftarrow D$, that F is any subformula of D and that a sentence $(F \leftrightarrow F')$ is implied by S,K. Let D' be the result of substituting F' for F in D. Then P_i is just $R \leftarrow D'$. Slightly more elaborately, suppose that a sentence $(F \leftrightarrow F') \leftarrow F''$ is implied by S,K and let D' be the result of substituting F' for F in D. Then P_i is just $R \leftarrow D', F''$. These substitutions are independent of the context of F in D. There are some useful variants of these cases which deal with substitutions of F' for F in D by exploiting weaker sentences like $(F \leftarrow F')$ and $(F' \leftarrow F)$ implied by S,K ; their applicability is usually dependent upon the context of F in D. We have not attempted to organise these inferences into a complete deduction system, but just emphasize them as the dominant features of our derivations.

Assuming that S and K are mutually consistent and that the inferences are sound, the derived procedure P_r is a true statement about R as defined by S, and can therefore only compute correct members of R. To confirm formally that a set of procedures is suff--icient for a goal $\leftarrow R$ requires the kind of case analysis used by Clark and Tarnlund in their termination proofs, though not necessarily retrospectively; it could be exploited instead during the exploration of the derivation tree to identify just those branches comprising a complete synthesis for R.

Two Examples

In the following examples it is assumed that the derived programs are intended for the simplest kind of Prolog-like interpreter which activates calls sequentially and selects clauses in order of presentation. The symbol ⊢ preceding a sentence means that the sentence is logically implied by S,K together with implicit axioms for primitives like =.

Example 1 : *Searching a list for duplicates*

The required program tests whether a list L with length N ⩾ 0 contains duplicates, in which event it satisfies a predicate duplic(x) whose specification S is :-

$$\text{duplic}(x) \leftrightarrow (\exists uij)(\text{item}(u,i,x),\text{item}(u,j,x),i\neq j)$$

where item(u,i,x) means that u is the i^{th} member of x. A naive way of solving ←duplic(L) uses the procedure trivially implied by S :-

⊢ duplic(x) ← item(u,i,x), item(u,j,x), i≠j.

Assume that calls to item select instances of i and j in the natural order 1,2,...,N . The first solution, if any, found by this algor-ithm (A1) gives the smallest distinct positions i := m and j := n at which some u occurs identically. A1 compares n+N(m-1) pairs of members to find this solution, of which ½(m)(m-1) are executed twice and a further m are just self-comparisons.

Avoidance of unnecessary comparisons requires knowledge of how duplicates may be distributed amongst the list's components ; so admit to K some basic facts about the structure of lists :-

$$\begin{aligned}
&(\exists w)(\text{length}(y,w)) && \text{(every list y has a length w)}\\
&\text{length}(y,w) \leftrightarrow (\forall i)(1\leqslant i\leqslant w \leftrightarrow (\exists u)\,\text{item}(u,i,y))\\
&\text{append*}(z,x',x) \leftrightarrow (\forall ui)(\text{item}(u,i,x) \leftrightarrow (u=z,i=1) \vee \text{item}(u,i-1,x'))
\end{aligned}$$

where append*(z,x',x) appends x' to the unit list (z) to give x. Then K implies a sentence (F↔F')←F" , namely :-

$$(\text{item}(u,i,x) \leftrightarrow (u=z,i=1) \vee \text{item}(u,i-1,x')) \leftarrow \text{append*}(z,x',x)\ .$$

This admits substitutions of F' for F in the duplic(x) procedure above in which F is identified with one of the item predicates :-

$$\begin{aligned}
\vdash\ &\text{duplic}(x) \leftarrow ((u=z,i=1) \vee \text{item}(u,i-1,x')),\\
&\qquad\qquad ((u=z,j=1) \vee \text{item}(u,j-1,x')),\ i\neq j,\ \text{append*}(z,x',x).
\end{aligned}$$

We now pursue two branches from this point. The first assumes the case where the first member z of x has duplicate occurrences; instantiate with u := z and i := 1 and simplify to give :-

$$\vdash\ \text{duplic}(x) \leftarrow ((j=1 \vee \text{item}(z,j-1,x')),\ 1\neq j,\ \text{append*}(z,x',x).$$

Now substitute for the subformula j=1 using another sentence (F↔F')←F" , namely (j=1 ↔ false)←j≠1 , and simplify the result :-

$$\vdash\ \text{duplic}(x) \leftarrow \text{item}(z,j-1,x'),\ 1\neq j,\ \text{append*}(z,x',x).$$

But the first two sentences of K (and properties of ⩽ and ≠) trivially imply a sentence (F ← F'), namely (1≠j ← item(z,j-1,x')) ; substitut-ing for the subformula 1≠j and merging the identical calls to item

then gives the first derived procedure for a new algorithm A2 :-

$\vdash$ duplic(x) $\leftarrow$ append*(z,x',x), item(z,j,x').

Backtracking to the branch point above, the second branch solves each conjunct (A $\vee$ B) by solving just B ; this deals with the case where the first member z has no duplicate occurrences in x :-

$\vdash$ duplic(x) $\leftarrow$ item(u,i-1,x'), item(u,j-1,x'), i$\neq$j, append*(z,x',x)
$\vdash$ duplic(x) $\leftarrow$ ($\exists$ui'j')(item(u,i',x'),item(u,j',x'),i'$\neq$j'), append*(z,x',x).

Denoting the first conjunct by F, S trivially implies the sentence (F $\leftrightarrow$ duplic(x')), so that substitution for F gives the second derived procedure for algorithm A2 :-

$\vdash$ duplic(x) $\leftarrow$ append*(z,x',x), duplic(x').

If the goal and the two new procedures are executed using the same control as A1, the resulting algorithm A2 compares only n+N(m-1)-½(m)(m-1)-m pairs of members. Each selected member u is compared by A2 with its successors only, whereas A1 also compares u with its predecessors and itself.

A more interesting algorithm (A3) is that which compares each selected member u with its predecessors only; to derive this we specify a new relation duplic* by admitting to K the sentence :-

duplic*(y,x) $\leftrightarrow$ duplic(x) $\vee$ ($\exists$uij)(item(u,i,y),item(u,j,x)).

The predicate duplic*(y,x) holds when the structure (y,x) contains duplicates but y does not. Now observe that K implies a sentence (F$\leftrightarrow$F')$\leftarrow$F" by suitably instantiating the definition of length(y,w) :-

$\vdash$ (false $\leftrightarrow$ ($\exists$u)item(u,i,y)) $\leftarrow$ length(y,0)

which admits a substitution into the definiens of duplic* to give :-

$\vdash$ (duplic*(y,x) $\leftrightarrow$ duplic(x) $\vee$ false) $\leftarrow$ length(y,0)
$\vdash$ duplic(x) $\leftarrow$ length(y,0), duplic*(y,x).

This is the first procedure for A3. Now exploit the append* definit-ion as before to derive two procedures solving calls to duplic*. One of these deals with the case where the first member of x also occurs in y, and its derivation should be obvious :-

$\vdash$ duplic*(y,x) $\leftarrow$ item(u,i,y), item(u,j,x)
$\vdash$ duplic*(y,x) $\leftarrow$ item(u,i,y),((u=z,j=1) $\vee$ item(u,j-1,x')), append*(z,x',x)
$\vdash$ duplic*(y,x) $\leftarrow$ append*(z,x',x), item(z,i,y).

The other procedure for duplic* deals with the case where the first member z of x does not occur in y yet has duplicate occurrences in the structure (y",x') which results from transferring z from x to y. The derivation is straightforward (if cumbersome) and gives :-

$\vdash$ duplic*(y,x) $\leftarrow$ append*(z,x',x), append*(z,y,y"), duplic*(y",x').

The last three derived procedures and Prolog-like control give algorithm A3 which compares (n-m)+½(n-1)(n-2) pairs and is therefore

independent of N. A list y (initially empty) records those members selected from x which are known to be distinct. Each call to duplic* selects a new member z from x and compares it with its distinct predecessors in y; a successful match solves the goal, but otherwise z is transferred from x to y and the search of x continues. In terms of comparison counts A3 is better than A2 if and only if we have $(n-m)(n-m-1) < 2(N+1-n)(m-1)$.

We can easily implement the calls to item, length and append* upon concrete representations of the lists. For example, choosing the conventional constructors . and nil, admit to K the definitions :-

item(u,i,nil) ↔ false
item(u,i+1,z.x') ↔ (u=z,i=0) ∨ item(u,i,x').

Trivial substitutions in the definiens' of length and append* give :-

⊢ length(nil,0)← and ⊢ append*(z,x',z.x')← .

Substituting these into the duplic and duplic* procedures of A3, for example, results in a set of macroprocessed procedures :-

⊢ duplic(x) ← duplic*(nil,x)
⊢ duplic*(y,z.x') ← item(z,i,y)
⊢ duplic*(y,z.x') ← duplic*(z.y,x')
⊢ item(z,1,z.x')←
⊢ item(u,i+1,z.x')← item(u,i,x')

sufficient to solve a goal like ←duplic(a.b.c.d.c.e.nil).

Example 2 : A selection-sort program

The desired program sorts a set x into an ordered list y by selecting the least member z from x to leave x', sorting x' into y' and appending y' to the unit list (z) to give y. A sufficient specification of the sort relation is as follows :-

sort(x,y) ↔ perm(x,y), ord(y)
perm(x,y) ↔ (∀u)(m(u,x)↔ m*(u,y)), (∀u)(m(u,x) ↔ once(u,y))
ord(y) ↔ (∀uvw)(u<v ← item(u,w,y), item(v,w+1,y))
once(u,y) ↔ (∃w)(item(u,w,y), (∀w')(w=w' ← item(u,w',y)))
m*(u,y) ↔ (∃w)item(u,w,y)

where m(u,x) means that u belongs to the set x, m*(u,y) means that u belongs to the list y and once(u,y) means that u occurs exactly once in list y. K just describes the construction of sets and lists and the meaning of selection :-

empty(x) ↔ (∀u)(m(u,x) ↔ false)
nil(y) ↔ (∀uw)(item(u,w,y) ↔ false)
union*(z,x',x) ↔ (∀u)(m(u,x) ↔ u=z ∨ m(u,x'))
append*(z,y',y) ↔ (∀uw)(item(u,w,y) ↔ (u=z,w=1) ∨ item(u,w-1,y'))
select(z,x',x) ↔ union*(z,x',x), (∀u)(z<u ← m(u,x'))

where union*(z,x',x) means that x is the set union of x' with {z} and select(z,x',x) means that x' results by deleting the least member z from x. Append*(z,y',y) has the same meaning as is given in *Example 1* and the meanings of the other predicates should be self-evident.

The presentation is simplified by deriving procedures for perm(x,y) and ord(y) separately and then conjoining them to give procedures for sort(x,y). Dealing with perm(x,y) first, we relate m(u,x) to m(u,x'), m*(u,y) to m*(u,y') and once(u,y) to once(u,y') in the reasonable expectation that perm(x,y) is solvable by manipulating the components u, x' and y' from which x and y are constructed. Firstly observe that K trivially implies :-

$\vdash$ (m(u,x) $\leftrightarrow$ u=z $\vee$ m(u,x')) $\leftarrow$ union*(z,x',x).

Secondly, substitute for item(u,w,y) in the definiens of m*(u,y) conditional upon append*(z,y',y) :-

$\vdash$ m*(u,y) $\leftrightarrow$ ($\exists$w)item(u,w,y)
$\vdash$ (m*(u,y) $\leftrightarrow$ ($\exists$w)((u=z,w=1) $\vee$ item(u,w-1,y'))) $\leftarrow$ append*(z,y',y)
$\vdash$ (m*(u,y) $\leftrightarrow$ u=z $\vee$ ($\exists$w)item(u,w-1,y')) $\leftarrow$ append*(z,y',y)
$\vdash$ (m*(u,y) $\leftrightarrow$ u=z $\vee$ m*(u,y')) $\leftarrow$ append*(z,y',y).

Thirdly, substitute similarly for the item predicates in the definiens of once(u,y) conditional upon (append*(z,y',y), $\sim$m*(z,y')) and simplify the result to give :-

$\vdash$ (once(u,y) $\leftrightarrow$ u=z $\vee$ once(u,y')) $\leftarrow$ append*(z,y',y), $\sim$m*(z,y').

Each of these three derived sentences has the form (F$\leftrightarrow$F')$\leftarrow$F" and admits an obvious substitution for some atom in the definiens of perm(x,y). They may be regarded as lemmas derived from K using the same basic ideas of alternate substitution and simplification. The perm(x,y) procedure is then derived as follows :-

$\vdash$ perm(x,y) $\leftarrow$ ($\forall$u)(u=z $\vee$ m(u,x') $\leftrightarrow$ u=z $\vee$ m*(u,y')),
($\forall$u)(u=z $\vee$ m(u,x') $\leftrightarrow$ u=z $\vee$ once(u,y')),
union*(z,x',x), append*(z,y',y), $\sim$m*(z,y')

$\vdash$ perm(x,y) $\leftarrow$ union*(z,x',x), perm(x',y'), $\sim$m*(z,y'),append*(z,y',y).

A derivation for ord(y) begins with substitutions for both item predicates in its definiens conditional upon append*(z,y',y). Note that in this - and in the above perm derivation - we are just investigating the effect upon the definiens implied by the assumption that the data structures are constructible as described by K. The procedure for ord(y) is a little tedious but quite straightforward :-

$\vdash$ ord(y) $\leftarrow$ ($\forall$uvw)(u<v $\leftarrow$ ((u=z,w=1) $\vee$ item(u,w-1,y')),
((v=z,w=0) $\vee$ item(v,w,y'))), append*(z,y',y)
$\vdash$ ord(y) $\leftarrow$ ($\forall$uvw)(u<v $\leftarrow$ item(u,w-1,y'), item(v,w,y')),
($\forall$v)(z<v $\leftarrow$ item(v,1,y')), append*(z,y',y)

(after some distributions and cancellations)

$\vdash$ ord(y) $\leftarrow$ ord(y'), ($\forall$v)(z<v $\leftarrow$ item(v,1,y')), append*(z,y',y)
$\vdash$ ord(y) $\leftarrow$ ord(y'), ($\forall$v)(z<v $\leftarrow$ m*(v,y')), append*(z,y',y)

(because if z is less than every member of y' then it is certainly less than any item with index 1 in y')

$\vdash$ ord(y) $\leftarrow$ ord(y'), ($\forall$v)(z<v $\leftarrow$ m(v,x')), append*(z,y',y), perm(x',y')

(using (m(v,x') $\leftrightarrow$ m*(v,y')) $\leftarrow$ perm(x',y') trivially implied by S)

$\vdash$ ord(y) $\leftarrow$ ord(y'), select(z,x',x), append*(z,y',y), perm(x',y').

Conjoining the derived perm and ord procedures and deleting the call union*(z,x',x) (since it is implied by the call select(z,x',x)) gives the desired recursive sort procedure :-

$\vdash$ sort(x,y) $\leftarrow$ select(z,x',x), sort(x',y'), append*(z,y',y), $\sim$m*(z,y').

But it is easy to show that S implies :-

$\vdash$ $\sim$m*(z,y') $\leftarrow$ $\sim$m(z,x'), perm(x',y')
$\vdash$ $\sim$m*(z,y') $\leftarrow$ select(z,x',x), sort(x',y')

so that the call $\sim$m*(z,y') in the sort procedure may be deleted to give the final version :-

$\vdash$ sort(x,y) $\leftarrow$ select(z,x',x), sort(x',y'), append*(z,y',y).

This procedure deals with the case where y is constructible by append*(z,y',y). The alternative case where y is the empty list is dealt with by the trivially derivable basis procedure for sort(x,y) :-

$\vdash$ sort(x,y) $\leftarrow$ empty(x), nil(y).

The calls to append*, empty and nil can be macroprocessed out by choosing terms as data structures as in *Example 1*. Alternatively, if more elaborate data structure representations are required, the sentences of K include sufficient specifications of append*, empty and nil to derive appropriate procedures for them. Taking the simplest option, choose . and nil to construct lists and choose : and $\emptyset$ to construct sets. Then the macroprocessed sort procedures are just :-

$\vdash$ sort($\emptyset$,nil) $\leftarrow$.
$\vdash$ sort(x,z.y') $\leftarrow$ select(z,x',x), sort(x',y').

S and K also provide enough information to derive sensible procedures solving the call to select(z,x',x) when supplemented by an assertion that < is transitive. Again, substitutions and simplific-ations of the kind illustrated here are sufficient to derive (quite easily) :-

$\vdash$ select(z,$\emptyset$,z:$\emptyset$) $\leftarrow$
$\vdash$ select(z,v:x',u:v:x") $\leftarrow$ u<v, select(z,x',u:x")
$\vdash$ select(z,v:x',v:u:x") $\leftarrow$ u<v, select(z,x',u:x").

Together these sort and select procedures behave reasonably well with an interpreter like Prolog. Given a goal $\leftarrow$sort(3:2:5:1:4:$\emptyset$,y), computation terminates successfully with the output binding y := 1.2.3.4.5.nil as required. Moreover, given a goal like $\leftarrow$sort(3:2:5:2:4:$\emptyset$,y), computation terminates unsuccessfully as required, since the input argument is not a representation of a set (due to the duplicate occurrences of '2'); computation fails because it cannot solve the eventual call 2<2. The desired property of the sort program - that it permutes a set into a list y having only distinct occurrences - is therefore satisfied.

Conclusion

Logic program derivations gain some uniformity from the inference rules illustrated here. Substitution of arbitrary subformulas

includes Darlington's 'fold', 'unfold' and 'law-application' inferences as special cases, but additionally allows much more general rearrangements of the knowledge expressed in S and K as well as facilitating its assimilation into the mainstream derivation of procedure-like statements.

In the examples presented here the final form of the derived procedures was known in advance, so that the derivations were really proofs of correctness which checked that the procedures computed the same relations as described by the sentences of S and K. The task of synthesis is obviously much harder because of the need to discriminate between alternative branches in the derivation tree in order to obtain a useable algorithm. Semi-automatic synthesis, in which the programmer is presented with a limited view of the derivation tree by an interactive program similar to Darlington's, appears to be feasible despite the greater syntactical difficulties offered by standard logic, but this remains a matter for future research.

Acknowledgement

I am grateful to both Robert Kowalski and John Darlington for their interest and encouragement in this work, and to the referees for their comments on the first draft. I have also been much influenced by the work of Keith Clark on the theory and methodology of logic programming.

References

1. Burstall, R. M. and Darlington, J. A transformation system for developing recursive programs. J.ACM 24, pp.44-67, 1977.

2. Clark, K. and Kowalski, R. Predicate logic as programming language. Research Report, Theory of Computing Research Group, Dept. of Computing and Control, Imperial College, 1977.

3. Clark, K. and Sickel, S. Predicate logic : a calculus for the formal derivation of programs. Proc. IJCAI-77, 1977.

4. Clark, K. and Tarnlund, S-A. A first order theory of data and programs. Proc. IFIP Congress, 1977.

5. Colmerauer, A., Kanoui, H., Pasero, R. and Roussel, P. Un systeme de communication homme-machine en francais. Rapport Preliminaire, Groupe de Researche en Intelligence Artificielle, Université d'Aix--Marseille, Luminy, 1972.

6. Hogger, C. J. Deductive synthesis of logic progams. Research Report, Theory of Computing Research Group, Dept. of Computing and Control, Imperial College, 1977.

7. Kowalski, R. Predicate logic as programming language. Proc. IFIP Congress, 1974.

8. Kowalski, R. Algorithm = logic + control. Research Report, Theory of Computing Research Group, Dept. of Computing and Control, Imperial College, 1976.

9. Robinson, J. A. A machine-oriented logic based on the resolution principle. J.ACM 12, pp. 23-41, 1965.

VISION

Berthold K. P. Horn
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139 U.S.A.

Underestimating the Difficulties

To us, vision is an immediate experience, not subject to careful introspection. We cannot write down protocols of processing steps that lie between the raw image intensities and our vivid impression of the surrounding scenery. Furthermore, we find ourselves in possession of this faculty long before we learn to master the more sequential processing tasks involved in the use of language, for example. Consequently, the difficulties of the vision process are often not appreciated. This is as true today, when many inroads have been made on the problem of understanding this process, as it was earlier when it was thought that vision could be understood simply in terms of some general ideas of artificial intelligence.

Divergence of Objectives

This difficulty is further compounded by the fragmentation of efforts resulting from widely varying motivations which bring researchers to this problem. These range from an intense desire to understand naturally ocurring vision systems to an interest in industrial application of machine vision. Somewhere in between we find those pursuing the information inherent in an image without regard to the implementation details of particular vision systems. It is not too surprising then that one finds widely diverging criteria for judging the importance of a particular piece of work.

Machine Vision is not the "I/O of A.I."

Still others use the vision domain only as a test bed to illustrate some general mechanisms currently favoured in other artificial intelligence work, or think of machine vision and manipulation merely as the "I/O of A.I.", the interfaces which allow the smart machine to interact intelligently with its environment. I contend that this is unreasonable since vision appears to have interesting features which do not have counterparts elsewhere, certainly not in the serial, linguistic kind of reasoning pursued in other areas of artifical intelligence. It is these features which make vision worthy of study in its own right. One such aspect which has not been approached seriously elsewhere is that of spatial reasoning which cannot be conveniently handled using the kinds of data structures explored so far and found so useful in other domains.

Task of a Vision System

What is the task to be tackled by a vision system? Despite widespread disagreement on many other aspects of vision, most would agree that a vision system is expected to produce a description of what is being viewed. The input may be one or more images, each a two-dimensional distribution of scene radiance values obtained from some sensing device. There is less agreement on the form of the output. What kind of description is acceptable? Clearly two criteria must be satisfied: The description must

(1) reflect some aspects of the three-dimensional reality, and
(2) be useful in carrying out a specified task.

Usually it is expected that the description take a symbolic form. Quite different kinds of descriptions are likely to be considered adequate when the system is part of a device which lines up integrated circuit chips for automated lead bonding [Horn, 1975b], as opposed to a stage of a system meant to express an opinion about the merits of a work of art. As is so common, the representation of the given information must be matched to the task at hand.

Task Independence

It would be much nicer if one common mode of description could be employed,

since the system dealing with visual inputs then could be designed in isolation, without considering the overall task. Perhaps this will turn out to be possible, at least for early stages of an image analysis system. Certain kinds of operations on images appear to be dictated by the image rather than the task and ought to be done without consideration for the task. At this point, however, it seems that task-dependent representations will be with us for a while.

Complexity of Machine Vision

An important lesson to be learned from the work on vision so far is that the problems are profound and not likely to succumb to the application of a bag of tricks from some other field, such as communications theory, statistics or linear systems theory. Machine vision merits its own methodology. Attention must be paid to both the physics of image formation as well as the information processing techniques which can produce the desired internal descriptions of what is being viewed. It now seems that this latter endeavour can be helped along by careful consideration of the human visual system and its strength and weaknesses.

Unfortunately biological vision systems are extremely complex and one can easily be led astray while studying them in isolation, without adequate tests of hypothesis one develops. Similarly, knowledge of certain physiological or chemical detail, for example, may not turn out to be very illuminating. What happens to the conformation of the rhodopsin molecule in the first few pico-seconds after a photon hits it is very exiting, but not helpful in the understanding of vision in a broader sense. One thing one learns quickly from even casual study of natural vision systems is that prodigous amounts of computation are involved in the processing of the image information.

A Peculiar Dichotomy

For a mobile biological entity above a certain size, vision is vital. It is hard to survive in a world where others have this faculty and use it in competition for food and in predator avoidance. Similarly, machine vision holds great promise for artificial systems. Many tasks cannot be done, or can be done only slowly or clumsily without it. So, vision, while difficult, is also very useful. As a result people will push the technology hard to get working systems. This has resulted in a peculiar dichotomy. There are two kinds of systems:

(1) systems which work at reasonable speeds, and
(2) systems which work reasonably well.

"Automated" Stereo

Illustrations of this curious phenomena abound. There are, for example, a variety of machines which extract topographic information at reasonable speeds from stereo pairs of aerial photographs. These devices use special purpose hardware to implement rather simple correlation techniques, and, as a result, require significant human assistance. First, the operator is obliged to help the system out of "trouble spots" where the correlation technique fails because either there is no detail, as on smooth sand or a lake, or because the two views are too different, perhaps because the slope is large. Many times the machine does not even note that it is in trouble and so records bad information. These "glitches" then have to be tediously removed in an interactive editing process if the data is to be at all useful.

On the other side of the coin, one finds many good ideas in the machine vision community which require sophisticated hardware and software for their implementation and which are slow on computers of todays ilk. Methods recently developed at Stanford [Quam 1971, Gennery 1977, Arnold 1978] and at M.I.T. [Marr 1974, Marr & Poggio 1976, Marr & Poggio 1977] are considerably more robust, but require staggering computing power on machines of standard architecture.

"Automatic" Terrain Classification

Systems have been developed for classification of terrain based on the application of pattern recognition techniques on a point by point basis. Special hardware has even been built to implement this simple process, perhaps prematurely, since the performance of this method leaves much to be desired. The classifier has to be trained anew for each image, it cannot deal with hilly terrain and changes in the lighting angles. Typically, the classifier is only used as a step in an iterative refinement process with the human operator making the real decisions. Even then many points are incorrectly classified if the separation between classes is not very distinct.

Yet at the same time work at Purdue [Landgrebe 1973, Landgrebe 1975, Gupta et al 1973, Swain 1973] and the University of British Columbia [Starr & Mackworth 1978],

has demonstrated the advantages of several methods for including contextual information. Growing small regions of similar spectral signature and classifying the regions, rather than individual points helps, as does the use of even rather primitive textural measures [Bajcsy 1973]. Amongst several other promising ideas are those recently expounded at the University of Maryland [Rosenfeld 1977b, Rosenfeld 1978] regarding the use of relaxation methods, also known as cooperative computation methods. While all of these methods produce results far superior to those generated by the point by point methods, it must be admitted that they require considerably more computing power.

Line-finding

As a last example we may look at edge-detection and line-finding. Many fast systems, some even running at full video speeds [Nudd 1978], use simple operations such as Robert's gradient, discrete approximations to the Laplacian operator or Sobel gradients. These produce visually pleasing result, but the edge fragments produced tend to be too noisy and ill-defined to succumb to concerted efforts to glue them into reasonably continuous lines and well-defined vertices.

Systems which do produce useable symbolic edge information such as those developed at M.I.T. [Griffith 1970, Horn 1971, Shirai 1975, Marr 1976, Marr 1978] require vast amounts of computing power both in terms of storage and machine cycles. Very similar sorts of things can be said about approaches which depend on scene segmentation using region growing techniques instead of edge-finding [Brice & Fennema 1970, Ohlander 1975, Tenenbaum & Barrow 1977].

Vision hardware

Part of the explanation for this dichotomy then lies in the impatience of the implementers and the real world need for solutions to pressing problems involving processing of visual information. It is natural to think in terms of special purpose hardware suited to particular algorithms. For specialized tasks it is possible to realize one to three orders of magnitude speed-up in processing with affordable special purpose devices. In the past the development of such hardware was perhaps inappropriate since no one had enough confidence in any particular scheme to commit resources to an implementation effort. Also, the existence of fast systems that use very

simple methods, has discouraged further work, since "the problem has been solved". A look at the results quickly convinces one that this is not so.

Roots

Several fields may be identified as having contributed major ideas to machine vision. I will single out just three here for discussion.

(1) Image Processing
(2) Pattern Recognition
(3) Scene Analysis

Each field has now matured sufficiently to have its basic tools documented in a number of monographs, collections and text books [Andrews 1970, Biberman 1973, Gonzalez & Wintz 1977, Huang 1975, Lipkin & Rosenfeld 1970, Rosenfeld 1969b, Rosenfeld 1976b], [Cheng 1968, Fu 1974, Fu 1976, Grasselli 1969, Tou & Gonzalez 1974, Watanbe 1969], [Duda & Hart 1973, Hanson & Riseman 1978, Winston 1969, Winston 1977], as well as hundreds of papers. Indeed, I cannot begin to do justice to these here, but instead refer the reader to A. Rosenfeld's excellent bibliographies issued annually [Rosenfeld 1969a, Rosenfeld 1972, Rosenfeld 1973, Rosenfeld 1975, Rosenfeld 1976a, Rosenfeld 1977a]. In order to see where we are and to discern possible future trends we should analyze the strength and weaknesses of each of these paradigms in tackling the basic task we have set out for a machine vision system.

Image Processing

Image processing, as the name suggests, is something one does with images. Herein lies both its strength and its weakness. Of the three fields mentioned, this is the only one which deals with images as input. Indeed the basic operations apply to arrays of raw image intensities. Many useful transfers of ideas to machine vision can be traced to this emphasis. Unfortunately, image processing also produces images as output -- not descriptions. Only in so far as these new, possibly enhanced, smoothed or sharpened images are easier to process are these techniques useful. Since, in the case of image processing, the final product is intended for human viewing, this is rarely the case. Further, one finds an unfortunate emphasis on linear, shift-invariant methods and consequently assorted transform techniques. Such ideas have only played a limited

role in machine vision.

Pattern Recognition

At the core of this field is a method, pattern classification, which is concerned neither with images nor with descriptions thereof. Pattern classification instead deals with the mapping of vectors into integers -- the vectors having components which represent measurements of some entity, the integers denoting the classes to which this entity might belong. This paradigm of feature extraction followed by pattern classification is of interest here, however, because many of its applications have involved features extracted from visual data. Several techniques used in the calculation of the numerical feature values for the classification process have found other applications in machine vision. Much of the sophisticated mathematical paraphenalia used to analyze the pattern classification stage has not.

Scene Analysis

Scene analysis concerns itself with the processing of descriptions of images into more sophisticated, or perhaps more useful, descriptions. In this category one finds much of the blocks-world work on line drawings [Roberts 1965, Clowes 1971, Huffman 1971, Waltz 1975]. As it turns out, obtaining the line drawings in the first place from the raw image information was the more difficult task; in fact, no system produces the perfect descriptions needed by early scene analysis systems [Winston, 1972, Grape 1972, Falk 1972].

More recently, discouraged by the complexity of the distributions of raw image intensities, researchers have turned to methods which exploit prior knowledge about the likely contents of the scene being viewed [Reddy et al 1973, Tenenbaum & Barrow 1976]. In Max Clowes' words: "Vision is controlled hallucination". The image contributes a small "controlling" influence on the vision system's "hallucinations" based on expectations and predictions. Similar ideas have taken hold in other areas such as speech, where researchers despair of dealing with the complexities of the raw acoustic waveform without guidance from various "knowledge sources". There is however an ever-present danger of "controlled hallucination" turning into "hallucination". I think we may have closed our eyes to the raw image for too long.

Things to avoid

The early years of any field tend to be characterized by a wide variety of approaches, many false starts and techniques based on inappropriate analogies. We can learn from these mistakes if we wish. Here are some things to avoid:

(1) Using a mechanism-oriented approach, instead of a problem-oriented one.

(2) Applying a known bag of tricks from another field.

(3) Believing that complexity will automatically give rise to interesting behaviour.

(4) Hoping that "learning" will provide a boot-strapping mechanism.

(5) Believing what works in a simple situation can be easily extended to a more complex one.

(6) Suffering from theorem-envy -- introducing unwarranted mathematical hair.

(7) Working only on the "interesting" sub-problem -- often not the weakest link.

(8) Following the latest fad. Create your own instead!

(9) Taking a random path through a maze of possibilities without explanation.

(10) Admiring the King's new clothes.

Current Trends

Attempts are being made to apply machine vision methods to so many different problems using so many different methods that it is impossible to give any kind of coherent summary. Furthermore, progress is being made in understanding several

important fundamental issues which cut across the spectrum of applications domains. It seems appropriate to concentrate attention to some of these issues.

Representation of Objects

If the task of the vision system is to produce useful descriptions of the scene being viewed, it is naturally important to pick a good representation for three-dimensional objects. If such a description is then to be used for recognition or in the determination of an object's position and orientation, it must capture information about the shape of the object and its disposition in space. This is an important problem, which does not occur in the processing of two-dimensional patterns such as microscopic image of bio-medical interest or in other areas such as finger-print identification or character recognition. A number of representations are currently being explored. One uses generalized cylinders or cones to approximate parts of objects after segmenting them into suitable pieces [Agin & Binford 1973, Nevatia 1974, Binford 1971a, Hollerbach 1976, Nevatia & Binford 1977, Marr & Nishihara 1977]. The information needed to construct such representations may be obtained by a variety of techniques including laser range finding [Nitzan et al 1977] and stereo disparity calculations.

SPINES

Another representation of the shape of an object uses surface normals or "spines". This was suggested [Horn 1977], as a more appropriate representation than one in terms of elevations above some reference plane [Horn 1975a], in part because surface normals undergo a simpler transformation under rotation. Indeed, human performance on shaded images suggests that we are rather poor at establishing relationships in elevation, but have a pretty good idea about the local surface orientation. Fortunately, methods for determining this kind of information exists, ranging from photometric stereo [Woodham 1977] to the shape from shading algorithm. More recently this representation has been suggested as a half-way step to representation in terms of generalized cones [Marr 1978].

Early Symbolic Description

From the discussion of the roots of machine vision it must be clear to the reader

that the crucial thing missing from all three ancestor fields is the lack of a method which takes one from raw image intensities to symbolic descriptions. Little thought had been given even to the problem of where the appropriate point for this transformation would be. Recent work suggests that the first symbolic description be obtained at an early stage [Marr 1976] of the processing of the visual information. That is, the initial symbolic description contains very many items, each of a rather simple nature. Further analysis is then carried out using symbolic information processing techniques on this initial data base.

This is a considerable departure from vision work in the blocks world, where the first real symbolic description was a complete line drawing. Even then it was clear that this was inappropriate, and crude symbolic description and the mechanisms for manipulating them, existed hidden in huge assembly language programs [Horn 1971].

Many of the ideas regarding the use of early symbolic descriptions have come from a better understanding of human vision. Conversely, computer implementations provide an outstanding way of testing emerging theories about visual perception [Marr 1978]. Without such checks speculation runs rampant.

Understanding Image Formation

It is not uncharacteristic of computer science to tackle a new domain with total disdain for the details of the mechanisms evident in that domain. Of more interest to the computer scientist are questions of computational structures and efficiency and whether a proposed algorithm will apply in the new domain. Machine vision is no exception in this regard. It seems that for a long time there was very little interest in the origins of the arrays of numbers given as input to a machine vision system. Recently it has been found that many constraints due to the physics of the real world situation can be succesfully exploited, once understood [Waltz 1975, Horn 1975a]. This enables processing not otherwise possible.

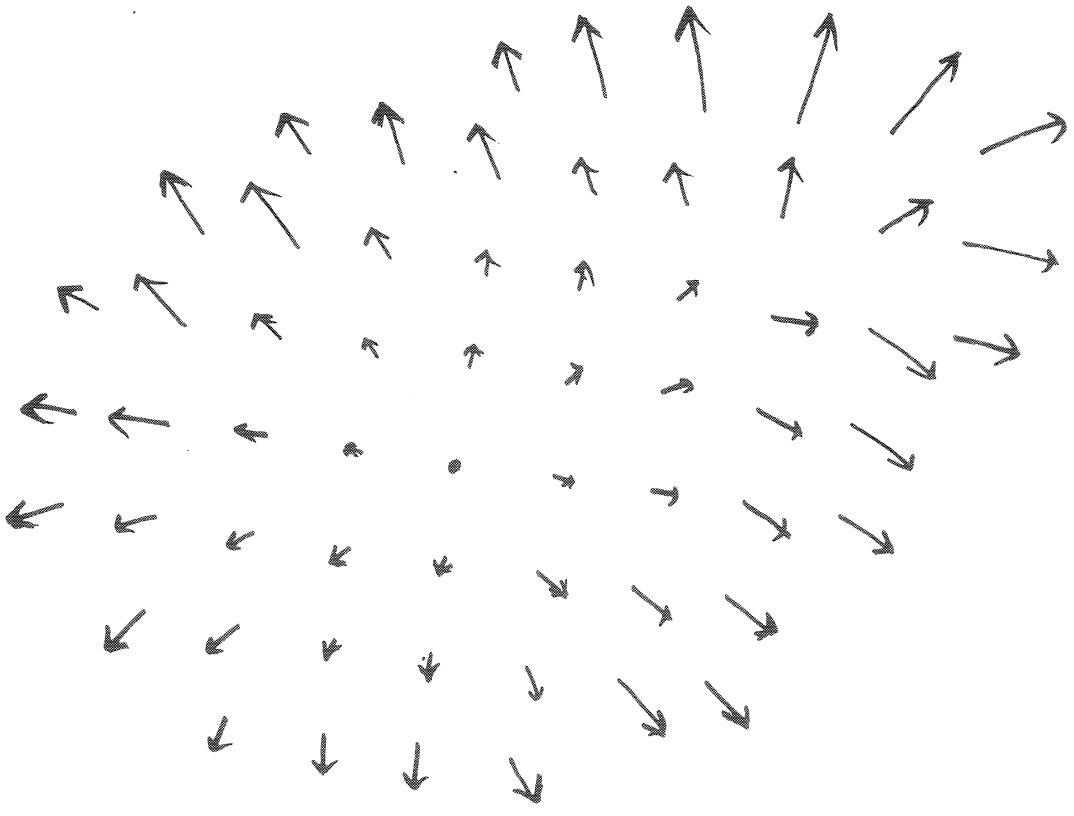
Understanding the process of image formation is helpful in inverting the image formation. That is, it is useful to know how objects are imaged, if one wishes to build a symbolic description of what is being viewed from the image. This kind of consideration has focused attention on smooth variations in intensities in an image [Horn 1977, Horn 1978]. Previously, image intensities were processed only to extract regions of more or less uniform properties or to locate points of more or less rapid

intensity change. At that point the image intensities themselves were discarded. This is unfortunate since a great deal of information about the objects being imaged is available there. This is quite different from the situation which applies in the case of binary images, useful in character recognition and printed circuit inspection for example.

Basically, what one is after is information about the permanent properties of the objects, such as reflectance, color and shape. This information is present in the raw image, but only in a coded fashion [Barrow & Tenenbaum 1978]. One may, for example, have to also deal with illumination conditions and shadowing. It is possible to extract all of this information from the raw image intensities, once the basic laws of image formation are understood. It is time to break the code.

Conclusion

Progress has been made -- at least we now know more about what we are up against. Much remains to be done. There is no shortage of good ideas right now, so we can discard some that no longer serve us well.



References

Agin, G. J. & Binford, T. O. (1973) "Computer Description of Curved Objects," Proc. 3rd Int. Joint Conf. on Art. Intell., Stanford University, Stanford.

Andrews, H. C. (1970) Computer Techniques in Image Processing, Academic Press, N.Y.

Arnold, R. D. (1978) "Local Context in Matching Edges for Stereo Vision," Proc. DARPA Image Understanding Workshop, May 1978, Baumann, L. (Ed), Science Applications, Inc.

Bajcsy, R. (1973) "Computer Identification of Visual Surfaces," Computer Graphics and Image Processing, Vol 2, No. 2, pp 118-130.

Barrow, H. G. & Tenebaum, J. M. (1978) "Recovering Intrinsic Scene Characteristics from Images," in Hanson, A. & Riseman, E. Computer Vision Systems.

Biberman, L. M. (Ed) (1973) Perception of Displayed Information, Plenum Press, N.Y.

Binford, T. O. (1971a) "Visual Perception by Computer," Proc. IFIP Conf., Dubrovnik, Yugoslavia.

Binford, T. O. (1971b) "Visual Perception by Computer," IEEE Conf. Systems and Control, Miami, Dec 1971.

Brice, C. & Fennema, C. (1970) "Scene analysis using regions," Artficial Intelligence, Vol 1, No. 3, pp 205-226.

Clowes, M. B. (1971) "On seeing things," Artficial Intelligence, Vol 2, No. 1, pp 79-112.

Cheng, G. C. et al (Eds) (1968) Pictorial Pattern Recognition, Thompson Book Co., Washington D. C.

Duda, R. O. & Hart, P. E. (1973) Pattern Classification and Scene Analysis, John Wiley & Sons, N.Y.

Falk, G. (1972) "Interpretation of imperfect line data as a three-dimensional scene," Artficial Intelligence, Vol 4, No. 2, pp 101-144

Fu, K. S. (1974) Syntactic Methods in Pattern Recognition, Academic Press, N.Y.

Fu, K. S. (Ed) (1976) "Digital Pattern Recognition, Springer, N.Y.

Gennery, D. B. (1977) "A Stereo Vision System for an Autonomous Vehicle," Proc. 5th Int. Joint Conf. Art. Intell., Cambridge, Mass.

Gonzalez, R. C. & Wintz, P. (1977) Digital Image Processing, Addison-Wesley, Reading, Mass.

Grape, G. R. (1973) "Model Based (Intermediate-Level) Computer Vision," Stanford Univ. A.I. Memo 201.

Grasselli, A. (Ed) (1969) Automatic Interpretation and Classification of Images, Academic Press, N.Y.

Griffith, A. K. (1970) Computer Recognition of Prismatic Solids, M.I.T. Project Mac, TR-73, M.I.T., Cambridge, Mass.

Gupta, J. N., Kettig, R. L., Landgrebe, D. A., & Wintz, P. A. (1973) "Machine boundary finding and sample classification of remotely sensed agricultural data," Machine Processing of Remotely Sensed Data, Purdue University, 4B25-4B35.

Hanson, A. & Riseman, E. (Eds) (1978) Computer Vision Systems, Academic Press, N.Y.

Hollerbach, J. (1976) "Hierarchical Shape Descrition of Objects by Selection and Modification Prototypes," M.I.T. A.I. T.R. 346.

Horn, B. K. P. (1971) "The Binford-Horn Line-Finder," M.I.T. A.I. Memo 285.

Horn, B. K. P. (1974) "Determining Lightness from an Image," Computer Graphics and Image Processing, Vol. 3, No. 1, pp 277-299.

Horn, B. K. P. (1975a) "Determining Shape from Shading," Chapter 8, in Winston, P. H. (Ed) The Psychology of Computer Vision

Horn, B. K. P. (1975b) "A Problem in Computer Vision: Orienting Silicon Integrated

Circuit Chips for Lead Bonding," Computer Graphics and Image Processing, Vol 4, No. 3, pp 294-303.

Horn, B. K. P. (1977) "Understanding Image Intensities," Artficial Intelligence, Vol. 21, No. 11, pp 201-231.

Horn, B. K. P. & Bachman, B. L. (1978) "Using Synthetic Images to Register Real Images with Surface Models," C.A.C.M. (in the press)

Hueckel, M. H. (1971) "An operator which locates edges in digital pictures," J. Assoc. Comp. Mach., Vol 18, pp 113-125.

Hueckel, M. H. (1973) "A local visual operator which recognizes edges and lines," J. Assoc. Comp. Mach., Vol 20, pp 634-647.

Huffman, D. A. (1971) "Impossible Objects as nonsense sentences," in Machine Intelligence, Vol 6, Metlzer B. & Michie, D. (Eds), Edinburgh University Press, Edinburgh, pp 295-323.

Huang, T. S. (Ed) (1975) Picture Processing and Digital Filtering, Springer, N.Y.

Landgrebe, D. A. (1973) "Machine Processing for Remotely Acquired Data," LARS Information Note 031573. Laboratory for Applications of Remote Sensing, Purdue Univ., Lafayette, Ind.

Landgrebe, D. A. (1975) "NASA Contract NAS9-1416 Final Report," Laboratory for Applications for Remote Sensing, Purdue Univ., Lafayette, Ind.

Lipkin, B. S. & Rosenfeld, A. (Eds) (1970) Picture Processing and Psychopictorics, Academic Press, N.Y.

Mackworth, A. K. (1973) "Interpreting pictures of polyhedral scenes," Artficial Intelligence, Vol 4, pp 121-138.

Marr, D. (1974), "A note on the computation of binocular disparity in a symbolic, low-level visual processor," M.I.T. A.I. Memo 327.

Marr, D. (1976), "Early processing of visual information," Phil. Trans. Roy. Soc. B

275, 483-524.

Marr, D. & Poggio, T. (1976), "Cooperative computation of stereo disparity," Science 194, 283-287.

Marr, D. & Poggio, T. (1977), "A Theory of Human Stereo Vision," M.I.T. A.I.Memo 451.

Marr, D. & Nishihara, H. K. (1977), "Representation and recognition of the spatial organization of three-dimensional shapes," Proc. Roy. Soc. B (in the press).

Marr, D. (1978), "Representing visual information," in Computer Vision Systems, Hanson, A. & Riseman, E. (Eds)

Nevatia, R. (1974) "Structured Descriptions of Complex Curved Surfaces and Visual Memmory," Stanford University, A.I. Memo 250.

Nevatia, R. (1976) "Depth Measurement by Motion Stereo," Computer Graphics and Image Processing, Vol 5, No 2., pp 203-214.

Nevatia, R. & Binford, T. O. (1977) "Description and Recognition of Curved Objects," Artficial Intelligence, Vol 8, No. 1, pp 77-98.

Nitzan, D. & Brain, A. E. & Duda, R. O. (1977) "The measurement and use of registered reflectance and range data in scene analysis," Proc. IEEE, Vol 65, No. 2, pp 206-220.

Nudd, G. R., Nygaard, P. A. & Thurmond, G. D. (1978) "Charge-Coupled Device Technology for Smart Sensors," Proc. DARPA Image Understanding Workshop, May 1978, Baumann, L. (Ed), Science Applications, Inc.

Ohlander, R. B. (1975) "Analysis of Natural Scenes," Carnegie-Mellon University, Department of Computer Science, Ph.D. Thesis.

Quam, L. H. (1971) "Computer Comparison of Pictures," Stanford University A.I. Memo 144.

Reddy, D. R., Erman, L. D. & Neely, R. B. (1973) "A model and a system for machine perception of speech," IEEE Trans. Audio and Electronics, AU-21, Vol 3., pp 229-238.

Roberts, L. G. (1965) "Machine Perception of Three-Dimensional Solids," in Optical and Electro-optical Information Processing, Tippet, J. T. et al (Eds), M.I.T. Press, Cambridge, Mass.

Rosenfeld, A. (1969a) "Picture Processing by Computer," Computing Surveys, Vol 1, pp 147-176.

Rosenfeld, A. (1969b) Picture Processing by Computer, Academic Press, N.Y.

Rosenfeld, A. (1972) "Picture processing: 1972," Computer Graphics and Image Processing, Vol 1, pp 394-416.

Rosenfeld, A. (1973) "Progress in picture processing: 1969-71," Computing Surveys, Vol 5.

Rosenfeld, A. (1975) "Picture Processing: 1974," Computer Graphics and Image Processing, Vol 4, No. 2, pp 133-155.

Rosenfeld, A., Hummel, R. A., & Zucker, S. W. (1976) "Scene Labelling by Relaxation Operations," IEEE Transactions on Systems, Man and Cybernetics, SMC-6, pp 420-433.

Rosenfeld, A. & Kak, A. C. (1976) Digital Picture Processing, Academic Press, N.Y.

Rosenfeld, A. (1976a) "Picture Processing: 1975," Computer Graphics and Image Processing, Vol 5, No. 2, pp 215-237.

Rosenfeld, A. (Ed) (1976b) Digital Picture Analysis, Springer, N.Y.

Rosenfeld, A. (1977a) "Picture Processing: 1976," Computer Graphics and Image Processing, Vol 6, No. 2, pp 157-183.

Rosenfeld, A. (1977b) "Iterative Methods in Image Analysis," Proc. IEEE Conf. on Pattern Recognition and Image Processing, June 1977, pp 14-18.

Rosenfeld, A. (1978) "Some Recent Results Using Relaxation-Like Processes," Proc. DARPA Image Understanding Workshop, May 1978, Baumann, L. (Ed), Science Applications, Inc.

Shirai, Y. (1975) "Analyzing Intensity Arrays using Knowledge about Scenes," Chapter 2,

in Winston, P. H. (Ed) The Psychology of Computer Vision.

Starr, D. W. & Mackworth, A. K. (1978) "Exploiting Spectral, Spatial and Semantic Constraints in the Segmentation of LANDSAT images," University of British Columbia, Computer Science T.R. 78-1.

Swain, P. H. (1973) "Pattern Recognition: A Basis of Remote Sensing Data Acquisition," LARS Information Note 111572. The Laboratory for Applications of Remote Sensing, Purdue Univ., Lafayette, Ind.

Tenenbaum, J. M. & Barrow, H. G. (1976) "IGS: a paradigm for integrating image segmentation and interpretation," in Pattern Recognition and Artficial Intelligence, Aacdemic Press, N.Y.

Tenenbaum J. M. & Barrow, H. G. (1977) "Experiments in interpretation-guided segmentation," Artficial Intelligence, Vol 8, No 3. pp 241-274.

Tou, J. T. & Gonzalez, R. C. (1974) Pattern Recognition Principles, Addision-Wesley, Reading, Mass.

Waltz, D. L. (1975) "Understanding Line Drawings of Scenes with Shadows," Chapter 2, in Winston, P. H. (Ed) The Psychology of Computer Vision.

Watanabe, S. (1969) Methodologies of Pattern Recognition, Academic Press, N. Y.

Winston, P. H. (1972) "The M.I.T. Robot," Machine Intelligence 7, Edinburgh University Press, Edinburgh.

Winston, P. H. (Ed) (1975) The Psychology of Computer Vision, McGraw-Hill, N.Y.

Winston, P. H. (1977) Artificial Intelligence, Chapter 8, Addison-Wesley, Reading, Mass.

Woodham, R. J. (1977) "A cooperative algorithm for determining surface orientation from a single view," Proc. 5th Int. Joint Conf. on Art. Intell., M.I.T., Cambridge, Mass.

Ani: An Example of Computational Creativity

Kenneth M. Kahn
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge Massachusetts 02139 USA

Abstract

A computer program named Ani is presented in this paper.[1] When completed Ani will be capable of producing non-representational narrative computer animation in response to very high-level incomplete descriptions. Ani brings together many different sources of knowledge in the form of *suggestions* which interact and are combined in complex ways. Initially Ani is faced with a very under-constrained situation. As choices are made constraints are added in such a way to keep the film both coherent and possible. This system is presented as one example of a computational process of creativity.

> Some of the programs we have described perform work that is considered difficult, and even mildly creative, when it is done by humans. Although these programs fall considerably short in performance of the highest levels of creativity of which humans are capable, there is every reason to suppose that they are qualitatively of the same genus as these more complex human problem solving processes.
>
> "The Processes of Creative Thinking" by Newell, Shaw and Simon [Newell 1962]

Introduction

Creativity is an adjective that is seldom applied to machines, except perhaps in the science fiction literature. Historically this has been justified because machines are rarely built with a creative capability which is based upon anything other than simulated flips of a coin. Recently computer programs have been built which exhibit a degree of creativity that, while nowhere near the level of a Beethoven, Shakespeare, Einstein or Picasso, is hard to deny. A program such as Talespin [Meehan 1976] can tell interesting original stories. That they are not as good as an Aesop fable is not critical; very few humans can make up stories of such quality either.

This paper presents a computer program called Ani which exemplifies the process of creating a complex structure from a high-level incomplete description. The complex structures that Ani will produce are simple narrative animations.[2] A user presents Ani with a script consisting of a description of the characters of a story, their relationship to each other, a general description of the cartoon's style, mood, focus, length etc., and a plot. A plot is a description of a series of scenes indicating the interactions and emotional states of the characters. The script which is presented to Ani in a very stylized format describes the film at a level such as "the heroine is shy, friendly, hard-working, and graceful", "the heroine is dominated by the antagonist who hates and mistreats her", "the film should be complex, fast-moving, and unusual", or "in the third scene the heroine should become very sad as a result of the antagonist doing something that prevents the heroine from meeting her lover".

---

1. Ani is a nickname for animation.
2. The use of the future tense for some of the description of Ani's behavior is expected to be replaced by the present tense this spring. The system is currently under development and is expected to behave as described herein by the end of May.

It is Ani's job to create a film starring simple geometric shapes fulfilling such a script. Ani is currently being tested on a short description of a simplified version of the story of Cinderella.

After receiving the script, Ani is faced with many choices upon which she must bring to bear different sources of knowledge. Most of this knowledge is represented as *suggestions* which recommend value ranges for descriptors such as size, typical speed, and shape. There are also suggestions for methods of establishing the personality and relationship of the characters, of conveying the interactions and emotional states of the characters, and of displaying the behavior of the characters.

The biggest problem that Ani faces is to make sense of all these suggestions from varied sources. First she combines those suggestions that are compatible with each other and notes any conflicts between incompatible suggestions. She proceeds to choose values for descriptors by making compromises between suggestions, by resolving conflicts and by gathering new suggestions. Ani will choose among the methods of establishing, conveying and displaying in such a way that those methods that are both appropriate and serve as many different purposes as possible are favored. In order to maximize the coherence and minimize the arbitrariness of any choice Ani views this as a simultaneous goal problem and makes those choices that are the easiest or most constrained first and only near the end will make the hard choices between many conflicting suggestions. Finally, she makes those choices for which she has the least basis.

A Script for Cinderella

The user presents Ani with a script which is a high-level description of the desired film. A script consists of descriptions of characters, the relationship between characters, the plot and global descriptors of the film. The plot is a description of the sequences of actions or scenes that are to take place. The complete script of a simplified version of the story of Cinderella that is presented to Ani is about two typed pages. Briefly the plot is as follows. Cinderella is good, kind and shy and dominated by her evil step mother. Her step mother prevents her from meeting the Prince. Cinderella's Fairy Godmother undoes whatever the step mother did, thereby enabling Cinderella to meet the Prince. They fall in love and are happy, while the step mother is unhappy. Presented here are portions of it to convey the level and incompleteness of scripts.

The entire description of Cinderella, for example, is

```
(define cinderella character ; Cinderella is defined to be a character
 (process initial description ; process and remember the following
  (physical (and beautiful shabby)) ; is beautiful and shabby
  (personality (and good friendly hard-working shy)); is good friendly ...
  (role-in-story most-important)))
```

All that Ani is told about the relationship of Cinderella and her step mother is that she is both tolerant of and obedient to her step mother who both dominates and hates Cinderella.

The first scene in the plot is described at a very high level thereby leaving Ani much leeway in determining the particular actions of the characters. She is asked to establish the personalities of Cinderella and her step mother, their relationship and their emotional state. The next scene in which Cinderella is kept from meeting the Prince is not described at such a high level, instead it is described at the level of conveying certain interactions between the characters. It is defined as

```
(define kept-apart scene
 (process initial description
  (sequence: ; the following parts are strictly ordered as presented
   (convey (wants cinderella (meets cinderella prince)))
   ; first show that Cinderella wants to meet the Prince
   (convey (prevents step-mother (meets cinderella prince)))
   ; next convey that her step mother prevents her from meeting the Prince
```

```
(and (establish cinderella (emotional-state (joy (negative high))))
     ; that Cinderella becomes very unhappy
     (establish step-mother ; and that her step mother is proud and happy
                (emotional-state (and (joy (positive medium))
                                      (pride (positive medium))))))))
(set your rhythm to (changes fast slow)) ; and its rhythm should lessen
(set your length to long)) ; this scene also should be long
```

One important point about the script is the use of words that are usually thought of as vague or squishy. Ani is perfectly at home with words like "hard working", "angry", "coherence" and "loves". These words have a direct effect upon the film Ani is producing. These words are primarily sources of suggestions that are tapped to make the myriad choices facing Ani. They are also used to generate character comparisons. Some of these words affect which methods of establishing the personalities or conveying the interactions and moods of the characters are chosen. Some methods are appropriate only if the actors involved have a particular relationship. For example, the step mother is able to use the method of chasing Cinderella off the stage because she "dominates" Cinderella.

A Cinderella Film

In the interests of simplicity Ani is limited to producing graphically very simple animation. The appearance of the characters is restricted to simple geometric shapes of a single color. The shapes, sizes and colors of the characters will be determined by Ani on the basis of the descriptions of the characters and their relationship, pair-wise comparisons of the characters that she generates, and the global description of the film.

One of the reasons for simplifying the graphic images of the characters is to avoid the necessity of placing in the program the tremendous amounts of real world knowledge needed to make representational images look and move in even a semi-realistic manner. The other motivation for keeping the graphics simple is to concentrate upon the dynamics of the animation. The complexity of Ani is in her determination of the movements and changes of appearance of the characters.

A typical scene is the one where Cinderella is prevented by her step mother from meeting the Prince. The script says that first the fact that Cinderella wants to meet the Prince should be conveyed, and then that she is prevented from doing so by her step mother. As a result of this she becomes very sad and her step mother is happy and gloating. The scene should be long and the rhythm should slow down. One of the simplier ways in which Ani can make such a scene begins with the Prince entering far from Cinderella. She then moves towards him only to be blocked by her step mother. She cannot get around her and the Prince continues past them without noticing them (i.e. making any special movement in their vicinity). Once the Prince has passed, Cinderella's movements become much slower, while her step mother trembles (with pride) and moves in fast long curves. Admittedly this deviates greatly from the original fairy tale, but the script is best viewed as inspiration to Ani rather than a detailed description of a desired film. The user can provide criticism, however, so that Ani can remake the film closer to his or her desires.

Suggestions

The reasoning Ani will perform to produce this film is complex. To choose a value, such as a typical speed for Cinderella, Ani begins by gathering suggestions from various sources, combining similar suggestions and noting any conflicts. If she finds some suggestions without any conflicts she has it easy and just follows the suggestions. If there are no suggestions she has to look harder for more suggestions. There are two additional types of sources of suggestions that she can tap: relative suggestions, which indicate the value of the element in question relative to the value of the same element of another character, and global descriptions of the film such as the desired level of variety, complexity, flashiness, etc.. When a conflict arises between suggestions Ani has several ways of resolving it. First she will try to gather more suggestions. Failing that she will reject one of the conflicting suggestions using various criteria, or try to make some compromise between the suggestions.

Character suggestion sources are primarily the descriptors of the character. The descriptors that define a character's personality, physical appearance, movement and role in

the story usually have associated with them suggestions for values. For example, the descriptor "hard working" is defined in Ani as,

```
(define hard-working personality-character-descriptor
 (process suggestions
   (for speed (not low) strength medium) ; typical speed is not low
   (for activity-level high strength high) ;; strongly suggested
   (for rhythmic-level (not low) strength medium)
   (for repetiveness high strength high)
   (for deliberateness high strength medium)
   (for (establish ?character hard-working)
       ; to establish that a character is hard working
        '(method: convey (does ,character task)
                  ; one method is to convey that the character does some task
                  best-when (difficult task))
                 ; this is best when the task is difficult
                  strength 'medium))
 (add industrious to list of neighbors) ; an indirect source of suggestions
 (add lazy to list of opposites)) ; consider the negation of lazy's suggestions
```

Relative suggestion sources are the descriptors of the relationship between the characters and comparisons between the characters. For example, the relationship descriptor "dominates" suggests that if character A dominates character B, then character A's typical speed should be greater than character B. Relative suggestions must interact with other choices to obtain an absolute value. Relative suggestions also come from comparisons that Ani makes. For example, the fact that two characters are very similar strongly suggests that they have the same values for many elements. Of course, this is too simple-minded -- sometimes what is crucial is the subtle differences between similar characters, or other times the difference between two similar characters should be accented by choosing divergent values for one or two elements to help the viewer distinguish the two characters. Similarly the finding that a character is unique suggests that the values of that character be different from the others.

Global suggestion sources are those descriptors that apply to the film as a whole. For example, when the energy level is high it suggests high values for the movement elements such as speed and acceleration. When the desired variety level is high it suggests values different from those chosen already. When all else fails then a value is obtained from the element itself (e.g. size or speed).

The problem of combining all these suggestions in order to actually choose a value is handled by Ani as follows

1) Combine those suggestions that are identical or nearly so

2) Combine those suggestions that can be combined without any information loss. E.g. the value "less-than high" and "medium" combine to "medium".

3) Note any conflicts between incompatible values

4) Compromise any suggestions whose value are "close enough", e.g. "medium" and "low".

To actually choose a value Ani analyzes the results from the combination step. If no conflicts are discovered and if there are enough strong suggestions then the value suggested is taken. If there are not enough suggestions, then Ani gathers up more information, if possible, and the process recurses. If more suggestions cannot be found because other choices have yet to be made, then those choices are given higher priority, the current choice is postponed, and the reason for its postponement is recorded. If conflicts were found then a long series of heuristics are used to try to resolve the conflict by either forcing a compromise or by rejecting one of the conflicting suggestions. Eventually a value is chosen. The decision as to which choice to work on next is based upon the idea that those choices that are the easiest to make, i.e. those that have the most non-conflicting suggestions should be made first, followed by those that are lacking essential (but obtainable) information, followed by those with conflicts, and finally those which have the least basis.

Generality and Conclusions

An important question one might have for a paper purportedly describing an example of computational creativity concerns its generality. Without more research, any answers are necessarily speculative, however, something can be said. I hope it is clear to the reader that the framework sketched above is appropriate for a broader class of animation. The system can be extended to include figurative images as well as more perceptual and less narrative animation.

The creation of an object often begins from a description of a set of parts, some relationships between the parts and a temporal description of the interactions of the parts. The usefulness of making comparisons between the parts to determine their degree of similarity and their level of uniqueness has a greater generality than animation. The framework in which suggestions from these various components interact and are combined is also of general applicability. The control structure guiding the creation of the animation was only touched upon in this paper. This control strategy is applicable to many domains where one is initially under-constrained and where the process of making choices adds new constraints. To know how far these ideas can be extended and to determine what other creative endeavors they apply, we will have to test them, i.e. to implement other examples of computational creativity.

> Stated otherwise, we would have a satisfactory theory of creative thought if we could design and build some mechanisms that could think creatively (exhibit behavior just like that of a human carrying on creative activity), and if we could state the general principles on which the mechanisms were built and operated. [Newell 1962]

I wish to thank Carl Hewitt, Henry Lieberman, Ira Goldstein, Luc Steels, Seymour Papert and Marvin Minsky for the help and advice they have provided me while doing this research. I would also like to thank IBM for providing me with a fellowship during the time of this research.

Bibliography

[Kahn 1977] Kahn, K.
"A Computational Theory of Animation", MIT AI Working Paper #145, April 1977

[Meehan 1976] Meehan, J.
"The Metanovel: Writing Stories by Computer", Yale University,
Department of Computer Science Research Report #74, September 1976

[Newell 1962] Newell, A., Shaw, J., and Simon, H.
"The Processes of Creative Thinking"
in Contemporary Approaches to Creative Thinking
ed. Gruber, H. Atherton Press, New York, 1962 pp. 63-119

[Weir 1975] Weir, S.
"The Perception of Motion: Action, Motives and Feelings" in Progress in Perception
Department of Artificial Intelligence Research Report No. 13
University of Edinburgh, December 1975

A SANE ALGORITHM FOR THE SYNTHESIS OF LISP FUNCTIONS FROM EXAMPLE PROBLEMS : THE "BOYER AND MOORE ALGORITHM".

Yves Kodratoff, Jean Fargues
G.R. 22 du CNRS, Institut de Programmation
4 Place Jussieu 75230 - PARIS CEDEX 05

Summary

We study the problem of synthesis of the algorithmic representation of a LISP function f, given its domain and a finite subset of its graph.
The solution is an interpretation of a program scheme. Instanciation of the scheme variables is made by a new method which uses subsumption and then, if the subsumption fails, uses generalization so that, at a further step, the subsumption of the generalized expressions succeed.

Key words

Difference equations. generalisation. LISP programs. Pattern matching. Program schemes. Program synthesis. Subsumption.

1. INTRODUCTION.

In a recent paper, P.D. SUMMERS [4] describes a methodology for the synthesis of LISP programs from example problems. These examples are given as a finite sequence $\{ x_i, y_i \}$, where $\{x_i\}$ is a finite subset of the domain of the function f which is to be found and where $y_i = f(x_i)$.

We have shown [2] that this methodology is actually a heuristic (called "insane" by SUMMERS) which instanciates the variables of a program scheme with relations found in the examples. In [2], we prove the validity of the choosen scheme for functions that increase like polynoms. We shall here consider in greater detail how to find these relations.
Briefly, it can be described as an attempt to subsume $f_i(x)$ to $f_{i+k}(x)$. If the matching fails, then we use the very way it failed to generalize $f_i(x)$ to $f_i(x_1,\ldots,x_n)$ so that a subsequent subsumption of $f_i(x_i,\ldots,x_n)$ to $f_{i+k}(x_i,\ldots,x_n)$ will succeed.
This recalls strongly the methodology used by BOYER and MOORE [1] in their celebrated paper on proving properties of LISP programs, and we propose to call our methodology : the BOYER and MOORE algorithm.

2. RESTRICTIONS ON THE DOMAIN AND THE RANGE OF f.

A detailed account on these restrictions is given in [2] : we shall recall them here briefly in a intuitive way.

2.1. Restrictions on $\{x_i\}$

2.1.1. Construction of the domain.

To put it in a wicked way, it is cheating to say that only a finite sequence $\{ x_i \}$ is given : we restrict ourselves to such simple domains that two examples are enough to define the whole domain.
We obtain x_{i+1} from x_i by substituting one atom At of x_i by a list L_i.
Further, we impose to L_i to be constant for each i and to contain a "signed" atom which will undergo the next substitution.

Examples of such domains are : {(A),(A B),(A B C),...} or {(A),(A B C), (A B C D E),...} or {(A),(A(B)),(A(B(C))),(A(B(C(D)))),...}.

As x_i undergoes only one substitution to lead to x_{i+1}, we call these functions : functions in one variable (they can be written as n-ary functions but depend on only one substitution).

2.1.2. The functions are defined on list-structures rather than on lists.
Let x = (A B C), LISP is well-suited to express x as a function of x by combining the two following rules.

1 - give to x its tree expression in CONS. Here,

x = (CONS A(CONS B(CONS C NIL))).

2 - give to each atom its functional expression, for instance, here B = (CADR x). Note that we write (CAR(CDR x)) as (CADR x), this convention will be used in the following. Here,

x = (CONS(CAR x)(CONS(CADR x)(CONS(CADDR x)(CDDDR x)))).

We shall say that each atom of x is then given as a fragment of x. Then we apply these two rules to $f(x_i)$: we write it as a tree of fragments of x_i, by which we find an explicit way to express : $f(x_i) = f_i(x)$.
For example, if x_3 = (A B C) and $f(x_3)$ =(C B A) then $f_3(x)$ = (CONS (CADDR x)(CONS(CADR x)(CONS(CAR x)NIL))).

It appears that we exclude all the functions that work upon the name of the atoms. For instance, a function given by x_1 = (A B), $f(x_1)$ = (A A B), x_2 = (A C), $f(x_2)$ = (A A A C) is excluded.
This means that we consider only functions that depend on the structure of the domain lists.

2.2. Restrictions on f.

Our method is the following : compare y_i and y_{i+k} so that you find relations between $f_i(x)$ and $f_{i+k}(x)$. Verify that these relations are the same for the examples given and make the induction hypothesis that these relations are true for each i. The reader will understand at once the analogy with differential equations except that we are on discrete domains, so it is more exact to say that we use difference equations. Then,our method reads : find a difference equation, make the induction hypothesis that the difference equation is invariant over the whole domain, try to solve the difference equation. Up to this point, we can state the following restriction of f : the difference equation between $f_i(x)$ and $f_{i+k}(x)$ are constant for each i.

3. SUBSUMPTION AND GENERALIZATION.

3.1. Matching two terms.

We say that $f_i(x)$ subsumes $f_{i+k}(x)$ when their functional expressions can be matched by means of two substitutions : 1 - (x/b(x)) where b is a composition of CAR and CDR, 2 - (NIL/h(x,NIL)) where h is any function (as matter of fact there are restrictions on h [2]).

Example : let us try to match (CONS(CAR x)NIL) with (CONS(CADR x)(CONS (CAR x)NIL)) : we get the substitutions (x/(CDR x)) and (NIL/(CONS(CAR x) NIL)). Subsumption can fail for several reasons, two of which we shall study.

The first one arises when x undergoes several different substitutions (it belongs then to contradictory classes).

Example 1. The matching of (CONS(CAR x)(CONS(CAR x)NIL)) and (CONS(CADR x)(CONS(CAR x)(CONS(CAR x)NIL))) leads to the substitutions : (x/(CDR x)),(x/x),and (NIL/(CONS(CAR x)NIL)). This is the type 1 contradiction.
We obtain the type 2 contradictions when NIL undergoes different substitutions.

Example 2. Consider the matching of (CONS(CONS(CAR x)NIL)(CONS(CAR x) NIL)) and (CONS(CONS(CAR x)(CONS(CADR x)(CONS(CAR x)NIL)))(CONS(CADR x) (CONS(CONS(CADR x)NIL)(CONS(CAR x)NIL)))). We obtain the following substitution(x/x), (NIL/(CONS(CADR x)(CONS(CAR X)NIL))) from the first half of these expressions, while the second half leads to : (x/(CDR x)),(NIL/(CONS(CONS(CADR x)NIL)(CONS(CAR x)NIL))) so that we get simultaneously contradictions of type 1 and 2.

3.2. Generalization for the type 1 contradiction.

We suppose that we are looking for relations on a sequence $\{f_i\}$, $1 \leqslant i \leqslant n$, working on the variable set V_o. We suppose further that each attempt of subsumption of f_i to f_{i+k} produces the same contradictions of type 1 and that we find p contradictions.

Then we build a new sequence f'_i which works on a set V of p variables. We link each of the variables of the sequence f'_i with one of the substitutions obtained before. Then, the matching of f'_1 and f'_{i+k} will clearly succeed.

Example.
We consider $\{x_i\}$ = {(A),(A B C),(A B C D E),...} $\{y_i\}$ = {(A A A), (B A C A), (C A E C A), (D A G E C A),... }

We can write :

$f_1(x)$ = (CONS(CAR x)(CONS(CAR x)(CONS(CAR x)NIL))),

$f_2(x)$ = (CONS(CADR x)(CONS(CAR x)(CONS(CADDR x)(CONS(CAR x)NIL)))),

$f_3(x)$ = (CONS(CADDR x)(CONS(CAR x)(CONS(CADDDDR x)(CONS(CADDR x)(CONS (CAR x)NIL))))),

$f_4(x)$ = (CONS(CADDDR x)(CONS(CAR x)(CONS(CADDDDDDR x) (CONS(CADDDDR x)(CONS(CADDR x)(CONS(CAR x)NIL)))))).

For each i we get the four different substitutions :
(x/(CDR x)),(x/x),(x/(CDDR x)),(NIL/(CONS(CAR x)NIL)).

Therefore, we build $f'_i(x_1,x_2,x_3,z)$ with

$f'_1(x_1,x_2,x_3,z)$ = (CONS(CAR x_1)(CONS(CAR x_2)(CONS(CAR x_3)z))),

$f'_1(x_1,x_2,x_3,z)$ = (CONS(CADR x_1)(CONS(CAR x_2)(CONS(CADDR x_3) (CONS(CAR x_3)z)))) and so forth.

Then the subsumption of $f'_i(x_1,x_2,x_3,z)$ to $f'_{i+1}(x_1,x_2,x_3,z)$ succeed for each i and we have :

$f'_{i+1}(x_1,x_2,x_3,z) = f'_i((CDR\ x_1),x_2,(CDDR\ x_3),(CONS(CAR\ x_3)z))$.

The use of the program scheme introduced in § 2 leads us to synthesize the program :

```
( DE  FEASY (X)  (F X X X X NIL))
( DE F (X1 X2 X3 Y Z) (COND
((ATOM (CDR Y)) (CONS(CAR X1)(CONS(CAR X2)(CONS(CAR X3) Z ))) )
(TRUE (F (CDR X1) X2 (CDDR X3) (CDDR Y) (CONS(CAR X3) Z  ))) ))
```

3.3.Generalization for the type 2 contradiction.

This generalization runs along the same lines than the preceding, an example will be the best way to explain how it works.
Example.
Let $\{x_i\}$ = {(A),(A B),(A B C),(A B C D),...} and $\{f(x_i)\}$ = {((A)A), ((A B A)B(B)A),((A B B C A)C(C C A)B(C)B),((A B C C B D A)D(D C B D A)C (D D A)C(D)C),...}.
We see that $f(x_1)$ and $f(x_2)$ were given under their tree functional expression (i.e. as $f_1(x)$ and $f_2(x)$) in example 2 of § 3.1. We shall match $f_2(x)$ and $f_3(x)$. For the sake of readability we index by the same number some of the CONS that are to be matched.

$f_2(x)$ = $(CONS_1(CONS(CAR\ x)(CONS(CADR\ x)(CONS_2(CAR\ x)NIL)))$
$(CONS_3(CADR\ x)(CONS(CONS_4(CADR\ x)NIL)(CONS_5(CAR\ x)NIL))))$.

$f_3(x)$ = $(CONS_1(CONS(CAR\ x)(CONS(CADR\ x)(CONS_2(CADR\ x)$
$(CONS(CADDR\ x)(CONS(CAR\ x)NIL)))))(CONS_3(CADDR\ x)$
$(CONS(CONS_4(CADDR\ x)(CONS(CADDR\ x)(CONS(CAR\ x)NIL)))$
$(CONS_5(CADR\ x)(CONS(CONS(CADDR\ x)NIL)(CONS(CADR\ x)NIL))))))$.

We obtain the substitutions :
from $CONS_1$: (x/x)
from $CONS_2$: (x/(CDR x)) and (NIL/(CONS(CADDR x)(CONS (CAR x)NIL))),
from $CONS_3$: (x/(CDR x)),
from $CONS_4$: (x/(CDR x)) and (NIL/(CONS(CADDR x)(CONS(CAR x)NIL)))
from $CONS_5$: (x/(CDR x)) an (NIL/(CONS(CONS(CADDR x)NIL)(CONS(CADR x)NIL))).

It is then clear that we need four variables x_1,x_2,z_1,z_2. For example, $f_2(x)$ becomes $f_2(x_1,x_2,z_1,z_2)$ = $(CONS(CONS(CAR\ x_1)(CONS(CADR\ x_1)(CONS$ $(CAR\ x_2)z_1)))(CONS(CADR\ x_2)(CONS(CONS(CADR\ x_2)z_1)(CONS(CAR\ x_2)z_2))))$.

For the sake of brevity we shall indicate where are these variables on the list expression, where the indexes of the atoms are those of the x, and where $)_i$ is the closing parenthesis of a NIL which has to be replaced by z_i :

$\{f_i(x_1,x_2,z_1,z_2)\}$ = $\{((A_1)_1A_2)_2,((A_1\ B_1\ A_2)_1B_2(B_2)_1A_2)_2,$
$((A_1\ B_1\ B_2\ C_1\ A_2)_1\ C_2(C_2\ C_1\ A_2)_1\ B_2(C_2)_1B_2),...\}$ as it is easy to see from the nature of the substitution each variable undergoes.
This means that we find the recursion relation :
$f_{i+1}(x_1,x_2,z_1,z_2) = f_i(x_1,(CDR\ x_2),\ g_i(x_1,x_2,z_1),\ h_i(x_2,z_1,z_2))$
As shown in [2], we iterate the process on h_i and g_i and generate the function :

```
(DE F2Z (X) (F X X X NIL NIL ))
```

```
( DE F ( X1 X2 Y Z1 Z2 ) ( COND
(( ATOM (CDR Y)) (CONS(CONS(CAR X1) Z1)(CONS(CAR X2) Z2)))
(TRUE (F X1  (CDR X2)(CDR Y)(G X1 X2 Y Z1)(H X2 Y Z1 Z2 ))) ))
( DE G (X1 X2 Y Z) (COND
 ((ATOM (CDDR Y)) (CONS (CADR X1) (CONS (CAR X2) Z)) )
(TRUE (G (CDR X1) X2 (CDR Y) Z))  ))
( DE H ( X Y Z1 Z2) (COND
(( ATOM (CDDR Y)) (CONS (CONS(CADR X ) Z1)( CONS(CAR X) Z2)))
(  TRUE ( H (CDR X) (CDR Y) Z1 Z2)) ))
```

Note that we do not believe that there exists any other system able to synthesize such a function from examples, though it is indeed very simple.

4. THE BOYER AND MOORE ALGORITHM.

At each iteration, the algorithm will work on a family $\{f_i^j\}$ of functions. Each f_i^j, for each i, has a set of variables named V^j (variables leading to a type 1 contradiction) and W^j (variables leading to a type 2 contradiction).
The sequence $\{f_i^j\}$ will be pointed at by a function-name pointer F^j. The F^j are set in a pile. At the initial state, the pile contains only F^o, pointer to the initial fragments sequence $\{f^o(X)\}$
Let F^j the current function-name which is at the top of the pile. We are going to try a subsumption of f^j_{i+k} by f^j_i

4.1. Terminal case 1

The matching is possible without any further iteration or new generalization. Suppose that n is the number of x-type variables and m the number of z-type variables in f^j_i. Then we have the relation:

$$f^j_{i+k}(x_1,x_2,\ldots,x_n,z_1,\ldots,z_m) =$$
$$A(x_1,\ldots,x_n,z_1,\ldots,z_{q-1},\ f^j_i(b_1(x_1),\ldots,b_n(x_n),\alpha_1(x_1,\ldots,x_n,z_1,\ldots,z_m),\ldots,$$
$$\alpha_m(x_1,\ldots,x_n,z_1,\ldots,z_m)),z_{q+1},\ldots,z_m).$$

where A and α_q are constant expressions obtained by composition of CONS, CAR and CDR and where b_p are constant compositions of CAR and CDR.
This solves the problem for F^j because this relation enables us to instanciate the variables of the scheme :

$$F^j(x_1,\ldots,x_n,y,z_1,\ldots,z_m) \Leftarrow$$

If $p_1(y)$ Then $f^j_1(x_1,\ldots,x_n,z_1,\ldots,z_m)$ Else...

If $p_k(y)$ then $f^j_k(x_1,\ldots,x_n,z_1,\ldots,z_m)$ Else

$$A(x_1,\ldots,x_n,z_1,\ldots,z_{q-1},\ F^j(b_1(x_1),\ldots,b_n(x_n),b_o(y),\alpha_1(x_1,\ldots,x_n,z_1,\ldots,z_m)$$
$$,\ldots,\ \alpha_m(x_1,\ldots,x_n,z_1,\ldots,z_m)),z_{q+1},\ldots,z_m).$$

4.2 Terminal case 2

The matching is possible by means of new generalization and the substitutions leading to a new z_q are constant for each i.
We shall not write down the explicit terms because they become really cumbersome because two cases can happen.

1. Else - we have to introduce new variables, then
   $f^j(x_1,\ldots,x_n,z_1,\ldots,z_m)$ has to be written as :
   $f^j(x_1,\ldots,x_n,x_{n+1},\ldots,z_1,\ldots,z_m,z_{m+1},\ldots)$.

2. Else - We suppress some variables introduced during the preceding iteration of the algorithm (for example, in F2Z, H is a function of x_2 only). Anyway, we use the same program scheme than in 4.1, with the appropriate number of variables.

4.3. General case.

In this case, for each i, we match f_i^j with a subtree of f_{i+k}^j. This induces one new problem to solve for the un-matched subtree with is at the left of the place f_i^j goes. Besides, this induces m new problems for each z_q, $1 \leqslant q \leqslant m$, that appears during the matching and such that the substitution $(z_j / h_i(x_1,\ldots,x_n,z_1,\ldots,z_m))$ depends on i. We write these substitutions as $(z_j / f_i^{(m+1)j+q}(x_1,\ldots,x_n,z_1,\ldots,z_m))$, $2 \leqslant q \leqslant (m+1)$.

The first induced sub-problem is given the name :

$F^{(m+1)j+1}$, and the others the name :

$F^{(m+1)j+q}$, $2 \leqslant q \leqslant m+1$.

We unpile F^j and apply the following scheme :

$F^j(x_1,\ldots,x_n,y,z_1,\ldots,z_m) \Leftarrow$

If $p_1(y)$ Then $f_1^j(x_1,\ldots x_n,z_1,\ldots,z_m)$ Else ...

If $p_k(y)$ Then $f_k^j(x_1,\ldots,x_n,z_1,..,z_m)$ Else

$F^{(m+1)j+1}(x_1,\ldots,x_n,y,z_1,\ldots,z_{q-1},F^j(b_1(x_1),\ldots,b_n(x_n),b_o(y)).$

$F^{(m+1)j+2}(x_1,\ldots,x_n,y,z_1,\ldots,z_m),\ldots,F^{(m+1)j+(m+1)}(x_1,\ldots,x_n,y,$

$z_1,\ldots,z_m)),\ z_{q+1},\ldots,z_m).$

Then $F^{(m+1)j+q}$, $1 \leqslant q \leqslant m+1$, are added to the pile.

It is clear that, as it is described, this algorithm stops after a finite number of iterations only if the initial sequence of examples increases like a polynom [2].

5. ABOUT A CONTER-EXAMPLE.

5.1. Description of the counter-example.

Let $\{x_i\}$ = (A),(A B),(A B C), (A B C D),...} and

$\{y_i\}$ = (A),(A B), (A B B),(A B B A B),(A B B A B A),...}

One way to obtain $\{y_i\}$ is to consider the limit word generated by the grammar $S \to A$, $A \to AB$, $B \to BA$ and to link an x_i of length k with the k first letters of this (infinite) word.

If we try to apply our method, we get at once :

$f_{i+1}(x,z) = f_i(x,h_i(x,z))$ and $\{h_i(x)\}$ = {(B),(B),(A),(B),...}

Now, we iterate the method on h_i and are unable to find a constant difference between $h_i(x)$ and $h_{i+k}(x)$ for any k. Our method is unable to solve this problem, but we urge the reader to use his human intuition (rather than ou method) and he will readily find that $h_i(x)$ is linked with $h_{2i}(x)$ and $h_{2i+1}(x)$: there is no constant interval of x, Δx on which we find a difference.

Let us then think again to our analogy with the diffrential equations. The fact that we cannot find a constant Δx on which we could find a difference is intuitively equivalent to the case of a function on R which is nowhere derivable. These functions indeed exist [3] but are truly far from any automatic prover or synthesizer !

5.2. A counter counter example.
Neverthless, if the above grammar is given with its full information at each step, then on the same domain, $\{y_i\}$ = {(A), (A B), (A B B A), (A B B A B A A B), (A B B A B A A B B A A B A B B A),... }

and a future implementation (not yet realized) of our system shall find the recursive program :

```
(DE FCOUSI(X)(F X X NIL))
(DE F(X Y Z)(COND
((ATOM(CDR Y))(CONS (CAR X)Z))
(TRUE(F X(CDR Y)(H X Y Z)))))
(DE H(X Y Z)(COND
((ATOM(CDDR Y))(CONS(CADR X)Z))
(TRUE(H X(CDR Y)(F X(CDDR Y)Z))))).
```

5.3. Remarks.

5.3.1. A synthesis system using this algorithm is completely implemented (except for the exponential increasing function as in § 5.2) in LISP 1.5.

5.3.2. The examples might seem too complicated. We could of course handle much simpler examples but they would not demonstrate the full power of the system.

5.3.3. This paper is a wide extension of SUMMERS' methodology [4]. The methods used by other authors in program synthesis from examples are entirely different (and, therefore, not cited here).

References.

[1] R.S. BOYER, J.S. MOORE, "Proving theorems about LISP functions", J. ACM. 22, (1975), 129-144

[2] Y. KODRATOFF, submitted to International Journal of Computer and Information Sciences.

[3] F. RIESZ, B. SZ.-NAGY, Leçons d'Analyse Fonctionnelle, Akadémia Kiado eds, (1953).

[4] P.D. SUMMERS, "A methodology for LISP program construction from example", J. ACM. 24,(1977), 161-175.

A STUDY IN THE APPLICATION OF THEOREM PROVING

David C. Luckham
Stanford University, Stanford, Ca., USA

Jorge J. Morales
Rutgers University, New Brunswick, NJ, USA

Joachim F. Schreiber
Technical University, Munich, BRD

Abstract

With selected examples we illustrate some useful applications of theorem proving, which have been considered during the last few years, and show, at the same time, the difficulties which have arisen. We aim at some kind of "hunch" language to guide in the generation of clauses. This paper is a short account of an unpublished manuscript by the same authors.

Descriptive terms: interactive theorem proving, choice strategy, edit strategy, answer extraction, application of theorem proving, exploratory use, proof checking, program verification, predicate logic programming.

INTRODUCTION

In order that this paper be self-contained, we present a short outline of the main features of PROVER, the Stanford interactive theorem prover. An earlier version is described in [AlLu 70].

PROVER is a resolution-based first order theorem proving program running on a PDP10. The program can be started in either of two modes, automatic or interactive. In automatic mode, PROVER uses some predefined

choice-and edit-strategies, which depend on the input. In interactive mode, the user is asked for choice-and edit-strategies. Choice-strategies specify which clauses are to be resolved or paramodulated with each other. Edit-strategies specify which clauses should be retained, and which should be deleted or simplified.

We now turn our attention to the heart of PROVER's interactive system, the control language.

PROVER has primitives for: (a)rules of inference (b)selection of strategies (c)information and display of results (d)selection of clauses (e) manipulation of the clause list.

(a)selction of rules of inference: The meanings of PARAMODULATE and RESOLVE are standard. PARAMODULATE is our equality substitution rule, and RESOLVE gives back all the resolvents of two clauses. There are furthermore comands SUBSTITUTE and SIMPLIFY. The former will substitute one term for another in a clause. The later one uses a list of equalities to simplify a given set of clauses.

(b)the rules of inference in (a) are used to build up the strategies. PROVER has several CHOICE-and EDIT-strategies. Built-in CHOICE-strategies are ANCESTRY, SUPPORT, VINE, UNIT P1, MODEL and DEFMODEL. MODEL expects an argument list which describes a binary partition of the predicate symbols, while DEFMODEL can be used to designate a LISP function to define a model. Built-in EDIT-strategies are DEMODULATION and all the clause selection functions described in (d). There is also a refinement of DEPTH called SELDEPTH which allows a restriction on the nesting of particular function symbols. More precisely the choice-, respectively the edit-strategy, can be a boolean expression of already described choice-, respectively edit-strategies. TRYTIL is an extension of TRY, i.e. it has an additional argument, a LISP function which defines the termination criterion of the proof search.

(c)Whenever the user wishes, he can stop PROVER and examine and select clauses. FLOATUP and FLOATDOWN run through the clause list and display each clause. UP n and DOWN n move the clause pointer up or down by n

clauses respectively. GO n moves the pointer directely to clause n. There is also a command which displays the entire deduction tree of a clause: ANCESTRY[clause]. Once the (sub-) proof has been found, the user has the possibility of obtaining information about the particular term-substitutions in the proof by using the answer extraction facility. The answer extractor is based on [LuNi 71].

(d)The basic command for the selection of clauses is "FIND set of clauses; pattern" where pattern is a boolean expression consisting of the following primitive patterns: TREE(X) matches all clauses which contain clause X in its deduction tree, predicates length and depth, OCR[ ] has as argument a term or a literal and matches any clause which contains that term or literal.

(e) The user can also change the clause list. Primitives for the manipulation of the clause list are commands for adding and deleting clauses to/from the clause list: ADD n and DELETE n add/delete a clause with index n to/from the clause list.

New features added to PROVER are a natural mathematical input/output notation and an extended command language. The extended command language permits better interaction between the user and PROVER. Further refinements are ordered resolution, a form of ordered paramodulation and a special form of locking resolution [Sc 75].

More details appear in later sections in connection with applications.

The applications of PROVER during the past five years can be naturally divided into two main areas:

1. Mathematical applications
2. Computer science applications.

In turn the mathematical applications can be subdivided as follows:

1.1. exploratory use of PROVER in finding proofs of recently reported theorems (without proofs) in group theory, ternary boolean algebras, and an open problem in Tarski geometries.

1.2. checking of proofs reported in the literature.

The computer science applications were also twofold:

2.1. finding first order proofs of verification conditions of programs.

2.2. predicate logic programming - an evaluation.

1.1 EXPLORATORY USE OF PROVER

Here we describe attempts to use our theorem proving program as a research tool for the mathematician. We believe that the computer can be a useful tool in the process of finding new results in abstract mathematics. Hence, we wanted to see how we would fare on mathematical problems of current interest to research workers who were unaided by a computing machine. The criterion of usefulness was twofold.

1. could we obtain the same or better results?
2. given that we obtained at least the same results, were there any advantages in using a computer?
   i.e. were the results obtained faster, cheaper?

The first experiment was undertaken during the summer of 1972. To show how mathematical research could be expedited by the user of a theorem prover, we scanned recent mathematical literature for problems we thought could be attempted by our program. In the Notices of the American Mathematical Society, we found the following announcement without proof [Co 72]:

Some new axiomatizations of group theory, a preliminary report:

We give axiomatizations of group theory and abelian group theory in terms of the binary group operation $f(a,b)=ab'$. Suppose G is a first order theory with equality and with a binary operation symbol f, whose non-logical axioms are: (G1) $f(x,x)=f(y,y)$, (G2) $f(x,f(y,y))=x$, (G3) $f(f(x,z), f(y,z))=f(x,y)$. Then G is an axiomatization of group theory [1 is defined as the (unique) element $f(x,x)$ and xy as $f(x,f(1,y))$]. For abelian groups, G1, G2, G3 are replaced by: (A1) $f(x,f(y,z))=f(z,f(y,x))$. (A2) $f(x,f(x,y))=y$, or by the single axiom (A) $f(f(x,y),f(f(x,z),y))=z$. Some other single axioms for abelian groups are (A') $f(f(x,f(y,z)),f(x,y))=z$ and (A'') $f(x,f(y,f(z,f(x,y))))=z$.

Proof of this announcement and an extension of the results, namely, a new single axiom characterizing abelian groups

A3: f(x,f(f(y,z),f(y,x)))=z

were found with the interactive aid of PROVER in about a month. Subsequently, we were able to send a copy of our report on this experiment to the authors of the announcement, complete with machine generated proofs of their announced results, before they themselves wrote them up (private correspondence with R.H. Cowen).

Computing times varied from instantaneous, for easy proofs, to 45 seconds for the hardest. The problem of finding proofs was left entirely to the program, and the correctness of the results was not known to the user in advance. The role of the user was to direct the process of checking the announced theorems by (a) formulating reasonable problems and then presenting them to the machine for solution, and (b) monitoring the process (e.g. the program sometimes solved another problem in the course of searching for a solution to a given one.)

Essentially the simplest possible method of using PROVER was employed: assumptions and theorem to be proved were input to the program from an on-line-console; a very elementary selection of efficiency strategies was employed, in most cases merely set-of-support and some reasonable bounds on the complexity of deductions (usually called clauses) that were retained in memory. The only real interactive help given by the user in proving any of the results was the choice of a set of previous results or "lemmas" to be used in simplifying deductions (demodulation list).

The program keeps only the simplified form of its deductions. For example, suppose a deduction such as (1) f(y,f(x,x))=y occurred during a proof search, and (2) f(x,x)=1 was already derived. It is easy to see that (1) reduces to (3) f(y,1)=y under simplifying substitution using (2), so that if the user felt that no useful deductions were likely to be generated by using substitution instances of (3) such as (1), he placed (2) on the simplification list. The program could then simplify (1) down to (3) and retain only (3) in memory.

The problem sequence

We now give a brief account of the sequence of problems fed to the program in order to establish these results. Files containing the axioms given in the announcement were created as follows (e.g. file AB1 contains axioms A1 and A2):

G={G1, G2, G3}
AB1={A1, A2}
AB2={A}
AB3={A'}
AB4={A''}.

Since by definition in the announcement, f(x,x)=1, this statement was added as a hypothesis when proving most of the results. The problems given to the program fall naturally into four sets as follows.

(1) The program was used to find proofs that G implies the axioms of group theory
   (i) the group operation is associativ.
   (ii) there is an element u ∈ G, with ua=a=au for all a ∈ G.
   (iii) for this element u, there is to each element a ∈ G, an elemetn a' ∈ G with aa' =u.

(2) The program was used to show that AB1 proves that the group operation f(x,f(1,y)) is commutative.
The program then established that AB1 proves G1, AB1 + G1 proves G2 and AB1 + G1 + G2 proves G3. From this sequence of proofs we may conclude that all models of AB1 are abelian groups.

(3) Continuing, we were able to show automatically that each of AB2, AB3,and AB4 proves AB1, thereby checking that their models are also abelian groups. Also, in the other direction, very simple proofs that AB1, AB2, AB3 and AB4 are interderivable were generated automatically.

(4) Then we used the program to prove that the axioms of group theory imply G and that those of abelian group theory imply AB1, AB2, AB3 and AB4, thereby demonstrating that G and AB1-4 characterize group theory and abelian group theory respectively.

Finally, the "new" single axiom characterizing abelian group theory:

X1:f(x,f(f(y,z),f(y,x)))=z

was one of the deductions generated by the program while trying to show that the axioms of abelian group theory (i,ii,iii above plus an axiom stating the commutativity of the group operation) proved AB2. Therefore, we only needed to check whether X1 implied any of AB1 through AB4, in order to demonstrate that it also was an abelian characterization.

The strategy used to select possible new candidates for abelian group characterizations consisted in selecting those clauses generated by PROVER that had three or more different variables on the left side of the equation and a single variable on the right side [Mo 73].

Encouraged by this experiment we decided to tackle a harder problem: pushing the program to find the answer to an open mathematical problem in subtheories of Tarski Geometry [Tar 51]. We were unable to solve the problem, but did derive interesting "partial solutions".

Further interesting experiments, results and open problems in this area can be found in [LMS 76].

1.2 PROOF CHECKING

Another important application of theorem proving is the checking of mathematical proofs wherever they come from - automatic generated proofs from other theorem provers; hand simulated proofs as they are often used in papers about theorem proving; proofs in mathematical papers, which sometimes forget to state used lemmas explicitly, etc.

In the following we will show one example and the features of PROVER which were used to verify it.

The first example we verified was a hand simulated proof by Henschen and Wos [HeWo 74], of the theorem: If a subgroup H has index 2, then it is normal. 4 subproofs were performed, the longest of which is 17 binary resolutions long.

The original set of clauses, not being a Horn set is renamed and split. After that several Horn subsets are obtained. All of these must be proven inconsistent to show that the original set of clauses is inconsistent.

For example: if the original unsatisfiable set is S U P(X,Y)vQ(X,Z) then upon splitting it the following two subsets result: S U P(X,Y) and S' U Q(X,Z). Of course there are also other possible splittings. Both subsets are now inconsistent and care must be taken that the common variables, in our example X, have same substitutions in both refutations. These are then refuted using a special strategy for Horn sets: positive unit resolution. After repeated splitting of the original set, we obtain 6 subproblems. The proofs are easy to obtain. SUBSTITUTE turned out to be very useful, a SPLIT command would be needed.

2.1 PROGRAM VERIFICATION

The first computer science application of PROVER was in the automatic verification of programs. Within the theoretical framework of a Hoare-type logic for a subset of the programming language Pascal [Hoare 72], [ILL 75], we were able to prove properties of many simple programs, including the ordered correctness of a large set of sorting algorithms.

The software basis of the verification project supported by Stanford's Artificial Intelligence Laboratory consists of VERIFY, an automatic verification condition generator and simplifier for PASCAL [ILL 75], [Suz 75]. The verification process began by submitting a PASCAL program, say INTERCHANGE-sort [King 69], with inductive assertions to VERIFY. This program would then generate a set of first order verification conditions.

There are atleast two ways of using PROVER for program verification. The first, and harder, is to feed the verification conditions immedeately to a theorem prover. The other way of finding computer assisted proofs of verification conditions has been explored by [HeLu 75] and [LuSuz76]. By using a subgoaling mechanism and a judicious choice of lemmas, they

are able to find faster proofs than by using PROVER. However, PROVER has been most valuable in demonstrating that the lemmas can be derived from a standard first order axiomatization.

In the following we illustrate the first way.

After some preprocessing, these verification conditions would in turn be fed to PROVER together with a set of axioms that characterized the ordered properties of the underlying mathematical structure.

The verifier takes as input a Pascal program, in our case this being the sorting routine INTERCHANGE, taken from [King 69], together with the inductive assertions. This program sorts by successively finding the largest element of an array A.

The output of the verification condition generator was 6 verification conditions. As an example we give the third one:

$$\begin{aligned}&VC3:(\forall K((J+1\le K)\wedge(K\le N-1)\supset A[K]\le A[K+1])\wedge\forall L((1\le L)\wedge(L\le 1-1)\wedge(1-1\le N)\supset A[L]\le BIG)\wedge\\&\forall M((1\le M)\wedge(M\le J)\wedge(J\le N-1)\supset A[M]\le A[J+1])\wedge BIG=A[LOC]\wedge\\&\quad 1\le LOC\wedge LOC\le J\wedge I\ge 2\wedge 2\le J\wedge J\le N\wedge I-1\le N)\wedge(I\le J)\rightarrow A[I]>BIG\\&\rightarrow\forall K((J+1\le K)\wedge(K\le N-1)\supset A[K]\le A[K+1])\wedge\forall L((1\le L)\wedge(L\le I-1)\wedge(I+1-1\le N)\supset A[L]\le A[I])\\&\quad\wedge\forall M((1\le M)\wedge(M\le J)\wedge(J\le N-1)\supset A[M]\le A[J+1])\wedge A[I]=A[I]\wedge 1\le I\wedge I\le J\wedge I+1\ge 2\wedge 2\le J\wedge J\le N\\&\quad\wedge I+1-1\le N\end{aligned}$$

Each of the verification conditions underwent propositional simplification before being used as input for PROVER.

Most but not all of the proof was found in automatic mode. For lack of space we do not give the details and proofs, but refer the interested reader to [LMS 76].

Discussion

The main conclusion obtained from these and other experiments in [LMS 76] was the ability to derive the ordered correctness of a collection of sor-

ting algorithms written in Pascal from a relatively small number of arithmetical axioms. We believe that a uniform theorem prover is very useful for the formal verification of simple lemmas employed in proving the original verification condition.

2.2 PREDICATE LOGIC PROGRAMMING

As has been pointed out by different authors, the equivalence of deduction and computation has been known to logicians for a long time. See for example [Kl 52].

In the last couple of years several papers on predicate logic programming have appeared in the literature. See for example [Bi 75], [BFS 77], [Col 72], [Hayes 73], [Ko 74a], [Ko 74b], [Mi 75], [Ta 75a], [Ta 75b].

A theorem prover can be used in two ways: Given a description of the problem it can be used to help establish an "equivalent and better description" of the problem. This "equivalent and better description" can either be transformed into a program in, e.g. Pascal, or, given input data, it can be directly executed by a theorem prover (this being the second way).

In this paper we are only concerned with the second way, i.e. using a theorem prover as an interpreter. In [LMS 76] we tried PROVER on several programs and analysed its behaviour.

Results with PROVER have shown that our current "hunch" language is too weak to definitely improve the proof search in the area of predicate logic programming. In our opinion control structures of a programming language might be useful as elements of a hunch language for PROVER.

Because we do not have any theoretical result (even for practical and small problems) which shows that there is no "good" uniform proof procedure, we can only give our current conjecture:

If one wishes to use predicate logic as a practical programming language,

then he is forced to introduce many of the usual constructions of programming languages. In other words control structures in a conventional programming language are very useful in directing the proof search and making it more deterministic, and thus efficient. The same is true for data structures. Automating the task of introducing the right control and data structures is very hard and may be even impossible without the use of other sources of knowledge.

REFERENCES

[AlLu 70] Allen, J., Luckham, D.(1970): An interactive theorem-proving program, "Machine Intelligence" (B. Meltzer and D. Michie eds), vol. 15, pp 321-336.

[Bi 75] Bibel, W.(1975): Predicative Programming, Séminaires IRIA 1975/76, Roquencourt.

[BFS 77] Bibel, W., Furbach, U., Schreiber, J.(1977): Strategies for the synthesis of algorithms, 5th Conference on Programming Languages of the GI, March 78, Springer

[BledTy 75] Bledsoe, W.W., Tyson, M.(1975): The U.T. Interactive Prover, Memo ATP-17, University Texas, Austin.

[Col 72] Colmerauer, A., Kanoul, H., Pasero, R., Roussel, P.(1972): Un system de communication homme-machine en francais, Rapport preliminaire, Group de Researche en Intelligence Artificialelle, Luminy.

[Co 72] Cowen, R.H., Frisz, H., Grenadir, A. (1972): Some New Axiomatizations in Group Theory, Notice 72T-H2, Notice of the American Mathematical Society, June 1972, issue 138.

[Gr 68] Green, C.C.(1969): Application of theorem proving to problem solving, Proc. Int. Joint Conf. on Artificial Intelligence, Washington, D.C.

[Hayes 73] Hayes, P.J.(1973): Computation and deduction. Proc. Math. Foundations Comp. Sci. Symp., Czechoslovakian Acad. Sci.

[HeWo 74] Henschen, L., Wos, L.(1974): Unit refutations and horn sets, J. ACM 1974, vol. 21, pp 590-605

[HeLu 74] Henke von, F.W., Luckham, D.C.(1974): Automatic Program Verification III, Memo AIM-256, Stanford University.

[Hoare 72] Hoare, C.A.R., Wirth, N.(1972): An axiomatic definition of the programming language Pascal. Berichte der Fachgruppe Computer-Wissenschaften 6, E.T.H., Zürich, November 1972.

[ILL 75] Igarashi, S., London, R.L. Luckham, D.C.(1975): Automatic Program Verification I: A Logical Basis and its Implementation , Acta Informatica 4, 145-182.

[King 69] King, J.C.: A Program Verifier. Carnegie-Mellon University, Pittsburgh, Ph.D. Thesis 1969. See also: Proc.IFIP Congress 71, Amsterdam North Holland, pp 234-249.

[Kl 52] Kleene, S.C.(1952): Introduction to metamathematics, North Holland, Amsterdam.

[Ko 74a] Kowalski, R.(1974a): Logic for problem solving, memo 75, department of computational logic, University of Edinburgh.

[Ko 74b] Kowalski, R.(1974b): Predicate logic as a programming language, Proceedings of IFIP 1974.

[LMS 76] Luckham, D., Morales, J., Schreiber, J.(1976): A study in the applications of theorem proving, unpublished manuscript, AI Lab, Stanford University.

[LuNi 71] Luckham, D., Nilsson, N.(1971): Extracting Information From Resolution Proof Trees, Artificial Intelligence, 1971, 198-213.

[LuSuz 76] Luckham, D., Suzuki, N.(1978): Automatic Program Verification V, Memo AIM-278, Stanford University.

[Mi 75] Van der Brug, G.J., Minker, J.(1975): State-space, Problem-reduction, and theorem proving - some relationships, CACM Feb. 1975.

[Mo 73] Morales, J.(1973): Interactive theorem proving , ACM National Conference, August 1973, pp 441.

[NeOp 78] Nelson, G., Oppen, D.C.(1978): A Simplifier Based On Efficient Decision Algorithms, 5th POPL.

[Suz 75] Suzuki, N.(1975): Ph.D. thesis, Stanford University.

[Sc 75] Schreiber, J.(1975): Some new strategies for the interactive theorem prover, unpublished notes, Artificial Intelligence Laboratory, Stanford.

[Ta 75a] Taernlund, S.A.(1975): On logic programming, Computer Science Department, University of Stockholm.

[Ta 75b] Taernlund, S.A.(1975): An interpreter for the programming language predicate logic, Fourth international joint conference on artificial intelligence, Tblilisi.

[Tar 51] Tarski, A.(1951): A decision method for elementary algebra and geometry, Second Edition, Berkeley and Los Angeles.

Acknowledgement

Thanks are due to H. Evans, D. Oppen and S. Taernlund for several critical remarks.

A QUASI-PARALLEL CONTOUR FOLLOWING ALGORITHM

L. Méró
Computer and Automation Institute of the Hungarian Academy of Sciences
1111 Budapest, Kende u. 13/17. Hungary

Abstract

A procedure for extracting the contour lines and arcs from digitized TV pictures is described. Initially the picture is subjected to an edge detection operation resulting in short edge-line pieces /strokes/. The bifurcation points /corners/ of the contours are detected based on the variances of the angles of the strokes.

Mathematical criteria are stated on the arrays of strokes /streaks/ to be found among the corners in order to approximate the contours optimally. An algorithm is given which finds the streaks satisfying these criteria.

Any number of processors can work simultaneously in performing the algorithms; therefore the algorithm is called quasi-parallel.

Key words: Picture processing, Contour following, Corner finding, Minimal cost path, Parallel algorithms.

1. Introduction

The results to be reported are details of an intelligent hand-eye system which recognizes and assembles industrial objects /Vámos, 1977/. The surfaces of the objects are homogeneous and surrounded by line and arc pieces. This paper presents an algorithm which provides these contour lines from a digitized TV picture. The result is a sight graph in the following sense:

Definition: A sight graph is a planar drawing. Its nodes are points in the plane and its edges are lines and arcs between the nodes.

Elliptic arcs are approximated by several circular arcs in the sight graph.

The sight graph supplies the input of a heuristic grammatical recognition procedure /Gallo, 1975/.

The basic element is a stroke. A stroke is an optimally fitted line segment found in a window of the noisy picture. A stroke generating algorithm is used in the course of our contour following. This algorithm provides the strokes in a window of the picture and a number between 0 and 1 as a measure of the quality of the fitting. An effective local

visual operator was described by Hueckel /1971/ and a simplified and fast version of it was reported by Mérő-Vassy /1975/.

The main task is to determine the nodes of the sight graph and to find sequences of strokes which have an optimal fit with the contour lines of the objects. These sequences of strokes are called streaks.

The strokes are generated in pre-determined windows independently thus any number of processors can work simultaneously in generating the streaks. The further algorithms are also constructed so that any processor could work simultaneously on them; therefore the algorithm is called quasi-parallel.

Another streak generating algorithm was reported by Ramer /1975/. This algorithm also works with quasi-parallel generated strokes, but its aim was not to produce a sight graph but long streaks at minimal cost.

A corner finding algorithm is described in § 2., which finds the nodes of the sight graph of degree 3 or more, using the streaks. A somewhat similar algorithm was described by Perkins and Binford /1973/ but they were searching corners of a given type.

2. A corner-finding algorithm

Let the input picture be covered by windows of k x k picture points, each overlapping its neighbour with m points in common, and let the stroke generating algorithm be executed in every window. The stroke matrix and the weight matrix are now defined as follows: let the j-th element of the i-th row of the stroke matrix be the angle of the stroke found in the j-th window of the i-th row of windows. If no stroke were found in that window, a special sign should be put in that place of the stroke matrix. The corresponding element of the weight matrix is the weight value W of that stroke.

The corner-finding algorithm is based on the following considerations: if at a point $/x_o, y_o/$ more homogenous regions are joining /i.e. the sight graph has a node of degree 3 or more at this point/ then in some neighbourhood U_o of $/x_o, y_o/$ taking an arbitrary straight line e crossing the point $/x_o, y_o/$ either the variance of those strokes which are in U_o and above the line e or the variance of those below e cannot be zero. Let $S_e /x_o, y_o/$ denote the larger of these two values. If $/x_o, y_o/$ is a corner point then the relation $S/x_o, y_o/ = \min_e \{S_e/x_o, x_o/\} > \emptyset$ will hold. Furthermore, the function $S/x, y/$ as defined above has a local maximum at the corner point $/x_o, y_o/$. Thus corner points are found by calculating $S/x, y/$ and locating the local minima.

The notion of variance in the preceding train of thought needs some more clarification, for the angles turn out to be periodical with a periode of π. Therefore the number σ^2 will be considered the variance of the angles $s_1 \ldots, s_t$ if $\sigma^2 = \min_s \left\{ \frac{\sum_{i=1}^{t} (s \div s_i)^2}{t-1} \right\}$ holds where $\div$ denotes the difference between the angles mod π. In the case of real

numbers this definition is equivalent with the original definition of variance.

3. The mathematical criteria on the streaks

Suppose, for the moment that the corner finding algorithm has produced all the nodes of the sight graph of degree 3 or more. The truth of this will be examined later and the cases of errors will be treated there, too. Now the stroke matrix and the weight matrix derived from the input picture are given and some of the strokes are marked as corners.

The original weight values W' measuring the goodness of the fitting of the strokes are transformed between and $/0 < \varepsilon_1 < \varepsilon_2/$ so that $W = \varepsilon_1 + (\varepsilon_2 - \varepsilon_1)(1 - W')$

The condition $\varepsilon_1 > 0$ is necessary because the streaks are to be assembled for the shortest paths of strokes /in some sense to be defined later/. The ratio of ε_1 and ε_2 will express how much the shortness of the path versus the goodness of the strokes constituting the streak should be taken into consideration when the streaks are being assembled.

Some denotations will be introduced. Let G be a graph and let the nodes of the graph be those elements of the stroke matrix where a stroke was found. Let the edges of G be among those nodes which are neighbouring in the stroke matrix, i.e. which have originated from overlapping windows. Let $\beta_1, \beta_2 \ldots, \beta_N$ denote the nodes of G, and $\alpha_1, \ldots, \alpha_c$ denote those of them which are marked as corners. The weight value W_β is assigned to every node β.

Before stating the criteria which should be satisfied by the streaks we need some more definitions.

Definition: The cost of a path $U = /\beta_1, \beta_2, \ldots, \beta_n/$ in G is defined as $c/U/ = W_{\beta_2} + \ldots + W_{\beta_{n-1}}$. The cost of the minimal cost path between β_1 and β_2 in G is denoted by $k/\beta_1, \beta_2/$.

Definition: Let α_i and α_j be two corner points in G so that $i < j$, and $U = /\alpha_i, \beta_1, \beta_2, \ldots, \beta_m, \beta_{m+1}, \ldots, \beta_n, \alpha_j/$ be a path between α_i and α_j. The points β_m and β_{m+1} are called the midpoints of U if $W_{\beta_1} + \ldots + W_{\beta_{m-1}} < W_{\beta_{m+1}} + \ldots + W_{\beta_n}$ and $W_{\beta_1} + \ldots + W_{\beta_m} \geq W_{\beta_{m+2}} + \ldots + W_{\beta_n}$ holds.

Definition: Two paths U and V in G are called equivalent if there exist paths $U_o=U, U_1, \ldots U_{n-1}, U_n = V$ in G such that for every $0 \leq i < n$ $c/U_i/=c/U_{i+1}/$ and U_i and U_{i+1} have a common midpoint.

Clearly an equivalence relation is defined among the paths.

Now we can state the criteria to be satisfied by the streaks.

Criterion a/ A streak is a path in G between two corner points. For a streak $/\alpha_i, \beta_1, \ldots \beta_n, \alpha_j/$, for every $1 \leq s \leq n$

$$\min\{k/\alpha_i, \beta_s/,\ k/\alpha_j, \beta_s/\} \leq \min_{t \neq i,j}\{k/\alpha_t, \beta_s/\}$$

must hold.

This criterion says that a streak must not pass near to a corner other than its ends. E.G. on Fig. 1. the corners α_1 and α_3 obviously should not be connected by a streak, however the minimal cost path between them does not contain α_2.

Criterion b/ For every $1 \leq s \leq n$

$$\min\{k/\alpha_i, \beta_s/,\ k/\alpha_j, \beta_s/\} = \min\{(W_{\beta_1} + \ldots W_{\beta_{s-1}}), (W_{\beta_{s+1}} + \ldots + W_{\beta_n})\}$$

must hold.

This criterion excludes the case that from any point of the path it could be completed "less expensively", i.e. the streaks should follow the contour lines optimally.

Criterion c/ For the midpoint β_m

$$\min_{t \neq i}\{k/\alpha_t, \beta_m/\} = k/\alpha_j, \beta_m/$$

must hold and one of the minimal cost paths from β_m to α_j must pass through β_{m+1}. The same should hold for the midpoint β_{m+1} if m and m+1 and i and j are interchanged.

This criterion completes criterion b/ which does not force the streak to be of minimal cost at the midpoints. E.G. on Fig. 2. the path through β_m and β_{m+1} satisfies b/, however the path through γ may be of less cost.

Criterion d/ From every class of equivalence of the paths which contain paths satisfying a/, b/, c/, precisely one path should be chosen as a streak.

This criterion warrants that every contour line should be followed by streaks and that all of them should be followed by only one streak.

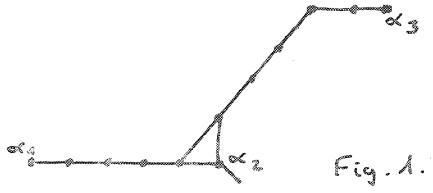


Fig. 1.

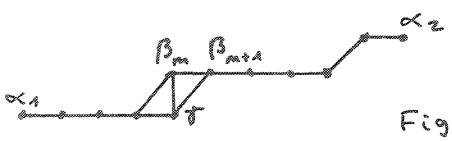


Fig. 2.

4. Assembling the strokes into streaks and the streaks into the sight graph

The algorithm assembling the strokes into streaks creates a kind of potential field around the corner points graded by the weight values of the strokes. The midpoints of the streaks are detected at the points where two potentials originated from different corners are meeting and the sum of the two potential values have local minima. Then the streaks are assembled following back the way of propagating the potential field. A detailed description of this algorithm and the proof that the streaks assembled satisfy criteria a/ - d/ will be described in a forthcoming paper.

As a result of this algorithm the nodes of the sight graph are given and the edges of it are approximated by a sequence of strokes. The assembly of the edges of the sight graph from the streaks is performed as described in /Mérő-Vámos, 1976/. The main idea is that an arc is defined if the differences between the angles of some consecutive strokes are of the same sign , and lines are detected if the difference of the angles of consecutive strokes is small.

Using only one processor the production of the sight graph from one 256 x 256 points TV picture takes about 15 s on a 16 bit, 24 K, 1 μs cycle time minicomputer. Serial sections of the algorithm requre less than 0,1 s, thus by using more processors the processing time can be reduced almost at will.

Acknowledgement

The author wishes to thank Prof. T.Vámos for the great help and constructive remarks during the writing of this paper.

References

V.Gallo: A Program for Grammatical Pattern Recognition; 4th IJCAI, Tbilisi, USSR, 1975, 628-634

M.H.Hueckel: An Operator which Locates Edges in Digitized Pictures; J.ACM 18 /1971/, 113-125

A.Martelli: Edge Detection Using Heuristic Search Methods; Comp. Graph. and in. Proc. 1 /1972/ 169-182

L.Mérő, Z.Vassy: A Simplified and Fast Version of the Hueckel Operator; 4th IJCAI, Tbilisi, USSR, 1975, 650-655

L.Mérő, T.Vámos: Real-time Edge Detection Using Local Operators; 3rd IJCPR, Coronado, California, 1976, 31-36

U.Montanari: On the Optimal Detection of Curves in Noisy Pictures; Comm. ACM 14 /1971/ 335-345

W.A.Perkins,T.O.Binford: A Corner Finder Visual Feedback; Comp. Graph. and Im. Proc. 2 /1973/ 355-376

E.U.Ramer: Transformation of Photographic Images into Stroke Arrays; IEEE Trans. Circ. and Systems CAS-22 /1975/, 363-373

T.Vámos: Industrial Objects and Machine Part Recognition; Chapter 10 in Applications of Syntactic Pattern Recognition /Ed. K.S.Fu/, Springer, Heidelberg, 1977

ADVICE TABLE REPRESENTATIONS OF CHESS END-GAME KNOWLEDGE

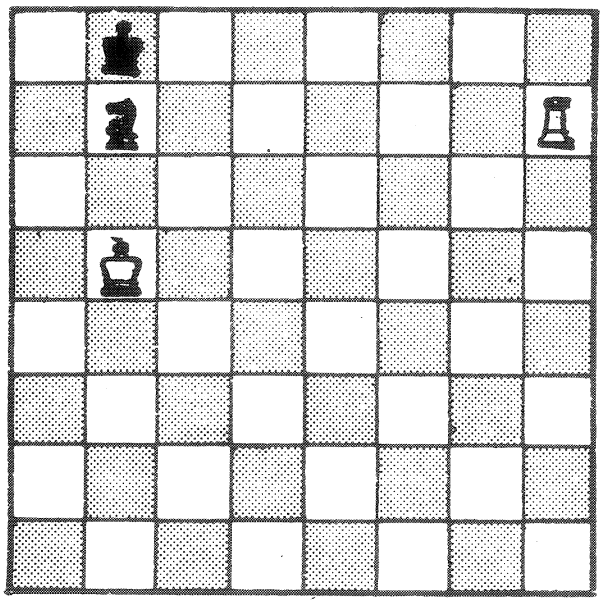
Donald Michie, Machine Intelligence Research Unit
University of Edinburgh, Hope Park Square, Edinburgh EH8 9NW
and
I. Bratko, Josef Stefan Institute, Ljubljana, Yugoslavia

Abstract

Computer implementation is described of the chess ending king and knight versus king and rook, and the trade-off between search and domain-specific knowledge is examined.

Introduction

Except for a few categories of positions, from which the rook's side can force a win, the ending King-Rook-King-Knight is thought to be a fairly straightforward draw. But the draw is not invariably agreed in tournament play and has on at least one recorded occasion slipped through a chess-master's fingers.[10] Moreover treatments of this ending in the standard books have been shown by computational analysis to be severely inadequate and marred by errors. As an arbitrary example, the following position is given by Fine[5] as a draw. Kopec and Niblett[7] report that it can be won by Kc6 in 15 moves. Their investigations, supported by experiments with human players, show that the KNKR problem, sometimes viewed as trivial, is so deep and complex that even masters must be said to have only a partial grasp. The size of the space of legal positions, after drastic reduction by disregarding symmetric cases, is approximately three million.



Where a win by the rook's side is possible against best defence, the forcing line can require as many as 27 moves (54 ply) to capture the Knight, according to Strohlein[11]. Fine[5] devotes 8 pages to its detailed execution. The broad strategy is to create mating threats so as to force the knight's capture, after which mate can be delivered in at most a further 16 moves (32 ply)[4]. The present note is concerned with the weaker side's task of frustrating these threats.

With the aid of an "Advice Taker" facility[8], a compact strategy was specified for the knight's side to enable drawn positions to be indefinitely defended (until terminated by the 50-move rule). Construction of such a strategy can be approached by negating the attacker's goals. The corresponding broad advice, labelled so as to display its structure, is as follows:

1. Avoid mate.
   1.1 Keep the king as far from the edge as possible.

1.2 Keep the king as far from the corners as possible.

2. Preserve the knight.

2.1 Keep the knight as near the king as possible.

It is interesting to consider what would be the effect upon the amount of search required to select a draw-preserving move if advice were added incrementally, a piece at a time.

Advice 1: "avoid mate"

The program searches forward along all paths, terminating the branching tree of possibilities only at check-mate positions or when a depth of 85 ply has been reached, - summing the two worst-case figures cited earlier. A draw-preserving move can always be extracted from such a search-tree by the minimax rule. The branching ratio of search exceeds 10. Hence the minimal advice, embodying only the applicable rules of chess, demands a search of more than astronomical proportions,- in excess of 10^{85} moves in the search tree.

As above plus Advice 2: "preserve the knight"

In the worst case a 54-ply search is required to ensure selection of a safe knight-preserving line.

As above plus Advice 2.1: "keep king and knight together"

A 12-ply search is required to ensure fulfilment of this advice over and above the two preceding items (but as will appear with some risk of error).

As above plus further refinements, including tactics for preserving king-centrality

An Advice Table thus equipped (Table 1) performs satisfactorily on the basis of a 4-ply search.

In fact it conducts the KNKR defence at least as well as a chess-master. Contrary to what was universal belief before the Edinburgh experiments, even chess-masters do not have a complete grasp of the ending. We now understand the KNKR problem well enough to be able to pose tests to chess-masters which they are virtually sure to fail.

How Advice Tables work

The basic concept of ALl is to adopt the master's attitude that patterns and goals are what counts, whereas individual moves are nothing but a chore, to be worried about separately. This division of function is shown in Figure 1.

A board position for which we want to find a move is matched by the system column by column against the rules of the Advice Table until a match is found. A rule is read from top to bottom, not from left to right. The upper part of each rule, by its pattern of "Y"'s ("yes"), "N"'s ("no") and "-"'s ("don't care") makes a selection from the predicates OKEDGE ("our king is on the edge"), OKONSEP ("our king and our knight are separated") and CORNCASE ("corner case"), negating those for which the symbol "N" is entered. For example the third rule, R2, matches with any position in which our king is on the edge and our king and knight are not separated. The bottom part of the rule is then taken as the "advice", - in this case the list of integers (1, 5, 6, 7, 8, 11, 12).

It is now up to the search module to find a strategy which satisfies the first advice on the list (advice "1" in our example) if this can

be done; failing that, then the second (advice "5"); failing that, then the third (advice "6") etc. The full interpretation of each such advice number is looked up in the "advice library" shown in the bottom half of the diagram. Thus advice "1", KILLROOK, consists of various goals, TRDEAD, NOT MATE, NOT ONLOST and TRDEAD OR CHECK. How does the search module discover whether a strategy exists which is guaranteed to satisfy all these pieces of advice?

For this we need Barbara Huberman's concept of a forcing tree (see Figure 2). The search module makes all possible moves and replies to a preset depth (4 ply in the KNKR case) to see whether a sub-tree of the tree thus grown possesses the properties of a forcing tree. These are:

1. Only one branch (move) from each us-to-move node.
2. All legal replies at each them-to-move node.
3. All terminal nodes satisfy the "better" goals.
4. All nodes other than the root node satisfy the "holding" goals.

An example of a "better" goal is TRDEAD ("their rook dead"). An example of a "holding" goal is NOT ONLOST ("our knight is not lost"). The second represents a desirable feature of the position which we are not prepared to sacrifice for any "better" goals however attractive.

What does AL1 do with its forcing tree once it has managed to construct one? Two main options exist:

(1) to print it out for examination, discussion, editing etc. by the user.

(2) to invoke the playing module and execute it as a strategy for the next 4 moves (2 by "us" alternating with 2 by "them"). The definition of "forcing tree" guarantees that at the conclusion of this exchange of moves all the "better" goals will have been achieved and all the "holding" goals conserved. The new position thus reached can now be fed back to AL1 and a new cycle initiated. In the special case that no "better goals" are specified, the forcing tree is executed for one move only, and control re-enters the Table after the opponents' reply ("weak" mode). The default goal is then adopted that the node should be on the lookahead horizon as set by the depth bound.

Experiments and results

Using the advice-taking package AL1[8] written in the POP-2 programming language[3], advice was communicated to the computer in the form of this single Advice Table. Preparation and check-out of a substantially correct version of the Advice Table occupied six man-weeks on the part of one of us (I.B.). Apart from his own chess expertise (approximately USCF 2300 rating) the main source of knowledge was Fine[5] and Keres[6].

The Table was set up on a PDP 10 computer using the AL1 package. A US National Master currently rated 2350 on the USCF scale, engaged the system in play for a total of 5 hours, with halts to enter new trial starting positions. The machine system was in all cases able to preserve the draw, and also to maintain the degree of centrality of its king. Three degrees of centrality are distinguished: the central 16 squares, the rank or file next to an edge, and an edge. Although it is possible to allow the king to be peripheralised and still draw, the defence becomes trickier as each level of centrality is ceded. The Advice Table thus resists, as does a Master, needlessly entering territory which, although still defensible, is adverse.

Correctness of the Table's strategy for all cases was subsequently belied by the discovery of a class of special positions, believed new to chess theory, in which the only correct continuation worsens the separation of king and knight. Two examples are shown in Figure 3. We are seeking to add a fifth rule to the Table which should precisely characterise and prescribe for this class.

The AL1 formalism was partly motivated by the need for a representation in which proof methods might be easier to develop. A proof scheme for an Advice Table is described by Bratko[2], using the simpler case of King-Rook-King.

Discussion and summary

The main features of interest are:

1. No computer implementation of Master skill in any non-trivial chess end-game has previously been achieved, other than as an exhaustive tabulation[1]. To check the possibility that such skill might lurk unsuspected in the better tournament programs, the task of defending knight against rook in three selected trial positions was submitted to Atkin and Slate's CHESS 4.5[9]. Their program although searching to a depth of 7-8 ply for selection of each move, mis-played in two of the three trials. Evidently the combination of deep search with good general chess heuristics cannot substitute for heuristics specific to the given end-game. In these tests the human opponent, although of "expert" level (USCF rating > 2000), also made errors as shown by subsequent analysis.

2. In comparison with exhaustive tabulation our representation achieved a memory saving of several hundredfold. As earlier stated, the number of king-knight-king-rook positions is approximately three million.

3. Experimental evidence affords an inadequate certificate that a Table is correct. Certification must be based on formal proof. Successful proof of an Advice Table has been possible for the easier problem of King-Rook-King.

Additional remark

As a vehicle for the transfer of chess knowledge from human to human, Advice Tables may also serve as a valuable supplement to the tradition of natural language text interspersed with diagrams. The Tables are more precise, more complete, and more open to systematic checking. The fact that they can also be run directly on the machine adds, of course, an additional dimension.

Acknowledgment

We are indebted to our colleague, US Master Danny Kopec, for assisting with trials of the program and for helpful discussion.

References

1. V.L. Arlazarov & A. Futer, in press, in Machine Intelligence 9. Chichester: Ellis Horwood, and New York: John Wiley.
2. I. Bratko, in press, in Inform. Proc. Letters.
3. R.M. Burstall, J.S. Collins & R.J. Popplestone (1971) Programming in POP-2. Edinburgh University Press.
4. M.R.B. Clarke (1977) in Advances in Computer Chess 1, pp. 116-118. Edinburgh University Press.
5. R. Fine (1964) Basic Chess Endings. New York: McKay (1st edition 1942).

6. P. Keres (1974) devotes two pages to the play of the knight's side in Practical Chess Endings. London: Batsford.
7. D. Kopec & T. Niblett, in press, in Advances in Computer Chess 2.
8. D. Michie (1976) Computer Bulletin, ser.2, 10, 12-14. I. Bratko, D. Kopec & D. Michie, in press, Computer J.
9. D. J. Slate & L.R. Atkin (1977) in Chess skill in man and machine, pp. 82-118. New York: Springer Verlag.
10. Steinitz-Neumann. In the position White: K at e6, R at h7; Black: K at f8, N at h6; Neumann played Ng4, which loses. Ng8 is necessary.
11. T. Strohlein (1970) Untersuchungen über Kombinatorische Spiele. Dissertation for Dr. Rer. nat. Munich: Technische Hochschule.

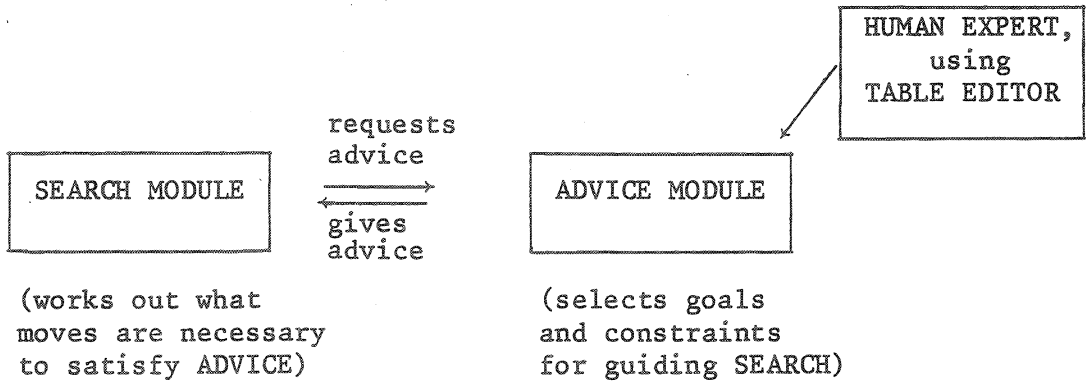


Figure 1. Division of function in AL1: Separation of search from advice.

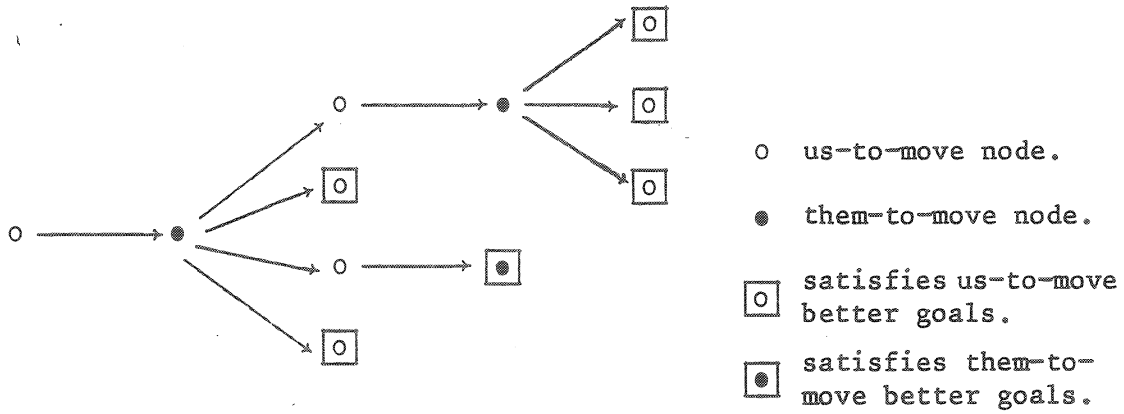
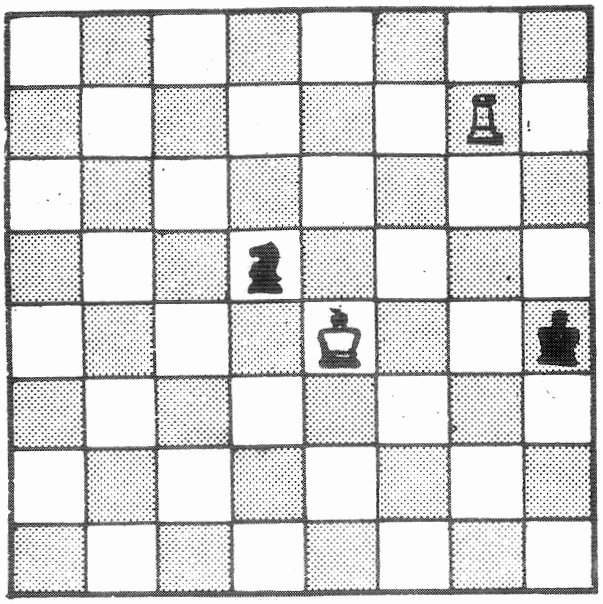
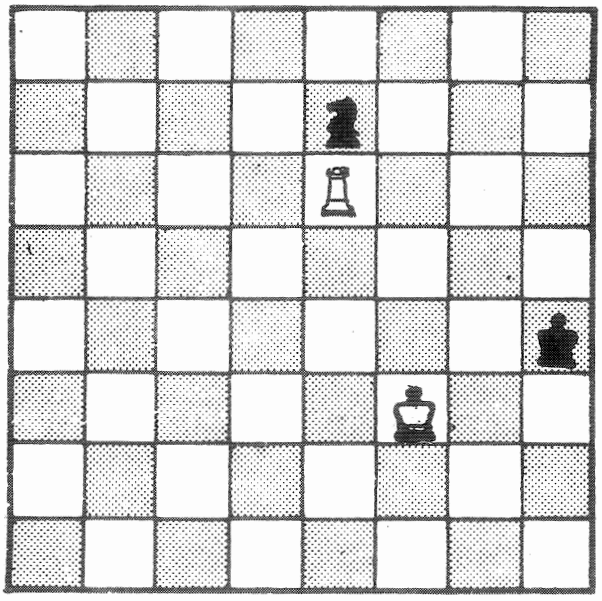


Figure 2. FORCING TREE depicts a strategy ensuring satisfaction of "better" goals in at most 4 ply (2 moves by "us" alternating with 2 moves by "them"). At us-to-move positions the tree only has a single branch, corresponding to the move designated by the strategy. All nodes except the root node are guaranteed to satisfy any specified "holding goals".



BLACK TO MOVE

Figure 3A. N-f6 would immediately bring N and K closer together, but N-c3 is the only drawing move.



BLACK TO MOVE

Figure 3B. N-f5 would bring N and K closer, but N-g8 is the only drawing move.

Figure 3. Two positions from the category for which qualification is demanded of the advice "Keep the knight as near the king as possible".

ADVICE TABLE

KNKR Table	CR	R1	R2	ER
OKEDGE	-	N	Y	-
OKONSEP	-	N	N	-
CORNCASE	Y	-	-	-
	9	1	1	10
		2	5	11
		3	6	12
		4	7	
		11	8	
			11	
			12	

rules

condition predicates

lists of pieces of advice

ADVICE LIBRARY

pieces of advice

	Better-goals		Holding-goals											
			us-to-move			them-to-move								
	OKONNDLT (utm)	TRDEAD (ttm)	NOT MATE	NOT ONLOST	ONSAFE2P	TRDEAD OR CHECK	NOT ONOKATT	OKEDNDGE	OKCONDGE	OKONDLE2	KINGSCLOSE	OKONDLE3	TKONDGT1	OKONDEQ4
1:KILLROOK	-	Y	(Y)	Y	-	Y	-	-	-	-	-	-	-	-
2:HOLD1	-	-	(Y)	Y	Y	-	Y	Y	Y	Y	Y	(Y)	Y	-
3:HOLD2	-	-	(Y)	Y	Y	-	Y	Y	-	Y	Y	(Y)	-	-
4:HOLD3	-	-	Y	Y	Y	-	Y	-	Y	Y	Y	(Y)	-	-
5:HOLDEDG1	-	-	Y	Y	Y	-	Y	-	Y	Y	-	(Y)	Y	-
6:HOLDEDG2	-	-	Y	Y	Y	-	-	-	Y	-	-	Y	Y	-
7:HOLDEDG3	-	-	Y	Y	Y	-	-	-	-	Y	-	(Y)	-	-
8:HOLDEDG4	-	-	Y	Y	Y	-	-	-	-	-	-	Y	-	-
9:CORNCASE	-	-	(Y)	(Y)	(Y)	-	-	-	-	-	-	-	-	Y
10:APPROKON	Y	-	Y	Y	Y	-	-	-	-	-	-	-	-	-
11:SURVIVE1	-	-	Y	Y	Y	-	-	-	-	-	-	-	-	-
12:SURVIVE2	-	-	Y	-	-	-	-	-	-	-	-	-	-	-

Table 1. The upper table is the KNKR Advice Table as written and tested. The integer lists index a repertoire of 12 pieces of advice. These are shown in the lower table expanded in the form of calls to indicated subsets of the 14 goal predicates listed along the top. "Y"s enclosed in brackets are logically implied by the other predicates selected in the same row. The symbols in parentheses (utm) and (ttm) mean "us-to-move" and "them-to-move" respectively. Fuller description and explanation is to be found in the papers of reference 8.

REPRESENTATION OF PHONETIC AND PHONEMIC KNOWLEDGE IN A SPEECH UNDERSTANDING SYSTEM

Renato DE MORI * e Pietro LAFACE **

* Istituto Scienze Informazione - Università di Torino - Corso Massimo d'Azeglio 42 - 10125 TORINO - Italy

** CENS - IENGF - Istituto di Elettrotecnica - Politecnico di Torino - Corso Duca degli Abruzzi 24 - 10129 TORINO - Italy

ABSTRACT

The paper describes the organization of knowledge source and the algorithms for evaluating hypotheses of the front-end acoustic classifier of a Speech Understanding System.

1 - INTRODUCTION

It has been demonstrated that there are no simple relations between phonemes and their acoustic realizations in a spoken language.

An analysis of the time evolutions of the acoustic parameters of speech shows that a complex coding, involving rules that may be context-dependent, relates abstract representations of speech to the physically observable acoustic patterns / 1 ÷ 3 /. These relations are not only parametric but very often they involve qualitative descriptions of speech patterns; furthermore, they are fuzzy in nature. The above statements are motivated considering that the most popular representation of speech is by spectrograms and most of the phonetic events can be described in terms of imprecise descriptions of the spectrogram patterns. For example, vowels are represented by high energy in the speech signal, little varying and well evident formants and energy at low frequencies being higher that energy above 5 kHz. The vowel / i / has a low first formant, an high second formant and an high third formant; stop consonants have characteristic formant transitions toward the successive vowel; fricatives have remarkable energy at high frequencies.

The degree of fuzziness that appears from the above sentences suggests to consider the qualitative attributes, on which the spectrogram interpretation is based, as labels for fuzzy sets because they may be present in the speech spectrograms with different grades of evidence. Furthermore, the evidence is reduced when sounds are pronounced in a sentence context / 4 /. It is known that there is a strong variability of the patterns, and a possibility exists that evidence of some acoustic cues has been lost in a spectrogram of a sentence that has not been enounciated clearly.

For these reasons, it is more convenient to express a grade of evi

dence by the subjectivistic concept of a fuzzy set with a heuristic assignment of a membership function instead of employing probability density functions that may be not satisfactorily estimated from even a large training set. Similar considerations may be done for those linguistic events that may be characterized only by acoustic parameters and for which decisions are based on the calculation of geometric inter-sets distances in a Euclidian space.

2 - PROCESSES INVOLVED IN PHONEME LABELLING

Fig. 1 shows a precedence graph of the processes composing the algorithm used for emitting hypotheses at the syllabic level; other papers describe the methodologies used for verifying hypotheses / 5 / and for processing informations at the lexical / 6 /, the semantic and the syntactic level / 7 /.

After sampling the signal at a rate of 20 kHz, a spectral analysis based on Fast Fourier Transformations (FFT) and Linear Prediction (LP) is performed. Such operations may be ascribed to the "auditory level" of the speech understanding system (SUS). The spectra are then processed in order to obtain some "gross spectral features" (see / 8 / for detail) and the position of the vowels are first hypothesized by process Γ using a syntactic method described in / 8 /.

Vowel detection and the extraction of context in dependent features for consonants corresponds to a so called "precategorical classification" because it can be performed without the knowledge of the phonetic contexts. Detection of vowel position may be ambiguous making it possible to have more than one processes of precategorical classification PC_i ($i = 1,2, \ldots, N1$), each process correspondign to a particular selection of vowel positions. Precategorical classification belongs to the "phonetic level" of the SUS and is based on features that can be recognized independently of the context. The detection of such features is ambiguous and may lead, for each process PC_i, to a variety of hypotheses about segmentation (i.e. more processes S_{ij}). For each hypothesized syllabic segment, an emission of hypotheses for the vowels is first performed; this emission is based on the detection of stable zones in the formant (whose extraction, performed by the process DF_{ij}, may be ambiguous) patterns and in their classification as belonging to vowel loci that are defined as fuzzy sets. Thus vowel recognition, the first step of the "phonemic level", may be ambiguous. For each choice of formants, a process V_{ijh} emits hypotheses about vowels and, for each choice of vowels in each syllable, context-dependent algorithms implemented by the process C_{ijhk}, are used for emitting hypotheses about consonants. Again, this operation may be ambiguous. Finally, the consonantal context may be used for detecting the cases of vowel reduction.

Requests for hypotheses verification can be issued at the vowel recognition level (new vowel positions are postulated and at the consonant recognition level (new precategorical classifications are postulated). Such requests cause repetitions of chains of processes whose first process is indicated by dashed lines in fig. 1.

3 - PHONETIC AND PHONEMIC KNOWLEDGE REPRESENTATION

Phonetic and phonemic knowledges are represented by augmented transition networks (ATN) that are subnetworks of the ATN representing the syntax of the natural language.

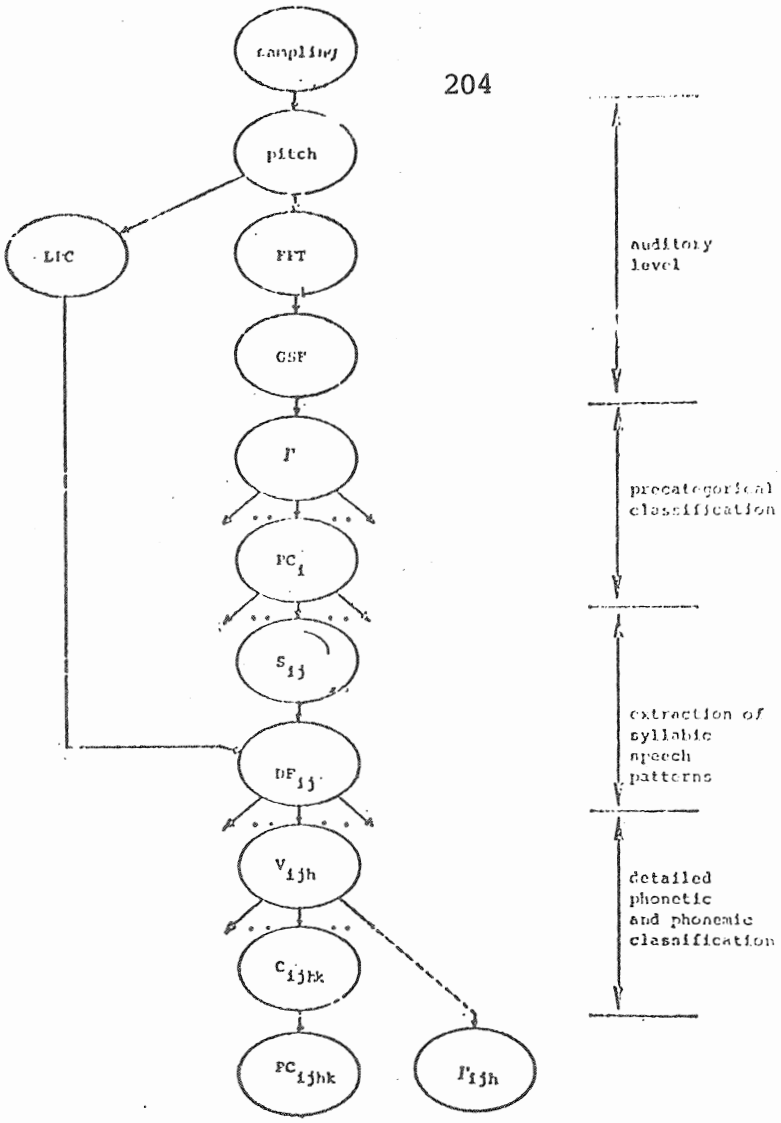
After precategorical classification, the speech signal corresponding to a sentence is segmented into pseudo-syllable segments (PSS) and for each type of PSS, phonetic and phonemic hypotheses are emitted and evaluated under the control of an ATN. Fig. 2 shows, as an example, the ATN for a PSS type made of a vowel in the time interval (t_1, t_2), a voiced (sonorant or nonsonorant-lax) consonant (t_2, t_3) and a vowel (t_3, t_4). Each transition of a network is consumed after a PUSH action that gives back a value of a membership function.

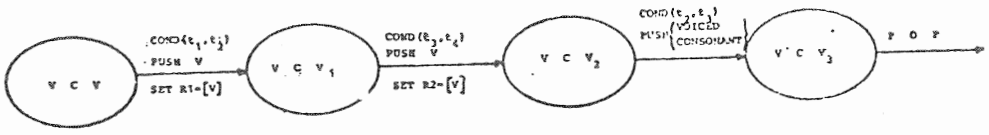
For example, the action PUSH ⟨NONSONORANT⟩ causes the evaluation of the hypothesis that the speech interval from time t_2 to time t_3 (the condition on the PUSH ⟨VOICED CONSONANT⟩) is nonsonorant. The hypothesis is evaluated under the control of phonetic rules and the value nonsonorant obtained is used for evaluating a total hypothesis. A total hypothesis corrsponds to a path from the state ⟨VOICED CONSONANT⟩ to the state VC14. The value of a total hypothesis is the minimum of the membership values computed for each arc of the path.

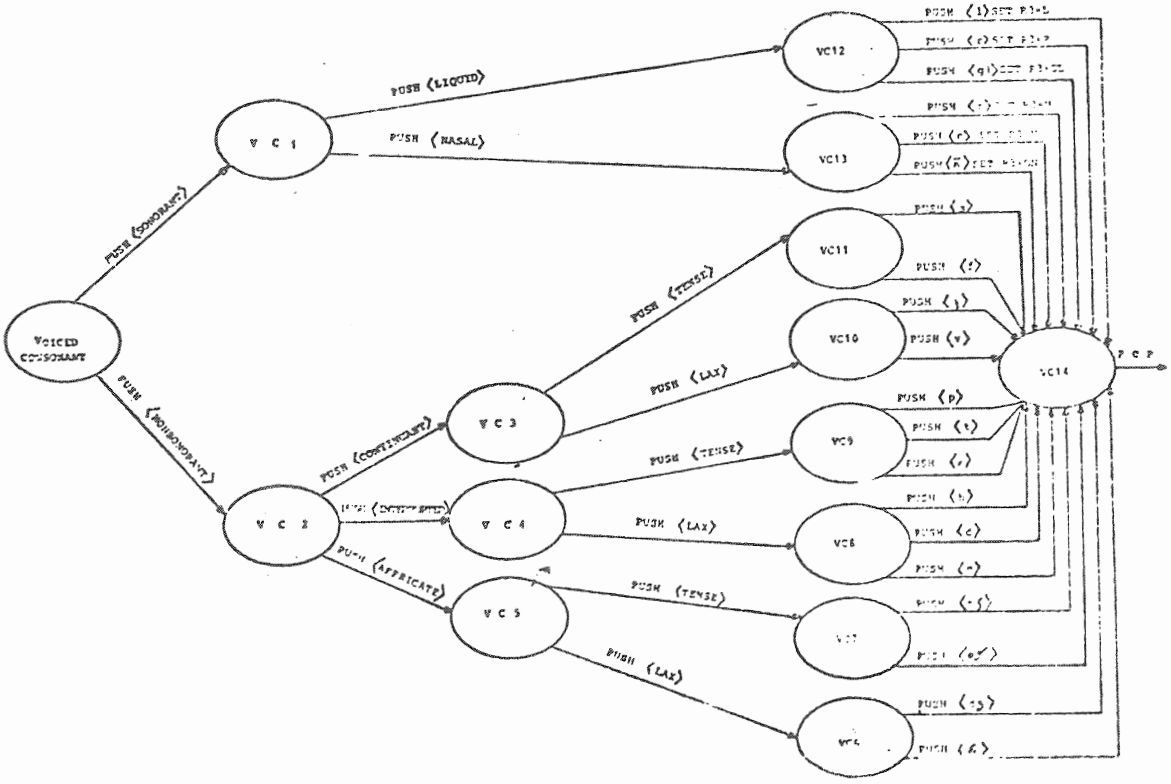
The action PUSH are ordered in accordance with the contextual effects of their results rather than in time. Thus, for example, for the PSS-type VCV the second vowel corresponding to the time interval (t_3,t_4) is hypothesized before the consonant that is pronounced before, in the time interval (t_2, t_3), because the knowledge of the two vowels is essential for hypothesizing the consonants. The recognized vowel, returned by the PUSH V actions are indicated as / V / and are stored into registers R1 and R2. The contents of these registers will be conditions for the recognition of consonants.

Emission of hypotheses under the control of the ATN types just introduced is ambiguous because several PUSH actions can be taken from many states. Furthermore, some PUSH actions require some features to be extracted. For example, the action PUSH ⟨NASAL⟩ requires that the formants of the PSS have been tracked previously.

For such reasons, the strategy that uses the phonetic and phonemic knowledge sources acts like an operating system that schedules the required tasks before performing a PUSH action, causes the storage of all the hypotheses whose value is above a confidence level on a data base and decide the transition from one state after having evaluated all the hypotheses represented by the PUSH actions associated with the arcs going out from that state. The arcs to be followed from each state are


sampling
pitch
LPC
FFT
GSP
F
FC_i
S_{ij}
DF_{ij}
V_{ijh}
C_{ijhk}
FC_{ijhk}
F'ijh
auditory level
precategorical classification
extraction of syllabic speech patterns
detailed phonetic and phonemic classification


V C V
COND(t_1,t_2)
PUSH V
SET R1=[V]
V C V_1
COND(t_3,t_4)
PUSH V
SET R2=[V]
V C V_2
COND(t_5,t_6)
PUSH (VOICED CONSONANT)
V C V_3
P O P


VOICED CONSONANT
PUSH (SONORANT)
PUSH (OBSTRUENT)
V C 1
V C 2
PUSH (LIQUID)
PUSH (NASAL)
PUSH (CONTINUANT)
PUSH (DENTAL STOP)
PUSH (AFFRICATE)
V C 3
V C 4
V C 5
PUSH (TENSE)
PUSH (LAX)
VC12
VC13
VC11
VC10
VC9
VC8
VC7
VC6
VC14
P O P

stored in an event queue and are ordered according with the highest membership value of the hypotheses they may generate. The evolution of a total hypothesis is aborted when its value falls below a confidence level. The policies associated with the strategy are very important for the performances of the system and are still under investigation.

Action associated with some transitions cause the writing of phonemic hypotheses on a data base.

The semantics associated to the syntactic rules represented by ATNGs basically concern the assignment of membership functions to fuzzy relations. Some of the relations involved in the automatic interpretation of spectrograms cannot be deterministic because more competing interpretations are often obtained from the same set of data; these ambiguities require each interpretation to be assigned a degree of consistence with the original data.

These degrees are used by the scheduling policy for the execution of the process that cause a transition to be consumed.

FIGURE CAPTIONS

Fig. 1 - Precedence graph of the processes composing the algorithm of emission of syllabic hypotheses.

Fig. 2 - Basic ATN for the recognition of vowel - Voiced consonant Vowel syllables.

REFERENCES

[1] A.M. LIBERMAN, "The grammars of speech and language" Cognitive Psychology I, Oct. 1970, pp.301-323.

[2] M.STUDDERT-KENNEDY, "Speech perception" Status Report No. SR 39-40, Haskins Laboratories, New Haven (Conn.), July/Dec. 1974, pp. 1-52.

[3] R.DE MORI, "Syntactic recognition of speech patterns" in Syntactic Pattern Recognition, Applications, ed. K.S.Fu, Springer Verlag 1977.

[4] P.MERMELSTEIN, "Distance measures for speech recognition - Psychological and Instrumental", Status Report on Speech Research, Haskins Laboratories July-Sept. 1976, pp. 91-103.

[5] R.DE MORI, P. LAFACE, E.MICHELIS, E.PICCOLO, "Emission and verification of hypotheses in a speech understanding system", Proc. IEEE Conference on Cybernetics and Society, San Francisco 1975, pag.284-290.

[6] R.DE MORI, P. TORASSO, "Lexical Classification in a Speech Understanding System Using Fuzzy Relations", Proc. IEEE-ASSP Conference, Philadelphia, 1976, pp. 565-568.

[7] M.COPPO, L.SAITTA, "Semantic support for a Speech Understanding system based on fuzzy relations"Proc. IEEE Conferences on Cybernetic and Society, Washington D.C., 1976, pag. 520-525.

[8] R.DE MORI, P. LAFACE, E.PICCOLO, "Automatic detection and description of syllabic features in continuous speech" IEEE Transactions, vol. ASSP-24,Oct. 1976, pp. 365-379.

THE LAST SURVEY OF REPRESENTATION OF KNOWLEDGE

Drew McDermott
Yale University
New Haven, Connecticut 06520, USA

Abstract: This is a survey of recent progress in "representation of knowledge." First the field is defined. Then progress in general techniques is surveyed. Then a catalogue of results in particular domains is given. Finally, a few recommendations are listed.

I The Problem

In this paper I will survey recent progress and the state of the art in the area of "representation of knowledge." This is an awkward topic to write on. As often happens in AI, the phrase is produced by taking two words, "representation" and "knowledge," highly charged with thousands of years of philosophical speculation, and putting them together to form a slogan. A glance at any AI conference proceedings will show that the phrase usually heads a list of papers with little in common. A paper surveying an area like this must either be completely unsystematic, or must reflect the author's prejudices to an embarrassing degree. This paper is of the second variety.

The first question to ask about knowledge representation is what distinguishes it from computer science in general. After all, it is the business of every computer program to solve problems in a given area of expertise. The program embodies the programmer's knowledge of that area, and it manipulates data structures representing information about a particular situation. So why isn't "representation of knowledge" the study of WHILE loops and hash tables?

The answer is that having knowledge embodied in a program is not the same as representing it. A program embodies a particular chunk of knowledge, whereas a representation system must specify how to represent any chunk of knowledge from a given domain. For example, the listing of a chess-playing program says nothing about how to add a new heuristic to it. A "chess-heuristics calculus" will specify a way of expressing any reasonable piece of advice for playing chess. Of course, even given a representation scheme, there will still be an element of trial and error in expressing a given fact, since the mapping supplied by the inventor of the system, its semantics, will go from expressions in the system to facts rather than the other way around. (A programming languages has semantics, too, but this semantics maps language constructs into computer operations, not into facts.)

Let us further stipulate that the job of a representation system is inference, in two senses. First, given the semantics of the notation, some expressions "follow from" others. (What "follow from" means varies a lot.) Second, an implementation of the notation must supply algorithms for manipulating data structures which stand for expressions: these algorithms carry out a set of inferences that

should match to some degree the inferences allowed by the semantics. The general structure of such a system is as in Figure 1. (Cf. Bobrow and Brown 1975.)

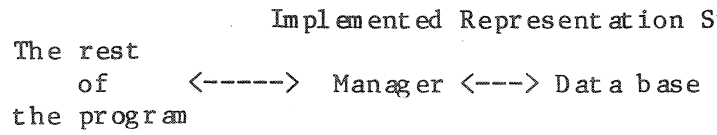


Figure 1 Structure of a Representation System

The rest of the program uses the representation system by asking the manager to store facts in the data base, and to retrieve answers to questions. The answers are inferences from the stored facts. We can distinguish between inferences made when facts are stored ("entry-time" inference) from inferences made when answers are sought ("retrieval-time" inference).

It is not useful in this field to distinguish plausible from "necessary" inference, since the data are always noisy. We can, however, differentiate "firm" from "speculative" inference. A firm inference is one made entirely within the representation system, such as the inference from, "John lives in the suburbs," to "John owns a car." A speculative inference is an inference made by the representation system, but passed back to the rest of the program for review. For example, from "John has blood in his urine," a representation system might infer, "Perhaps John has nephritis." The rest of the program must decide whether to accept this, on the basis of further inferences or tests made in the real world. Obviously, the sense of "follows from" that is involved here is different from the usual one, and is less well understood. (The distinction between firm and speculative inference may be the same as the distinction between "deductive" and "abductive" inference. Pople 1973)

Two observations about Figure 1 are in order. First, the algorithms in the data-base manager, while they may embody very complex knowledge, do not themselves represent anything (unless there is another representation system hidden in the manager). For our purposes, a representation system is more interesting the more facts are actually represented. Second, the efficiency of the entire program depends only partly on the efficiency of the representation system. For example, a problem solver might use a very selective representation system to retrieve possible actions, but then use them in an exhaustive way.

Why do people bother to write programs with the structure of Figure 1, when any given chunk of expertise can just be put in a program? The only possible reason is a concern with change in a program's expertise This can be as modest as a desire for ease of experimentation with the program, or as ambitious as a desire for a theory of learning.

With these observations in mind, we might expect a knowledge-representation system to be described as follows: a domain of facts and useful inferences are singled out for representation.

Then the notation of the system, and its semantics in the given domain, are specified; the reader is convinced of the breadth of the system. Next, algorithms are given for carrying out inferences on machine representations of the notations. These algorithms are shown to combine reasonable completeness and soundness (missed inferences and wrong inferences are rare) with reasonable efficiency. Finally, it is explained how change in a data base is managed.

Unfortunately we are in most cases very far from being able to carry this program out. There are unsolved theoretical problems at every step. So we must usually settle for a result of one of the two following kinds:

(1) A fragment of a notation, a semantic framework, and/or some domain-independent inference algorithms. For example, someone might suggest a notation for talking about temporal sequence and duration of facts, and algorithms for efficiently performing an apparently important class of inferences across time.

(2) A notation for a given domain, and a set of algorithms for performing some inferences on it. An example would be a chess-heuristics language, plus algorithms for making appropriate inferences about chess situations. For a result of this kind to be valuable there should be enough experience with the algorithms to show their worth.

Notice that results of the second kind usually draw on results of the first kind. A chess-heuristics language will probably require inference across time. In general applying a general result requires adding a large vocabulary of primitive symbols (cf. Schank 1975), and showing that the general algorithms can be specialized to this domain efficiently.

In the rest of this paper, I will survey results of both these types, under the headings "Techniques" and "Applications." The survey is incomplete, because of space limitations, my ignorance (especially of European work), and, of course, my prejudices. In general, I avoid the use of popular phrases (like "frame" and "production system") which have become hopelessly ambiguous.

II Techniques

Results in this class include discoveries of new notations, the semantics of general concepts like "cause" and "know," and programming techniques. The first two kinds have been sparse. Although some complaints have been heard about the notation and semantics of formal logic, it remains true that almost all of the working systems are based on predicate-argument ("propositional") notations, and that there has been very little work on radically new semantics for such notations.

II.A Programming Techniques

Most progress has been in the realm of implementation techniques for propositional systems. In principle, a data-base manager could store a new fact in data structures quite different from its external form. (Sussman 1975) In practice, there is usually a simple correspondence between data-base structures and expressions. Inference techniques may thus be thought of as operating on expressions directly.

One of the most popular techniques is resolution. (Robinson 1965) This is the technique of retrieving answers to a deductive question Q by chaining through rules of the form

P1 & P2 & ... & Pn -> Q

where the Q in the rule only has to unify with the Q in the question. Unification is the process of finding substitutions for the variables in the two patterns which will make them equal; these substitutions are used in P's to generate "sub-questions." For example, given the statement:

FRIEND(x, y) & OCCUPATION(y, AI-RESEARCHER)
-> MENTAL-STABILITY(x, LOW)

the question

MENTAL-STABILITY(FRED, z)

leads to the subquestion

FRIEND(FRED y) & OCCUPATION(y, AI-RESEARCHER)

This "chaining" continues until simple facts (like FRIEND(FRED, FRANK)) are reached; a conjunctive subquestion is handled by generating answers (like y = FRANK) to one conjunct, and testing them against the others. If the search succeeds, the top-level answer is z = LOW.

We can use a similar unify-and-detach scheme to infer Q from P->Q when P is added to the data base by the manager. This entry-time resolution is called forward chaining. Retrieval-time resolution is called backward chaining. Most systems allow the user to declare which way an implication is to be used. (This may be thought of as a simple language for expressing facts about how other expressions are to be used; these facts are not themselves accessed by resolution, but affect the way expressions are "indexed." See below.)

Resolution was publicly condemned a few years ago, but it lived on in the "AI languages" (Bobrow and Raphael 1974), and has now become a sturdy tool in the AI kit. There are two main reasons for its quiet success:

(1) It is being applied only where it works efficiently.

(2) It is used in conjunction with sophisticated "indexing" techniques.

Of these two, the first is more important. The best example of this is the MYCIN system (Shortliffe 1976), which chains through deductive rules about identifying bacterial infections. In MYCIN, resolution is limited in various ways for efficiency. In particular, a major source of complexity in resolution is its reliance on "generate and test" as its method for attacking conjunctive goals. In MYCIN, things are arranged so that conjuncts never share free variables; this ensures that there is at most one answer to each conjunct. There are other clever specializations used in this system, which might be of use in other situations.

In general, resolution systems work properly when the generate-and-test problem is not allowed to interact with "recursive rules," rules which can generate subgoals that unify with the same rules. (See Moore 1975 for a more thorough analysis.) For example, any axiom specifying the transitivity of a relation, such as

ABOVE(x, y) & ABOVE(y, z) -> ABOVE(x, z)

can cause trouble. (Sometimes subquestions can be caught before they feed back through the original rule. See Shortliffe 1976, McDermott 1977, Shapiro 1977b.)

Nowadays, all propositional representation schemes are "indexed." (Warren and Pereira 1977, McDermott 1975, Fikes and Hendrix 1977) That is, facts are not arranged in some long list, but are organized by easy access from the symbols that occur in them. That way, the symbols in a question can be inspected to find rules relevant to answering it. One way to do this is to point from a symbol to the facts whose conclusions it appears in.

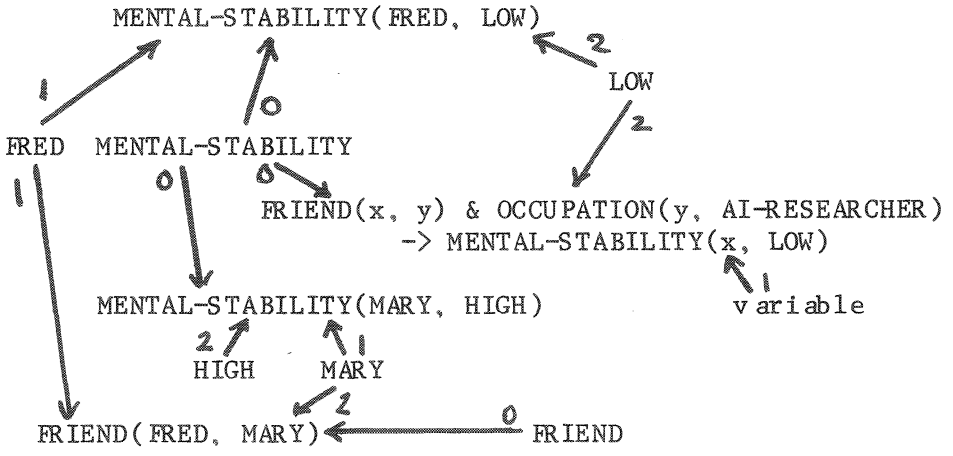


Figure 2 Simple Inversion Indexing of Expressions

Now, to answer a question, we use rules which are pointed to by the constants in it. In particular, we want to retrieve any expression which, for each constant in a given position, has either that constant or a variable in that position. This algorithm has retrieval time that depends linearly on the number of formulas containing the question symbols in the given positions. (Shapiro 1977a, Fikes and Hendrix 1977) I will call this "simple inversion" indexing, since the data structure depends on back-pointers from symbols to formulas. Other

systems use a discrimination-tree index structure (McDermott 1975, Rieger 1977, Ernst and Newell 1969), in which the formulas are found at the terminals of a tree of discriminations on constants found in a question. The access time for this algorithm depends mainly on the size of the question.

Figure 2 may remind you of "semantic networks," and it should. In some cases (Shapiro 1977a), a semantic network is nothing but an indexing scheme for resolution. But these networks are better known for their use of a kind of set-theoretic inference, "property inheritance," which has recently, in the hands of researchers like Brachman (1977) Fahlman (1978), Hayes (1977), and Hendrix (1975), become a technique of some power. I will use Fahlman's scheme as an illustration. Let us say we wish to encode the fact,

```
FORALL x (x IN PERSONS ->
          (EXISTS (y z)
               (AND (y IN PERSONS) (z IN PERSONS)
                    FATHER(x, y) MOTHER(x, z)
                    LOVES(y, z)))
```

In Fahlman's system, we index this as follows

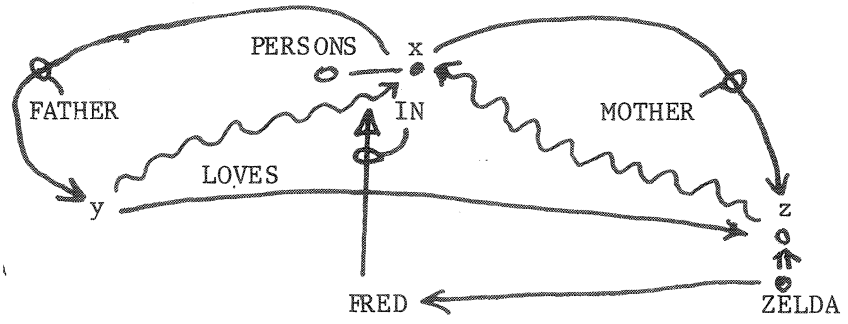


Figure 3 Short-Circuit Indexing

which also includes the structures for "Fred is a person" and Zelda is Fred's mother." (I have used the usual abbreviation convention for binary relationships to draw a link from first argument to second argument labeled with the relation; the actual list structures are the same as in Figure 2.) Solid nodes are universally quantified variables; open nodes are object symbols, or existentially quantified variables when a wavy arc points to a dominating universal. The double-shafted arrow is a kind of equality: "ZELDA = z(FRED)," when z is taken as a Skolem function.

Inference on this structure is not done by unification and chaining, but by a marker-propagation scheme. To answer the question, "Who does Fred's father love?," we start at "Fred " and mark that node and all the nodes that can be reached by following "IN" links this marks "PERSONS." Now we propagate a mark across a "FATHER" link from all universals bound to marked sets; in this fragment, "y" will be marked. Next a mark is propagated across "LOVE" arcs from all previously-marked nodes this gives us "z." In order to get "ZELDA," we take advantage of an ingenious feature of Fahlman's scheme: when "FRED" is marked, the link from "ZELDA" will act as a "short-circuit,"

so marking "z" marks "ZELDA" as well.

More detail on the powers of this scheme can be found in Fahlman (1978). Notice that the scheme requires search through supersets of a node, so it does not beat ordinary resolution except by some constant factor. The manipulations at each level are so simple, however, that this factor can be significant; Fahlman points out that special-purpose hardware could do these searches in parallel.

Another point to make is that the retrieval algorithm causes it to appear as if the large structure surrounding "x" in Figure 3 had been copied, with its variables instantiated, so that the copy hung on FRED. The IN link is here functioning as a virtual copy link. This depends on the fact that during index search "short circuits" are established, on the one hand between FRED and the universals it gets bound to, and on the other hand between existentials dependent on those universals and objects related to FRED. This problem of copying and instantiating large sets of facts with local variables (like y and z) is rather important. Techniques like Fahlman's (cf. also Moore 1975, McDermott 1975, and Charniak 1977) are useful in solving this problem, whenever such a large set occurs on the right-hand side of a simple implication like

```
FORALL x (x IN S -> (EXISTS (y1 y2 ...) (....)))
```

Another technique for attacking this problem is "sequential indexing." For example, in the SAM system (Cullingford 1978), the sets of facts, called scripts, are of the form

```
FORALL x (x is a vehicle accident ->
           (EXISTS (driver, vehicle, obstacle,
                    ambulance, ...)
             (driver drives vehicle)
            then
             (vehicle hits obstacle)
            then
             (driver hurt)
            then
             ((ambulance comes) then ...)
            or
             ((driver gets worse) then ...))
```

In this case, the standard inference problem is, given a vehicle accident and some sequence of events, to identify them with some subsequence of events in this set (at the same time, identifying mentioned objects with the predicted script existentials). In such an application, the events to be processed come in in the same order as their script counterparts, and a linear search through the list is acceptably efficient. (Bobrow and Winograd 1977 argue for the central importance of this kind of inference problem, "matching" one set of facts against another, where the facts are structured to make straightforward search possible.)

This concludes my survey of the latest indexing and searching techniques It may seem somewhat spotty, but our knowledge is quite spotty in this area. To summarize, here are the known techniques--

> Resolution (or backward and forward chaining): for deductions with non-recursive rules.

> Indexing (or "object-centered representation"): for efficient retrieval of single expressions.

> Short-circuit indexing (or property inheritance): for random-access retrieval from an instance ("virtual copy") of a large conjunction of formulas, when no backward chaining is required.

> Sequential indexing: for efficient retrieval from a set in applications where the set has a natural order.

II.B Semantics

Now let's look at a slightly higher level, at efforts to extend traditional semantics. The most important work in this area, in my opinion, is the work of Sussman, Doyle, and their colleagues on non-monotonic logic. (de Kleer et. al. 1977, Doyle 1977. See also McCarthy 1977.) A non-monotonic logic is one that allows provisional inferences that may be wiped out by more complete information. For example, we will want to assume tht a professor has a Ph.D., because most do, and suspending judgment in each case is expensive. All AI programs do this kind of inference, and everyone realizes how important "default" assumptions are. (Minsky 1975) However, until recently no one had worked on some important questions: (1) What is the class of legitimate non-monotonic inferences? (2) How do you actually arrange things so that learning more complete information causes obsolete inferences to be undone? The first of these questions is semantic, the second implementational; more work has been done on the second, but some thought has been given to the first. (Kramosil 1975, McCarthy 1977, McDermott and Doyle, in progress)

The algorithms studied by Sussman and Doyle are called non-monotonic data-dependency systems. The inference maker keeps notes telling what each inferred datum was inferred from, plus the data whose absence was used in the inference. (For example, we might have "Mary is a Ph.D." supported by the presence of "Mary is a professor," and the absence of "Mary doesn't have a Ph.D.") Every time the status of a datum changes, the statuses of the data that depend on it must also be changed. In practice, it is possible to isolate a small set of data whose statuses could possibly change, and analyze this set in linear time. (Doyle 1977) So dependency analysis is efficient enough.

The interesting problems in this area for both theory and practice involve circularities in dependencies. If P can be proved from the unprovability of Q, and Q from the unprovability of P, is either provable? (The answer is, "No, but you may want to believe one of them for the time being.")

Another good piece of work on extending semantics is Moore's recent (1977, 1978) work on the semantics of the expression, "person A knows proposition P." His theory borrows from the work of Hintikka (1971) the definition, "P is true in all possible worlds compatible with what A knows." Moore's contribution is to encode this semantic

definition in a first-order language, for which the original knowledge statements become objects to be manipulated. That is, we introduce the relation K(person, w1, w2) to mean, "w2 is a possible world as far as person knows in w1," and give the axiom, "A knows P in w1 iff P is true in all worlds w2 such that K(A, w1, w2)."

This approach has two advantages. It provides an elegant framework for representing difficult statements like, "Fred doesn't know the combination to Mary's safe," by subsuming the scoping of acquaintance under the quantifier scoping supplied by Hintikka's definition; this statement would get expressed as, "There exists no object such that, in all worlds compatible with what Fred knows, that object is the combination to Mary's safe." Second, by calling for manipulations in the first-order calculus, the approach allows us to hope that standard implementation techniques will be applicable.

II.C New Notations

There hasn't been so much work on notations other than traditional propositional constructs. One example is the work of Funt (1977a,b) on a diagrammatic notation. In this system, two-dimensional views of scenes are represented by sets of marks in a two-dimensinal array. A connected blob of X's, for instance, means, "Object X has the same shape as this blob, and is located at the real-world position corresponding to the blob's location."

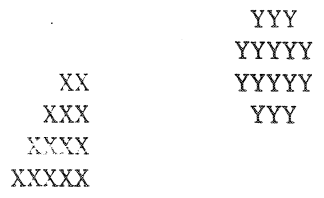


Figure 4 Object X and Object Y

The system makes inferences by performing operations on mark groups. For example, to infer that X will hit Y if it moves east, the system moves X's east one step at a time, looking for a collision. Funt discusses several kinds of inference. See also Kosslyn and Schwartz 1977.

II.D Managing Change in a Data Base

The whole purpose of using a knowledge representation system is to allow change to a program's knowledge by just adding a new fact. Most researchers take this for granted, and say little about the difficult problems of acquiring and "debugging" facts. One notable exception is Davis's (1976, 1977) work on assimilating new rules into MYCIN. His system translates rules acquired from an expert, and tests them by seeing if they actually solve the problems the expert intended them to solve.

There has been work on the problem of rule induction, automatic generalization and specification of rules to cover a training set. See especially Mitchell 1977.

III Applications

I have used most of my space to talk about general techniques. This means my overview of applications can be little more than a catalogue, each entry of which cannot do justice to the amount of work some researcher has invested in trying to capture some real-life set of facts. This catalogue is not intended to be a survey of all "knowledge-based" programs (since, as I argued, all programs are knowledge based), but only those that use a representation system as described in Part I of this paper. Nor will I be concerned with entire programs, but only with their representation systems. All I can do in the space allotted is summarize what class of facts and inferences each researcher has attacked, what techniques he used, and how completely he has solved the original problem. A complete solution is one which specifies a vocabulary capable of capturing all necessary nuances, and an implementation that can handle data bases of realistic size.

Seemingly the largest number of researchers has explored domains involving everyday causality and planning. (The intended application is often "story understanding," although the stories are not often real.) The researchers include Wilensky (1978), Cullingford (1978), Rieger (1976), Charniak (1977), McDermott (1978), and Sacerdoti (1977).

For example, Wilensky (1978), drawing on the work of Schank and Abelson (1977), has constructed notations for expressing such facts as, "Mary has the goal of obtaining money," and "If someone acquires a weapon, he may want to use it to threaten someone." The system, called PAM, speculatively infers what goals and plans a character might have from statements obtained from a story text. The main technique used is discrimination-tree indexing of rules used for forward chaining.

Cullingford's (1978) SAM program infers facts about characters' actions, given that they are involved in a stereotypical situation like riding on the subway or being involved in an auto accident. The notation, based on Schank's (1972) "conceptual dependency," expresses facts like, "Fred's car hit the tree," and "In an accident, after the vehicle hits an obstacle, an ambulance comes." The system infers missing pieces of such causal and temporal chains The main technique used is sequential indexing. (See above, Section II.A.)

Rieger and his students (Rieger 1976, Rieger and Grinberg 1978) have explored the use of a notation called "Common-Sense Algorithms" for expressing facts about causality. The class of facts they have explored include things like, "While the switch is closed, electricity can flow through the wire." The class of inferences they have looked at so far are inferences of temporal sequences of states of devices, given initial states. (They hope to expand this class.) The have used entry-time resolution rules, indexed in a discrimination tree.

Charniak (1977) has studied a notation for expressing facts about causality and planning, such as, "Mary painted a chair," "After painting something, you wash your brush," and "Not washing the brush would cause the brush to become unabsorbent." The system infers missing steps (as Cullingford's does), and also infers consequences of omitting steps. Facts about causation and planning come in sets; when a causal pattern or plan is referred to, an instance of its corresponding set become active. An active instance behaves like a virtual copy (in Fahlman's sense) of the set's contents.

McDermott (1978) has examined the domain of facts a problem solver needs to express about its own state, like, "I have the goal of coupling two electronic circuits," and "When I am planning to bias circuits and couple them, I should do coupling before biasing." This system uses resolution with simple inversion indexing to infer plans for carrying out tasks.

Sacerdoti's (1977) SOUP system expressed similar knowledge of plans for his NOAH problem solver. The facts he could express were of the form, "To put one block on top of another, clear both blocks, then move one to the other." The system inferred plan steps using sequential indexing; it marched through a SOUP plan, instantiating steps.

The main criticism to make of all these researchers is their inattention to questions of completeness. The place where they are weakest is in representing time. They all claim to represent facts about causality and planning, but they don't seem to extend robustly beyond a small range of inferences in this area. For example, none of them can even say, "This device (or plan) will loop forever if something isn't done to stop it," let alone infer it efficiently. Many of their systems have never dealt with data bases of reasonable size.

Many other domains have been studied. There is the classic work by Shortliffe (1976) on antibiotic therapy. It uses an unusual form of backward chaining (see above) to infer facts of the form, "It is known with certainty factor 0.8 that the identity of organism-1 is pseudomonas." It appears to be a fairly complete notation for its domain. (The description of its limitations is clearer than for most systems.)

David Barstow's (1977a,b) PECOS program contains a representation for facts about refining abstract programs to LISP code, facts like, "A collection may be represented as a list," and "The typical element of collection-1 is a symbol." The representation system mainly retrieves rules for use by the rest of the system, but it does some simple backward chaining. The system is fairly complete, if for a somewhat obscure domain of facts.

Sussman and his colleagues (Stallman and Sussman 1977, de Kleer et. al. 1977) are developing the EL system for expressing and inferring facts about electronic circuits. A typical fact is, "The current into a resistor equals the current out." The system infers things like voltage values, given resistances and other voltage values. It uses discrimination-tree indexing, entry- and retrieval-time resolution, and, most interesting, non-monotonic data dependencies. It appears to be very complete for expressing facts about steady-state

device states; it doesn't handle time or causality.

Kuipers (1977a,b) has written a program to perform inferences about space. His implementation allows the expression of facts like, "Central Square is between Harvard Square and MIT," and "Inman Square is ninety degrees away from Harvard Square, with respect to Central Square." The main technique used is simple inversion indexing, and sequential (here, spatial) arrangements of lists of expressions. The inference rules are themselves not represented, but are part of the LISP code making up the data base manager. It is hard to say how close this work comes to being a complete solution to the problem; this is a good place for further work.

There are many other programs, either new or improved, which use representation systems. For example, Novak (1977) and de Kleer (1977) have written physics problem solvers which retrieve "virtual copies" of chunks of knowledge like equations known to be useful in familiar situations. The PROSPECTOR system of Hart and Duda (1977) makes speculative inferences about mineral deposits, using forward- and backward-chaining rules indexed by Hendrix's (1975) version of short-circuit indexing. Lenat's (1977a, b) AM system discovers mathematical concepts and conjectures, by a process of iterated speculative inference; he used inversion indexing and property inheritance on a notation containing a small vocabulary of predicates ("facets") and a growing collection of concepts. The Meta-DENDRAL project (Buchanan and Mitchell 1978) is studying automatic induction of rules about what happens when complex molecules break apart.

IV Conclusions

Work on knowledge representation continues. Some interesting techniques have been discovered; some domains have been solved to some degree.

I would like to make two sorts of recommendations for future work in this area. First, I think there are certain problems that need more attention than they are getting. We need a good theory of time and causality, and an efficient implementation of it. (McCarthy 1977) Research in this area can go on only so long making intuitive classifications of types of causation. We also need a theory of speculative inference, especially of explanation. What are people looking for when they seek an explanation? When is an explanation plausible enough to be assumed? When a contradiction occurs in the data base, how hard is it to eliminate it (i.e., find the best set of changes that makes the data base consistent)?

Second, I would like to recommend a terminological reform. The phrase "representation of knowledge" is vague and misleading. (For some very different views of "knowledge representation," see Bobrow and Winograd 1977, Hawkinson 1975, Forgy and McDermott 1977, Hewitt 1975, and Schank and Abelson 1977.) To many people inside and outside AI it connotes a search for a "philosopher's stone" -- a system for representing anything. It contains a word, "knowledge," which is there only as a conjuring device. (What else can you represent but "knowledge"?) Students in AI are trained by phrases like this into an

attitude of cynicism toward the whole field. I propose, therefore, that from now on we call it "notational engineering," and define it as the study of mechanically (formally) manipulable notations for objects and facts. This study includes classical concerns with syntax and semantics, but also takes into account efficiency and completeness of inference algorithms. (Arabic numerals were the first major discovery of notational engineering.) If we concentrate on these issues, we can leave for later judgment whether what gets represented is knowledge wisdom, or lies.

Acknowledgments: I thank David Barstow, Chris Riesbeck, and Bob Wilensky for helpful comments.

Bibliography

David Barstow 1977a A knowledge-based system for automatic program construction. Proc. IJCAI 5, p. 382.

David Barstow 1977b Automatic construction of algorithms and data structures using a knowledge base of programming rules. Stanford AI Laboratory Memo 308.

Daniel G. Bobrow and Allan Collins (eds.) 1975 Representation and Understanding. New York: Academic Press.

Daniel G. Bobrow and B. Raphael 1974 New programming languages for artificial intelligence research. Computing Surveys 6, no. 3.

Daniel G. Bobrow and Terry Winograd 1977 An overview of KRL-0, a knowledge representation language. Cognitive Science 1, no. 1.

Robert J. Bobrow and John Seely Brown 1975 Synthesis, analysis, and contingent knowledge in specialized understanding systems. In Bobrow and Collins 1975

Ronald J. Brachman 1977 What's in a concept: structural foundations for semantic networks. Int. J Man-Machine Studies 9, no. 2.

Bruce G. Buchanan and Tom M. Mitchell 1978 Model-directed learning of production rules. In D.A. Waterman and F. Hayes-Roth (eds.) Pattern-Directed Inference Systems. New York: Academic Press.

Eugene Charniak 1977 A framed PAINTING: the representation of a common sense knowledge fragment. Cognitive Science 1, no. 4.

Richard Cullingford 1978 Script application: computer understanding of newspaper stories. New Haven: Yale Computer Science Research Report 116.

Randall Davis 1976 Application of meta-level knowledge to the construction, maintenance, and use of large knowledge bases. Stanford: Stanford University Heuristic Programming Project Memo 76-7.

Randall Davis 1977 Interactive transfer of expertise: acquisition of

new inference rules. Proc. IJCAI 5.

Johan de Kleer 1977 Multiple representations of knowledge in a mechanics problem-solver. Proc. IJCAI 5

John de Kleer, Jon Doyle, Guy L. Steele, and Gerald J. Sussman 1977 AMORD: explicit control of reasoning. Proc. Symposium on AI and Programming Languages. (SIGART Newsletter 64).

Jon Doyle 1977 Truth maintenance systems for problem solving. Cambridge: MIT AI Laboratory Technical Report 419.

George W. Ernst and Allen Newell 1969 GPS: A Case Study in Generality and Problem Solving. New York: Academic Press.

Scott Fahlman 1978 A system for representing and using real-world knowledge. Cambridge: MIT AI Laboratory Technical Report 450.

Richard Fikes and Gary Hendrix 1977 A network-based knowledge representation and its natural deduction system. Proc. IJCAI 5.

C. Forgy and John McDermott 1977 OPS, a domain-independent production system language Proc. IJCAI 5.

Brian V. Funt 1977a WHISPER: a problem-solving system utilizing diagrams and a parallel processing retina. Proc. IJCAI 5.

Brian V. Funt 1977b WHISPER: a computer implementation using analogues in reasoning. Vancouver: UBC Computer Science Technical Report 76-9.

Peter E. Hart and Richard O. Duda 1977 PROSPECTOR -- a computer-based consultation system for mineral exploration. Menlo Park: SRI International AI Center Technical Note 155.

Lowell Hawkinson 1975 The representation of concepts in OWL. Proc. IJCAI 4.

Philip J. Hayes 1977 On semantic nets, frames, and associations. Proc. IJCAI 5.

Gary Hendrix 1975 Expanding the utility of semantic networks through partitioning. Proc. IJCAI 4.

Carl Hewitt 1975 How to use what you know. Proc IJCAI 4.

J. Hintikka 1971 Semantics for propositional attitudes. In L. Linsky (ed.) Reference and Modality. London: Oxford University Press.

Stephen M. Kosslyn and Steven P. Schwartz 1977 A simulation of visual imagery. Cognitive Science 1, no. 3.

Ivan Kramosil 1975 A note on deduction rules with negative premises. Proc. IJCAI 4.

Benjamin J. Kuipers 1977a Representing knowledge of large-scale space.

Cambridge: MIT AI Laboratory Technical Report 418.

Benjamin J. Kuipers 1977b Modelling spatial knowledge. Proc. IJCAI 5.

Douglas B. Lenat 1976 AM: an artificial intelligence approach to discovery in mathematics as heuristic search. Stanford: AI Laboratory Memo 286.

Douglas B. Lenat 1977 Automated theory formation in mathematics. Proc. IJCAI 5.

John McCarthy 1977 Epistemological problems of artificial intelligence. Proc. IJCAI 5.

Drew V. McDermott 1975. Very large PLANNER-type data bases. Cambridge: MIT AI Laboratory Memo 339.

Drew V. McDermott 1977 Flexibility and efficiency in a computer program for designing circuits. Cambridge: MIT AI Laboratory Technical Report 402.

Drew V. McDermott 1978 Planning and Acting. Cognitive Science 2, forthcoming.

Marvin Minsky 1975 A framework for representing knowledge. In Patrick H. Winston (ed.) The Psychology of Computer Vision. New York: McGraw-Hill.

T.M. Mitchell 1977 Version spaces: a candidate elimination approach to rule learning. Proc. IJCAI 5.

Robert C. Moore 1975 Reasoning from incomplete knowledge in a procedural deductive system. Cambridge: MIT AI Laboratory Technical Report 347.

Robert C. Moore 1977 Reasoning about knowledge and action. Proc. IJCAI 5..

Robert C. Moore 1978 Reasoning about knowledge and action. Forthcoming MIT Ph.D. Thesis.

Gordon S. Novak 1977 Representations of knowledge in a program for solving physics problems. Proc. IJCAI 5.

Harry Pople 1973 On the mechanization of abductive logic. Proc. IJCAI 3.

Chuck Rieger 1976 An organization of knowledge for problem solving and language comprehension. Artificial Intelligence 7, no. 2.

Chuck Rieger 1977 Spontaneous computation in cognitive models. Cognitive Science 1, no. 3.

Chuck Rieger and Milt Grinberg 1978 A system of cause-effect representation and simulation for computer-aided design. Proc. IFIPS

Workshop on AI and Pattern Recognition in CAD. Grenoble.

J.A. Robinson 1965 A machine-oriented logic based on the resolution principle. J. ACM 12.

Sacerdoti Earl D. 1977. A Structure for Plans and Behavior. New York: American Elsevier Publishing Company, Inc.

Roger Schank (ed.) 1975 Conceptual Information Processing. Amsterdam: North-Holland Publishing Company.

Roger Schank and Robert Abelson 1977 Scripts, Plans, Goals, and Understanding. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Stuart Shapiro 1977a Representing and locating deduction rules in a semantic network. Proc. Workshop on Pattern-Directed Inference Systems, Hawaii. (SIGART Newsletter 63)

Stuart Shapiro 1977b Compiling deduction rules from a semantic network into a set of processes. Proc. Workshop on Automatic Deduction. MIT.

E.H. Shortliffe 1976 Computer-Based Medical Consultations: MYCIN. New York: American Elsevier Publishing Company, Inc.

Richard Stallman and Gerald J. Sussman 1977 Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. Artificial Intelligence 9, no. 2.

Gerald J. Sussman 1975 A Computer Model of Skill Acquisition. New York: American Elsevier Publishing Company.

David H D. Warren and Lous M. Pereira 1977 PROLOG -- the language and its implementaion compared with LISP. Proc. Symposium on AI and Programming Languages, Rochester.

Robert Wilensky 1978 Machine understanding of human intentionality. Yale Computer Science Department, forthcoming Ph.D. thesis.

BEST-FIRST INTERPRETATION OF IMPERFECT OBJECT CONTOURS

Bernd Neumann
Fachbereich Informatik, Universitaet Hamburg
Schlueterstr. 70, D-2 Hamburg 13

0. Abstract

A scene-analysis system is presented which identifies objects by means of straight-line contour approximations. Descriptions of prototypes are stored in a relational-model database. Scenes may be disturbed and may contain objects which are partially occluded by another object. A segmentation algorithm extracts straight edges by multidirectional sampling of the picture gradient matrix. Interpretation hypotheses are generated using 2-step relational matching. A best-first strategy selects the final interpretation using constraint filtering for ambiguous situations. Results are presented for scenes containing configurations of artificial shapes (card-board pieces) and real-life objects (tools). They indicate that in systems designed for application one should sophisticate interpretation rather than segmentation.

Keywords: scene analysis, interpretation, contour approximation, occlusion, constraint filtering.

1. Introduction

In scene analysis much attention has been paid to the interpretation process over the last few years. While most of the early recognition programs were essentially one-level with knowledge contained in templates and heuristics, it was realized that at various levels of abstraction various sorts of knowledge have to be applied in order to extract meaning from real-life scenes [1]. Thus new techniques had to be developed to represent knowledge and organize multilevel interpretation.

Relational data structures proved to be particularly useful for describing objects and their interrelations [2]. A hierarchical organization may permit effective matching of uninterpreted scene data to relational models describing known objects or object configurations [3].

Recently a different view point has gained increasing acceptance: Knowledge is introduced into the interpretation process via constraints. A constraint may be employed as a filter which eliminates hypotheses violating that constraint. This technique attributed to Waltz [10] has been successfully applied to segmentation [7,12] as well as higher-level processing [8,12].

Confidences in assigning interpretations may be used as an additional means of guiding the interpretation process. Yakimovsky and Feldman [11] employ Bayesian methods for choosing between region interpretations. While a strict probabilistic model may not always be applicable the general idea of following a path of most likely hypotheses has been widely accepted, see for example [9].

This contribution presents a scene analysis system which uses two-level relational matching, confidence guidance and constraint filtering to interpret imperfect views of object configurations. The system was designed with the intention that neither the class of admissable scenes nor the amount of processing required for analysis would preclude any application. Following this guideline scenes have been selected where objects may be recognized by shape, i.e. by the

edges within and around an object view, neglecting gray value or texture information. Past experiences throughout the scene analysis community suggest that low-level processing of real scenes will only give low-level results [4], that is, will not provide a segmentation into meaningful units. Thus this system is explicitly designed to analyze imperfect segmentations putting the burden on high-level rather than low-level processing. This has many advantages. First, the system is expected to show graceful degradation, since the interpretation algorithm is prepared for segmentation faults (and will not produce random results). Secondly, using simple segmentation techniques on the account of a sophisticated interpretation strategy tends to reduce the overall computational expenses because of the reduced data volume in high-level processing. Thirdly, the system may be capable to handle a considerable amount of systematic degradation of the segmentation results, e.g. due to partial occlusion of one object by another or perspective distortion of object views.

Section 2 contains a brief description of the straight-edge finder which is used for segmentation. The algorithm (which appears to be new) uses exhaustive search on a gradient matrix along 8 different directions.

Object models are also straight-line contour approximations with some angle information extracted for matching. They are stored in a relational data base. A particular model serves to recognize all objects whose contours match after proper translation, rotation and dilatation. The model structure is discussed in section 3.

Section 4 describes the interpretation algorithm. Two confidence thresholds are used to distinguish between hypotheses which are safe, acceptable but unsafe, and not acceptable. While safe hypotheses are final interpretations, the medium category will typically contain candidates where an immediate decision may be erroneous. After generating all hypotheses these candidates are reconsidered in the order of decreasing confidence, accepting those which still pass the lower threshold while reducing the confidence of conflicting hypotheses.

Section 5, finally, presents results of 2 scenes containing configurations of plane shapes and 2 scenes with assorted tools. Input data have been recorded by an ordinary TV-camera.

2. Segmentation

The segmentation method used in this system extracts edges as a basis for subsequent interpretation. Since edges are expected to form neither accurate nor complete object descriptions the algorithm generates only straight-line approximations and does not attempt to link edges to closed contours.

In the first step the original 574 x 512 gray value picture (8 bits per pixel) is transformed into a 191 x 256 gradient matrix with directional information coded by 4 bits and magnitude by 1 bit after thresholding. In the second step consecutive edge elements whose directions differ by no more than one quantization unit (22.5 degree) are combined to "strokes". This is achieved by passing over the gradient matrix 8 times sampling the edge elements along 8 different directions. Since real edges may be curved and are not confined to 8 directions, strokes of neighboring sampling lines tend to overlap. They are merged if they constitute a sufficiently straight edge, otherwise they stay single. Pictures 3 - 6 show original scenes and their segmentations. Note the circular shape in Picture 3a and its approximation by tangential edge pieces. Since stroke finding and merging is carried out independently for 8 directions, the task is highly suited for parallel processing.

3. Models

Knowledge about object shapes is contained in a model data base. Each object model is defined by a model edge relation MER which links particular edges to object names. Each edge is represented by an edge identifier and by starting and end point coordinates. Edges are directed implying that the darker area is to the right. A second relation MAR links pairs of consecutive edges to the enclosed angle. This does not add new information but provides clues for matching. Formally the edge relation is

MER = [< o e <x1 y1> <x2 y2> >]

and the angle relation is

MAR = [< a e1 e2 >]

where the symbols denote:

o	object name
e,e1,e2	edge identifiers
a	angle
x1,x2	x-coordinates in gradient matrix
y1,y2	y-coordinates in gradient matrix

Notable for their absence in this model definition are other descriptive features like elongatedness, symmetry information, number of corners, etc. They are not expected to be very useful for identification because of their sensitivity to imperfect segmentation.

A model serves to identify all objects whose shapes are similar in the geometric sense. The edges of a model object describe one view from which all other views must be computed by translation, plane rotation and dilatation. A set of scene edges is identified as object o if there is a similarity transformation such that the transformed model edges of o match the scene edges. Of course, the match need not be exact. A confidence function evaluates the degree of exactitude permitting many scene edges to account for one model edge and vice versa. Essentially, the object confidence is defined to be the quotient of matching edge length to total edge length with a malus assigned to completely missing edges. An object model may be generated by simply viewing and segmenting a prototype. This should be done under favourable conditions to minimize postediting.

4. Interpretation

After segmentation meaning must be assigned to extracted edges. More precisely, edges must be associated with an object name o and the translation, rotation and dilatation parameters t,r,d of the corresponding model transformation. A quadruple <o t r d> is termed hypothesis. An interpretation is the relation

I = [< e o t r d >]

linking scene edges to hypotheses. While a single hypothesis may naturally receive a confidence value from matching, there is no straight-forward way to select those hypotheses which as a whole constitute the interpretation with highest confidence. This problem has been generally recognized. One approach is to introduce additional knowledge in the form of constraints and use relaxation techniques to find the globally best interpretation [8]. The approach of this system is different in that instead of relaxation a fast best-first method is used. As a constraint it is required that each edge may be assigned only one meaning, i.e. may only be part of a single object.

It is assumed that some objects will be recognizable beyond all doubt and will survive conflict testing. The corresponding hypotheses are called safe and are taken to be those with a confidence value above a certain upper threshold. Immediately upon discovery they are accepted and all associated scene edges are removed from further consideration. Ambiguous configurations, however, are assumed to give rise to hypotheses with confidence values lower than the upper threshold but

still high enough to pass a lower threshold which serves as a rejection level. These hypotheses are called unsafe while all the rest is unacceptable.

Unsafe hypotheses are collected and ordered according to confidences. They are expected to offer alternative interpretations for certain edges. The list of all unsafe hypotheses is repeatedly subjected to the following 3-step procedure until no acceptable hypotheses remain.

(i) Check for conflict between first candidate and any of the rest. Assign conflicting edges to first candidate and degrade confidence of other candidate accordingly.
(ii) Remove first candidate from list and include it in interpretation.
(iii) Reorder list according to confidences.

This method may be termed iterative best-first acceptance. It tends to produce an interpretation which maximizes the sum of the selected object confidences. A formal analysis of the exact behaviour of this algorithm has not been attempted up to the time of this writing.

The following will give some indication of how hypotheses are generated. First, the segmentation results are stored in a relational form identical to the model structure except that no object names are attached to edges. In particular, all edges are entered into the scene edge relation SER and those who form a pair with head and tail sufficiently close are entered into the scene angle relation SAR. Now the tuples of SAR are considered one after the other for a possible angle match with any of the tuples of MAR. Matching pairs define object name o and rotation angle r of a hypothesis. Translation t and dilatation d are calculated on the basis of the matching pairs and a third scene edge which is chosen to match with one of the remaining edges of the object model. At this point the two-level bottom-up procedure reverses and a top-down match is performed by transforming all model edges according to the parameters t,r,d. A comparison with the available scene edges yields the confidence value of the hypothesis <o t r d>.

5. Results

The system has analyzed many scenes. Results of 4 typical scenes are shown in Pictures 3-6. Scene 1 (3a) is a configuration of card-board pieces with a modest amount of occlusion. About 60 edges were extracted (3b) and about as many pairs were found for the angle relation SAR. The interpretation result is depicted in (3c) showing those scene edges (or parts thereof) which were found to agree with a model. All object interpretations resulted from safe hypotheses except POT(2) and TRIANGLE(2). Computation time was 60 sec for segmentation and 15 sec for interpretation on a DECsystem-10 with KI-processor. All programs were written in SAIL.

Scene 2 (4a) shows a configuration with more difficult occlusion. Note that many edges are not visible at all and edges of separate objects are merged. All objects were found except one of the crossing rectangles. A proper hypothesis has been generated but did not survive since the other rectangle deprived it of the lower edge. Only SQUARE(1), SQUARE(2) and TRIANGLE(1) were safe hypotheses. A large number of candidates was generated resulting in an interpretation time of 32 sec.

Scenes 3 and 4 (5a,6a) show some tools viewed from above. Due to reflection, texture and shadows many noise edges were generated, especially in scene 4. All objects were found correctly except

NEEDLE(1) in (6c) which was discovered within the noise edges at the bottom of the scene. Interpretation time was 15 sec and 52 sec, respectively.

A comparison with the work of Perkins [6] indicates that similar results may be obtained using a different approach. Perkins analyzed a scene with 5 overlapping objects (machine parts) in 15 sec on a IBM 370/168 using curved contour pieces as descriptive features.

In conclusion, the system presented in this paper shows considerable immunity against segmentation imperfections and yet obtains results in reasonable time. The performance is largely due to a careful design of the interpretation process. It is suggested that a combination of simple segmentation with sophisticated interpretation may be useful elsewhere.

Hardware and software supporting this work have resulted from the combined efforts of R.Bertelsmeier, P.Cord, I.Heer, H.Kemen, H.-H.Nagel, B.Neumann and B.Radig.

6. References

[1] H.G.Barrow, J.M.Tenenbaum, "Representation and Use of Knowledge in Vision", TN 108, SRI, Stanford, 1975

[2] H.G.Barrow, R.J.Popplestone, "Relational Descriptions in Picture Processing", in: Machine Intelligence VI (Meltzer/Michie, Eds.), p.377, 1971

[3] H.G.Barrow et al., "Some Techniques for Recognizing Structures in Pictures", in: Frontiers of Pattern Recognition (Watanabe, ed.), p. 1, 1972

[4] D.Marr, "On the Purpose of Low-Level Vision", AI-Memo 324, MIT, Cambridge, 1974

[5] B.Neumann, "Identifikation von gestoerten Objektansichten unter Verwendung geradliniger Konturapproximationen", FBI-HH-B-42/78, Fachbereich Informatik, Universitaet Hamburg, 1978

[6] W.A.Perkins, "A Model-Based Vision System for Scenes Containing Multiple Parts", Proc. IJCAI-77, p. 678, 1977

[7] B.Radig, "Auswertung von digitisierten Fernsehbildern zur Beschreibung bewegter Objekte", in preparation, Fachbereich Informatik, Universitaet Hamburg, 1978

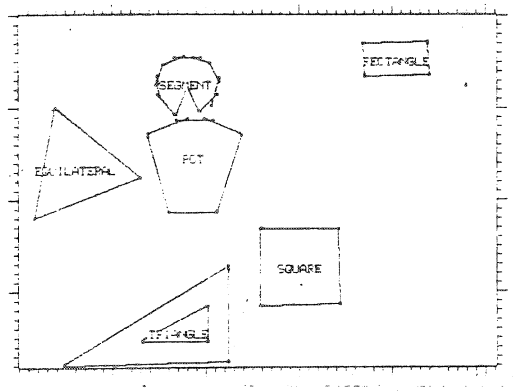
[8] A.Rosenfeld et al., "Scene Labelling by Relaxation Operations", IEEE Trans. Systems, Man and Cybernetics, SMC-6, p.420, 1976

[9] S.Rubin, R.Reddy, "The LOCUS Model of Search and its Use in Image Interpretation", Proc. IJCAI-77, p.590, 1977

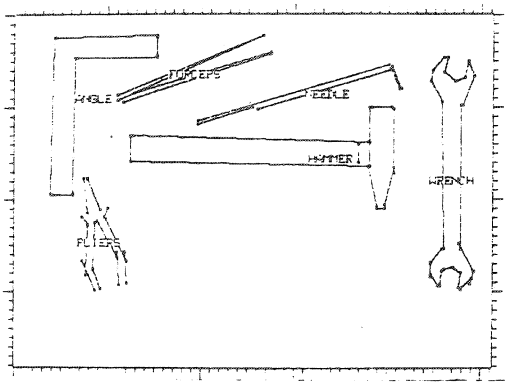
[10] D.Waltz, "Understanding Line Drawings of Scenes with Shadows", in: The Psychology of Computer Vision (Winston, ed.), McGraw-Hill, 1975

[11] Y.Yakimovsky, J.A.Feldman, "A Semantic-Based Decision Theory Region Analyser", Proc. IJCAI-73, p.580, 1973

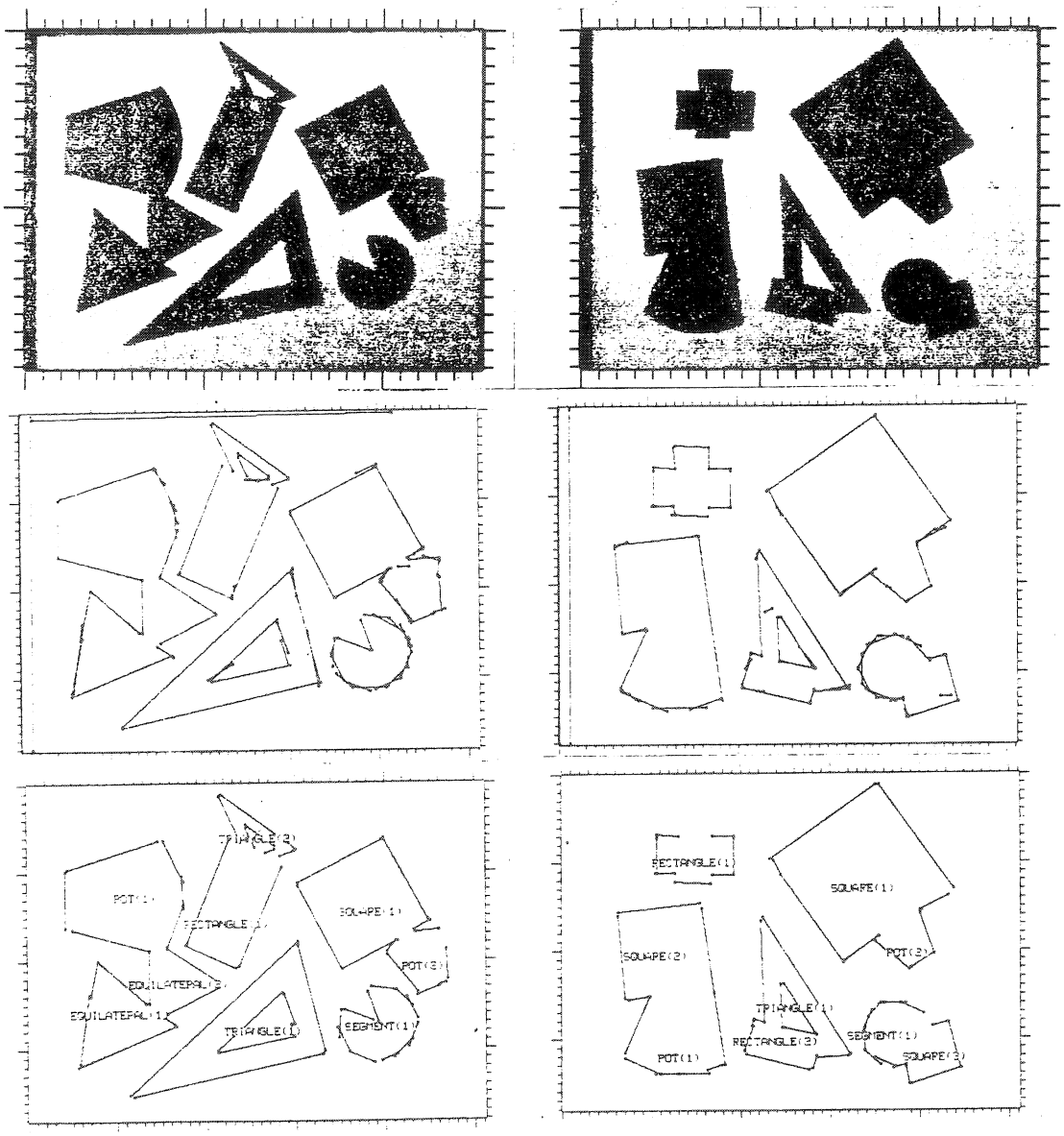
[12] S.W.Zucker, "Relaxation Labelling, Local Ambiguity, and Low-Level Vision", in: C.H.Chen (Ed.), "Pattern Recognition and Artificial Intelligence", Academic Press, N.Y., p.593, 1976



Picture 1: Models for scene 1 and 2

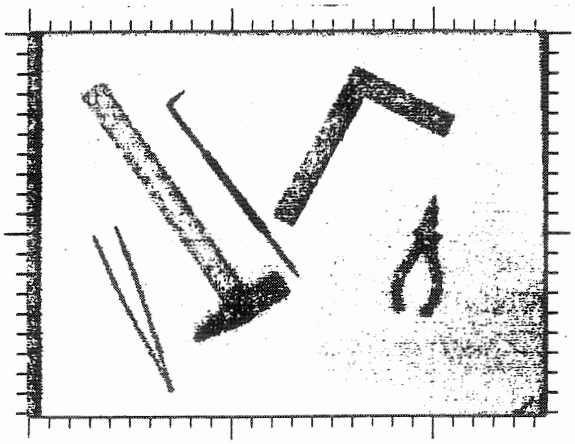


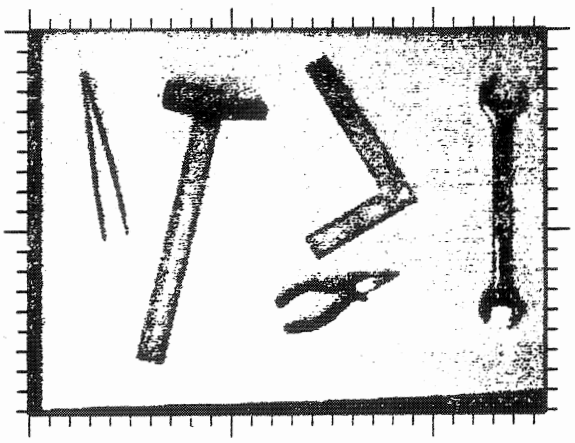
Picture 2: Models for scene 3 and 4

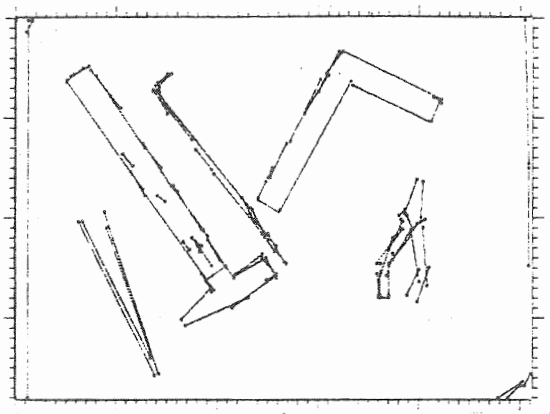


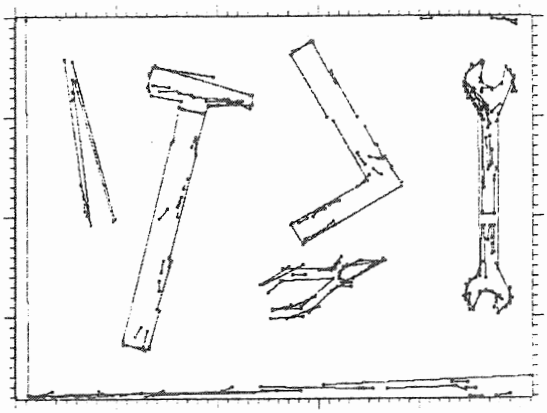
Picture 3: a) Scene 1
b) Segmentation
c) Interpretation

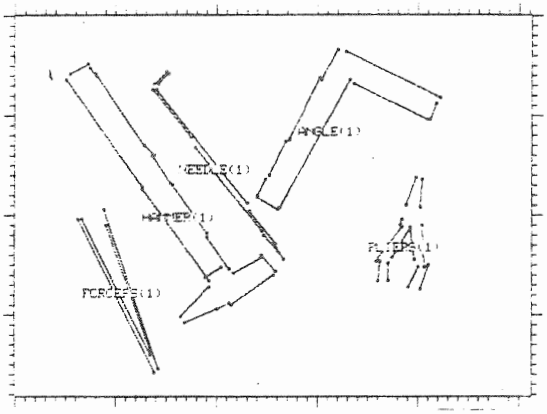
Picture 4: a) Scene 2
b) Segmentation
c) Interpretation

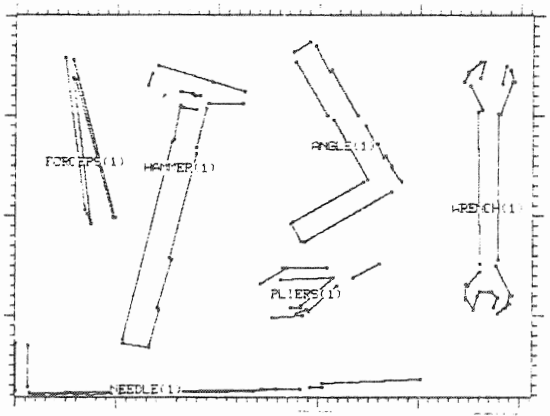












Picture 5: a) Scene 3
b) Segmentation
c) Interpretation

Picture 6: a) Scene 4
b) Segmentation
c) Interpretation

A PRODUCTION RULE ACCOUNT OF ERRORS IN CHILDREN'S SUBTRACTION

Tim O'Shea and Richard M. Young
Department of Artificial Intelligence
University of Edinburgh
Edinburgh Scotland

ABSTRACT

Many of the errors that occur in children's subtraction are due to the use of incorrect strategies rather than to the incorrect recall of number facts. A production system is presented for carrying out two-column subtraction using the method of decomposition. We show that many of the incorrect strategies used by schoolchildren can be modelled as the consequences of simple changes to this production system, such as the omission of individual rules or the addition of rules appropriate for other arithmetical tasks.

Introduction

We present an analysis of various correct and faulty strategies for two-column subtraction. At one time children's errors in subtraction were largely attributed to failure to remember 'subtraction facts' correctly (e.g. 9 - 7 = 2). Other causes of error generally recognised (see Downes & Paling, 1958) included faulty setting out of examples, carelessness, fatigue and lack of concentration. Thyne (1954) in his careful and detailed taxonomy of errors in arithmetic clearly demonstrated that what was occurring was a failure in the *process of recalling or determining* the facts. Various models for recall (Sloman, 1973) and determination (Woods & Hartley, 1971; Woods, Resnick & Groen, 1975) of number facts have been advanced. But careful analysis of the errors in computation shown by individual pupils (see Lankford, 1977) forces the conclusion that many errors result from a failure in the *process of executing* the subtraction rather than from carelessness or defective number facts. These errors fall into quite clear patterns and contemporary workers in education attribute them to the use of 'erroneous algorithms' (Ashlock, 1976).

This paper is an attempt to present such erroneous algorithms in an explicit form. We will present a model for correct subtraction in the form of a set of rules. Different kinds of error will be accounted for by the omission of various rules from this set.

Methods for subtraction

Children are taught two main methods for carrying out multi-column subtraction (Williams, 1971). In both methods the numbers to be subtracted are written one beneath the other with the corresponding digits aligned:

e.g.	7	4	(← minuend)	M_2	M_1
	− 2	8	(← subtrahend)	− S_2	S_1
			(← difference)	D_2	D_1

and in both methods the columns are processed in order from right to left. They differ only in the techniques used for "borrowing", i.e. when the subtrahend digit (S) is greater than the minuend digit above it (M), (S > M). In either case, a "ten" is "borrowed" from the column to the left, thus in the example above $M_1 = 4$ is treated as 14. In the method of *decomposition* this ten is payed back by decrementing the digit in the minuend, thus $M_2 = 7$ becomes 6. In the *equal addition* method the paying back is done by incrementing the digit in the subtrahend: the $S_2 = 2$ would be treated as 3.

Errors in subtraction

Figure 1 shows eight examples of incorrect subtraction sums. These are taken from Bennett's (1976) analysis of the errors in 1360 subtraction sums done by 10 year olds. Similar examples are found in other analyses (e.g. Lankford, 1973).

Figure 1: Examples of subtraction errors

A	B	C	D
3 7	4 8	2 3 8	1 2 1 7
− 1 9	− 1 9	− 1 5	− 1 2
2 2	3 9	3	1 5

E	F	G	H
4 5 6	4 5 2	1 2 1 5	7 1 4
− 3 2 7	− 4 3 7	− 1 7	− 4 9
1 1	8 5	1 8	3 6

Example A is a very common type of error where the student does not borrow but just takes the difference of the two digits in each column. In example B the student has mentally added 10 to the minuend on the right but has not payed it back. In example C the student has borrowed when it was not appropriate (i.e. $M_1 > S_1$) then having taken the difference of 18 and 5 has dealt with the 'ten' in the result of 13 by carrying it into the second column. Carrying is of course not normally a step in subtraction, and its consequence is that the answer is 20 too low (since the second column has effectively been "borrowed from" twice, both times unnecessarily). In example D the student has again performed an inappropriate borrow and then carried. In this case the digits in the left-hand column differ by one (i.e. $M_2 = S_2 + 1$), so the subtrahend plus carry is now greater than the minuend minus borrow. Typically, as in this case, the student simply takes the smaller digit from the larger yielding a difference of '1' which is "correct" though for the wrong reason. Example E is problematical since parts of both the equal additions and the decomposition methods seem to have been used but as in example A the result in the right-hand column seems to have been derived by simply taking the difference of the two digits. In example F the method of paying back from equal additions and the method from decomposition seem both to have been applied and the resulting numbers in the tens column added. Example G is similar except that the difference between the resulting numbers in the tens column has been taken.

Example H includes a common type of subtraction fact error (14 - 9 = 6).

Our analysis will make no attempt to account for the common errors in number facts such as those that occur with 9s (result often wrong by ±1) or with 0s (0 - 7 = 0 or 7).

Production systems

Our analysis will make use of a form of information processing model known as a *production system*. A production system (PS) consists of a collection of rules (C —> A), whose left-hand sides consist of conditions C which specify when the right-hand side actions(s) A are meant to apply. PSs were introduced as psychological models by Newell and Simon (1972) who used them to model the behaviour of adults solving certain kinds of puzzles. Since then, they have been used as a medium for expressing theories of various topics in cognitive psychology (Anderson, 1976; Klahr & Wallace, 1976). The modularity of the individual rules makes PSs especially suitable for describing the learning of a skill, since its growth can be modelled by the acquisition of new rules (Young, 1976). Our previous work using PSs for subtraction showed too that they offer a natural way to model the way that subtraction behaviour often consists of a mixture of different strategies (Young, 1977)

Production system conventions

The production systems we use follow the conventions of the OPS PS architecture (Forgy & McDermott, 1977). The architecture has three parts:

(a) A working memory (WM), which is simply a set of elements such as (S EQ M) or (RESULT 5). The elements are regarded as being unordered, though they are implicitly tagged with the time at which they were asserted (i.e. added to the WM). The WM is a strict set: no element can appear more than once.

(b) A production memory (PM), which holds a collection of production rules. The left-hand side of each rule is a conjunction of patterns which have to match against distinct elements of WM. The right-hand side consists of a sequence of actions to be taken or items to be added to WM. Example are given below. The rules are regarded as being unordered, though in the case of a growing PS, OPS keeps note of the order in which the rules are added to the system.

(c) A *conflict resolution* method which specifies which rule is to be fired when more than one is applicable. OPS interprets a PS by repeated execution of a *recognise-act* cycle. The interpreter first finds which rules currently have their LHSs satisfied. These rules are potential candidates for firing, and form the *conflict set*. The conflict resolution method is applied to the conflict set to discard all but one of the rules in it. That one remaining rule is then "fired", i.e. its RHS is obeyed, and then the cycle repeats. OPS makes use of the following principles (for further justification, see McDermott & Forgy, 1978):

(i) Refractoriness: No rule is ever fired more than once matched to the same elements in WM.

(ii) Recency in WM: Rules whose LHSs match more recent elements

in WM knock out rules that match only older elements.
(iii) Special case: If rule R1 matches all the elements that R2 does, and more besides, then R1 is a *special case* of R2, and R2 is discarded.
(iv) Recency in PM: If after applying principles (i)-(iii) there is still more than one rule left in the conflict set, then OPS discards all but the one most recently added to the system.
(v) Arbitrary choice: If there is still more than one rule in the conflict set, then OPS simply picks one arbitrarily.

An example: PS for decomposition

As an illustration, Figure 2 shows a simple PS for carrying out the basics of two-column subtraction using the decomposition technique for borrowing.

Figure 2: Partial PS for subtraction by decomposition

```
TS   ((PROCESSCOLUMN) --> (TAKEDIFF))
B2A  ((S GTR M)       --> (BORROW))
BS1  ((BORROW)        --> (CROSSOUT))
BS2  ((BORROW)        --> (DECREMENT))
BS3  ((BORROW)        --> (ADDTENTOM))
CM   ((PROCESSCOLUMN) --> (COMPARE))
DTD  ((TAKEDIFF)      --> (DO TAKEDIFF))
```

This PS demonstrates the essentials of the subtraction process, essentials that are shared by the more complete PSs we will be offering later. The labels down the left-hand column are simply tags to allow us to refer to the individual rules, and play no part in their actual functioning. To consider the rules briefly in turn:

Rule TS says that an appropriate response to the goal of subtracting two (small) numbers is to set up the subgoal of applying the action of TakeDiff to them, i.e. the action of finding the absolute difference between them.

Rule B2A indicates that if the subtrahend digit is greater than the minuend digit - (S GTR M) is one of the possible outputs of the Compare operator, as we will see below - then the appropriate response is to perform a Borrow.

Rules BS1, BS2 and BS3 spell out the components of Borrowing. BS1 and BS2 are responsible for crossing-out and decrementing the digit borrowed from, either physically on paper or just "in the head", while BS3 adds the borrowed ten to the minuend digit.

Rule CM says that an appropriate behaviour for subtracting a column is to Compare its two digits. Compare deposits in WM an element indicating the relative sizes of the subtrahend and minuend digits, one of (S LESS M), (S GTR M) or (S EQ M). We assume that CM is a product of a child's training in multi-digit subtraction and is therefore acquired later than rule TS.

Finally, rule DTD is responsible for translating the subgoal of TakeDiff into action.

The easiest way to understand the function of this little PS is to trace its behaviour on an actual problem. Figure 3 summarises its

behaviour. We assume that a child is faced with the problem we saw earlier

```
  7  4
- 2  8
------
```

and that his attention is focussed on the first (RH) column. Initially we assume his WM to contain just one relevant item, the symbol (PROCESSCOLUMN) which acts as a goal (Newell & Simon, 1972) for the subtraction performance.

Figure 3: Firing order of rules in PS of Figure 2.

Step	Rules applicable	Rule fired	Action taken
0.			WM has (PROCESSCOLUMN)
1.	TS, CM	CM	Do Compare; Assert (S GTR M)
2.	(CM), TS, B2A	B2A	Assert (BORROW)
3.	(CM, B2A), TS, BS1, BS2, BS3	BS1 (or BS2 or BS3)	Do Crossout
4.		BS2 (or BS1 or BS3)	Do Decrement
5.		BS3 (or BS1 or BS2)	Do AddTenToM
6.	(....), TS	TS	Assert (TAKEDIFF)
7.	(...., TS), DTD	DTD	Do TakeDiff

Initially WM holds just (PROCESSCOLUMN). The conflict set consists of rules TS and CM, since they are the only ones whose LHSs are satisfied. By recency-in-PM, rule CM is chosen and fired. Thus the child performs a Compare on the units column of the sum, which puts the symbol (S GTR M) in WM. On the second cycle, the applicable rules are TS, B2A and CM, but CM is discarded because it has already fired on the same data, and by the principle of recency B2A wins out over TS. So B2A fires, depositing the element (BORROW) in WM. Since (BORROW) is the most "recent" element, it is clear that all rules except BS1, BS2 and BS3 are ruled out. There are no grounds for choosing between them so they fire off in any order one after the other on successive cycles. They carry out the borrowing.

By now all the rules which for one reason or another had priority over TS have fired, so it is now the turn of TS. It deposits the goal (TAKEDIFF) in WM, which finally triggers DTD. The actual differencing now takes place, between an M of 14 and an S of 8, yielding the element (RESULT 6) in WM.

Modelling of correct subtraction

Figure 4 shows a complete PS for the method of decomposition. Note that rule TS has been changed to ensure that the second column is

processed. We have added a rule for writing down the answer digits (WA) and a rule which characterises the processing of the second column (FIN). We have also added a rule (B2C) for a shortcut in processing the special case where S = M. In a rule like TS where the RHS specifies more than one element to be added to WM, the convention is that the leftmost one is treated as the most 'recent' for the purposes of conflict resolution, so that by the principle of recency the TakeDiff symbol will take precedence over NextColumn in triggering further rules.

Figure 4: Full PS for subtraction by decomposition

```
TS    ((PROCESSCOLUMN) --> (TAKEDIFF) (NEXTCOLUMN))
WA    ((RESULT =x)     --> (WRITE =x))
FIN   ((NEXTCOLUMN)    --> (SHIFTLEFT) (TAKEDIFF) (STOP))
B2A   ((S GTR M)       --> (BORROW))
B2C   ((S EQ M)        --> (RESULT 0) (NEXTCOLUMN))
BS1   ((BORROW)        --> (CROSSOUT))
BS2   ((BORROW)        --> (DECREMENT))
BS3   ((BORROW)        --> (ADDTENTOM))
CM    ((PROCESSCOLUMN) --> (COMPARE))
AC    ((RESULT 1 =x)   --> (CARRY) (RESULT =x))
DTD   ((TAKEDIFF)      --> (DO TAKEDIFF))
```

We have also added AC, a rule from addition for performing carries. In the PS as it stands, rule AC will never fire because in correct subtraction the item (RESULT 1 =x) will never be found in WM. However if any of the subtraction rules is omitted then there will be circumstances in which AC may fire.

We can justify the analysis implicit in these rules by citing contemporary methods for teaching subtraction. Typically (see Fletcher, 1971) the students are given *separate* practice in taking differences and comparing. They are initially given examples that do not require comparison. Borrowing is taught as the three actions specified in BS1, BS2 and BS3.

A PS for subtraction by equal additions can be obtained from the rules in Figure 4 by replacing rules BS1 and BS2 with a single rule BS4 which reads:

```
BS4 ((BORROW)  -->(CARRY)).
```

Modelling the errors

The children whose work was analysed by Bennett had been taught the method of decomposition. In order to model their errors we changed or omitted various rules in the PS of Figure 4 and then ran them using the OPS2 interpreter. The error seen in example A (which accounts for 50% of all Bennett's errors) can be obtained by omitting the comparison rule CM or rule B2A which triggers the borrowing process. Either hypothesis is plausible. For a particular student, analysis of past protocols might provide the information for determining which is in fact the case. Another possibility is that all three rules for the borrowing process, BS1, BS2 and BS3 are missing.

Errors of type B can be accounted for by omission of rule BS3. The error seen in example C is an 'inappropriate borrow', and is also very common. We can account for it by replacing the comparison rule CM

by a borrowing rule which *always* fires, such as

B1 ((PROCESSCOLUMN)) --> (BORROW))

The occurence of such a rule could result from a student believing that borrowing was an essential part of the subtraction process or as a result of being given a sequence of examples in which borrowing was always necessary. In example C, having taken the difference between 18 and 5 and obtained 13, the PS deals with the ten by rule AC. Students almost invariably complete subtraction sums finding some way to use up any digits calculated en route. The same happens with example D, and a trace of the modified PS running on this problem is given in Figure 5.

Figure 5: Trace on Example D of PS of Figure 4 together with rule B1

```
(SUBTRACTION 2 7  -  1 2)
0. B1
    ADD = ((BORROW))
1. BS2
***** DECREMENTING M DIGIT - NOW 1
2. BS1
***** CROSSING OUT M DIGIT
3. BS3
***** ADDING 10 TO M - NOW 17
4. TS
    ADD = ((TAKEDIFF) (NEXTCOLUMN))
5. DTD
    ADD = ((RESULT 1 5))
6. AC
***** CARRYING TO S2 - NOW 2
    ADD = ((RESULT 5))
7. WA
*****|WRITE DIGIT 5|
8. FIN
***** SHIFTING LEFT TO NEXT COLUMN
    ADD = ((TAKEDIFF) (STOP))
9. DTD
    ADD = ((RESULT 1))
10. WA
*****|WRITE DIGIT 1|
```

It is a common complaint of teachers that as a result of changing school, pupils are taught both methods of subtraction and end up being confused between them. We can account for example E by supposing that the student has learned a muddled version of borrowing including the BS1 and BS2 rules for decomposition *as well as* rather than as an alternative for the BS4 rule for equal addition, but omitting the essential BS3. Example F is similar except that we must hypothesise that the strategies for addition and subtraction are also muddled so that rule FIN reads

FIN ((NEXTCOLUMN) --> (SHIFTLEFT) (FINDSUM) (STOP))

Errors of strategy of the type shown in example G can be accounted for by adding the rule BS4 for equal addition to the full decomposition method given in Figure 4.

Many types of error such as inappropriate borrowing do not occur in the special case where $S_1 = M_1$. We account for this by use of rule B2C.

These models can be adapted for multi-column subtraction essentially by changing the RHS of FIN to be ((SHIFTLEFT) (PROCESSCOLUMN)) - see Young (1977).

Discussion

By using PSs with the stated conventions we have been able to construct parsimonious models of children's subtraction behaviour which

1. are of the appropriate "grain size" for the processes we are interested in,
2. explain many of the observed types of errors as the consequence of omitting one or more of the rules from the "correct" PS, and
3. account for the effects of employing components of strategies for different tasks or methods simultaneously (i.e. the two subtraction methods and addition).

The models presented here could be used as student models (see Self, 1974) for a teaching program for subtraction. In contrast to the procedural network approach to diagnostic modelling outlined by Brown *et al.* (1977), these models can easily be manipulated by adding or deleting individual production rules. Such models could also be employed with a computer tutor such as that of O'Shea (1977) where the tutorial strategy is represented as a set of production rules.

The model we have presented of course has its limitations. Apart from the question of mistakes in the recall of number facts, the model fails to account for phenomena such as runs of errors or the effects of "set" (Thyne, 1954). Furthermore, although this work has taken the view that learning subtraction consists of the acquisition of the appropriate new rules, we have not actually addressed the question of where these new rules come from. This issue will be tackled in a subsequent paper.

Acknowledgements

The starting point for this research was Michael Bennett's perspicacious analysis of children's errors in subtraction. We are grateful to Benedict du Boulay for constructive criticism and for introducing us to the literature on teaching arithmetic. The research was financially supported by grants from the Social Science Research Council held by Jim Howe.

REFERENCES

Anderson, J.R. *Language, Memory and Thought.* Erlbaum, 1976.

Ashlock, R.B. *Error Patterns in Computation.* Bell and Howell, 1976.

Bennett, M. SUBSTITUTOR: A teaching program. Unpublished project report, Department of Artificial Intelligence, University of Edinburgh, 1976.

Brown, J.S., Burton, R.B. & Hausmann, C. Representing and using procedural bugs for educational purposes. *Proceedings of the ACM,* October, 1977.

References (contd)

Downes, L.W. & Paling D. *The Teaching of Arithmetic in Primary Schools.* Oxford University Press, 1958.

Fletcher, H. *Mathematics for Schools.* Addison-Wesley, 1971.

Forgy, C. & McDermott, J. The OPS2 reference manual. Department of Computer Science, Carnegie-Mellon University, 1977.

Klahr, D. & Wallace, J.G. *Cognitive Development: An Information Processing View.* Erlbaum, 1976.

Lankford, F.G. Some computational strategies of seventh grade pupils. ERIC reports. School of Education, Virginia University, 1972.

McDermott, J. & Forgy, C. Production system conflict resolution strategies. In D.A. Waterman & F. Hayes-Roth (Eds) *Pattern Directed Inference Systems.* Academic Press, 1978.

Newell, A. & Simon, H.A. *Human Problem Solving.* Prentice-Hall, 1972.

O'Shea, M.M. A self-improving teaching system. Ph.D. dissertation, Department of Computer Studies, University of Leeds, 1977.

Self, J.A. Student models in CAI. *International Journal of Man-Machine Studies, 6,* 1974, 261-276.

Sloman, A. On learning about numbers. *Proceedings of First AISB Conference,* Sussex, 1974.

Thyne, J.M. *Patterns of Error in the Addition Number Facts.* University of London Press, 1954.

Williams, J.D. *Teaching Technique in Primary Maths.* National Foundation for Educational Research in England and Wales, 1971.

Woods, P. & Hartley, J.R. Some learning models for arithmetic tasks and their use in computer based learning. *British Journal of Educational Psychology, 41,* 1971, 39-48.

Woods, S., Resnick, L.B. & Groen, G. An experimental test of five processes models for subtraction. *Journal of Educational Psychology, 67,* 1975, 17-21.

Young, R.M. *Seriation by Children: An Artificial Intelligence Analysis of a Piagetian Task.* Basel: Birkhauser, 1976.

Young, R.M. Mixtures of strategies in structurally adaptive production systems: Examples from seriation and subtraction. In *Proceedings of Workshop on Pattern-Directed Inference Systems,* Special issue of SIGART Newsletter, June 1977.

A PHILOSOPHY OF 'WICKED' PROBLEM IMPLEMENTATION

Derek Partridge
Department of Computer Science
New Mexico State University
Las Cruces, New Mexico 88003 USA

Abstract

An important characteristic of intelligence is the ability to deal adequately with 'wicked' or ill-structured problems. A methodology is proposed for the computer implementation of certain classes of 'wicked' problems. Curiously, this methodology contrasts sharply with the current artificial intelligence paradigm: a rationalization of this disparity is given. Lastly, the programming methodologies of software engineering appear to be most closely akin to the current inquiry, and this contention is examined.

Introduction

In a recent overview of approaches to design theory, Spillers [18] briefly considers several projects that relate to 'wicked' problems. He mentions Bazjanac [2] who claims that "architectural design is a 'wicked' problem which: (1) has no definitive formulation, (2) has no stopping rule, (3) has no solution in the absolute sense, and (4) each of which is unique".

The claim is also made that, "in fact, 'wicked' problems are so complex that they defy logical treatment"; it is proposed that we can do better than this even with 'wicked' problems.

The term 'wicked problem' was first defined by Rittel [14] who originally presented a list of eleven characteristic properties which were first reported and discussed by Churchman [4]. Bazjanac [2] gives this background along with a comprehensive discussion of the subsequent development of this concept and other relevant models of architectural design (including that of Alexander [1] to which we will return).

Within computer science, psychology and management science the terms "ill-defined" [13], "ill-structured" [17], or "ill-formed" [23] have been used to label much the same concept. Each of the few substantial discussions known to the author appear to be largely isolated, one-time expositions of this concept that are of little direct relevance to the current argument.

Software engineering does not figure in the above list despite one decade of the term 'software crisis' [12] which can be expanded as: the problem of designing software systems of the complexity and reliability demanded when the necessary tools and design methodologies are lacking--a 'wicked' problem.

The field of software engineering has developed a battery of sophisticated but largely non-formalized methodologies for coping with its own brand of 'wicked' problems (e.g., assertional meta-languages, structured programming, the super programmer team, stepwise

refinement and top-down everything). The larger problem is by no means solved, tar-pits still abound, but substantial progress has been made and no slackening of the pace is apparent.

Alexander's exposition of an architectural design methodology [1] is seminal to the theory described herein. Alexander's contribution, from the current viewpoint, was to extol the value of mistakes and to point up the difference between evaluating a putative problem 'solution' as wrong and prescribing correct 'solutions'. But Alexander is judged to have somewhat over-formalized his methodology. An important ground rule for the current viewpoint is that although formalisms have their place and can offer undoubted advantages for the development of theories, the use of a formalism invariably entails a loss of flexibility; the selection of a formalism necessarily excises from the problem area all problem structure non-compatible with the formalism. This is, of course, no problem when the chosen formalism does, in some sense, mirror the problem structure. But all too often an untimely haste to acquire the respectability of a formalism results in a well-structured problem (WSP) that bears little resemblance to the original and yet becomes the prime object of interest and inquiry. Formalisms are too often selected for the simplicity and ease of manipulability that they exhibit rather than their ability to reflect concisely and systematically the original problem structure. The frequent use of various "context-free" formalisms is often a case in point. A formalism should not be selected primarily for the desirability of its structure especially when it then becomes the object of prime interest and the question of its relationship to the original problem is temporarily shelved with perhaps a tacit agreement that suitable perturbations to the formalism will later be applied to reduce this relationship to one of 'sufficient similarity'.

Formalization is thus seen as a process of transmutation: ill-structure to well-structure. Automation of a problem entails formalization. 'Wicked' problems cannot then be automated, but we can automate more or less similar analogues of the original problem and a subsequent process of iteration can be used to decrease the inevitable dissimilarity to some acceptable degree. It is maintained that the initial method used to automate a problem and the resulting transmutation critically limit the possibility of subsequent modifications for producing an acceptable version of the original 'wicked' problem; and furthermore, two types of modification procedures can be distinguished, the 'traditional' approach and a neglected alternative which is presented herein.

An Implementation Methodology: Initial Problem Formulation Strategy

It is proposed that a maximum of effort and expertise be expanded to produce an initial problem formulation (a) which is complete at some level of real-world adequacy thus enabling subsequent adequacy judgments; (b) in which a minimum of the original problem structure has been irrevocably destroyed thus facilitating evolution of the implemented version; and (c) which has an active bias and thus can always generate a potential 'solution'.

The 'completeness' criterion is problematic because of point (1) given earlier (a 'wicked' problem has no definitive formulation). Elaboration of this point yields the proposition that any problem

worthy of this designation will embody numerous non-formalized, and perhaps non-formalizable, features. The difference is that a formalism may just not be apparent as opposed to being just non-formalizable. It is conceivable (and perhaps critically dependent upon what is admitted as 'a formalization') that the latter category being of a residual nature could become an empty one. Aesthetic decisions would fall solidly in this domain but by stretching the concept of 'formalization' and/or substituting an acceptable analogue of the initial problem, some 'formalization' of 'aesthetic' decision making might be forthcoming. With respect to aesthetic decisions, Hess [9] uncovered one formal feature which contributes to judgments of the attractiveness of a face, it is pupil size; dilated pupils are attractive. This example illustrates the point that even for a seemingly non-formalizable feature of the 'wicked' problem we may be able to elucidate formalizable facets of this sub-problem. We then have available guidelines for the automatic production of decisions for these 'wicked' problem features; such features will, in general, be formalizable to some degree. Thus we can automate in a better fashion than mere arbitrary or 'random' decisions; we can utilize some mechanisms for 'informed guessing'.

Minimal destruction of original problem structure, the second criterion, and the completeness criterion have roughly the relationship of algorithm and data structure, respectively. To implement, an initial version of the problem assumptions must be made and restrictions must be imposed, but the processing algorithm formulated should not be critically reliant on the detailed structure of these features. This is in contrast to an algorithm which would widely exploit the introduced structure in order to support subsequent successful functioning. A formulation of restraint as opposed to one of perpetuation.

For example, in the design of a man-machine interface, a typical software problem, certain aspects of the initial 'wicked' problem are clarified and simplified by the assumption that user input will be perfect or at most involve only a small set of possible imperfections. We can then distinguish input processing mechanisms which accept the above restrictions as a given, fixed, structural feature of the problem at hand and thus exploit this well-defined problem feature to the full; typically such a mechanism involves hashing techniques which can deal accurately with exact match situations, maybe some particular error processing routine(s) to process the set of well-defined 'errors', and error exits which dismiss with minimal processing and thus maximal efficiency those input structures which now fall outside the problem implemented. The alternative approach being a mechanism, undoubtedly more complex and less immediately machine efficient, which accepts all input as more or less 'correct' and for which perfectly specified input is a limiting case. Ideally such a mechanism might monitor the actual input and modify its functioning to adaptively optimize the processing of such input. The overall result of these two input processing mechanisms might be superficially very similar with the second approach looking decidedly the worse in view of the fact that the mechanism is larger, more complex and less efficient (when efficiency is measured rather simply in terms of the average time to process of some unit of input once). But if this current implementation is viewed as a precursor of some final implementation of the original 'wicked' problem rather than an end in itself, then, as we shall see, the latter approach would appear to hold far more promise of success than does

the former. In essence the 'mechanism of restraint' provides a basis for evolution. This particular example has been both implemented and tested within the context of a programming language compiler [10].

Flexibility is then a general feature of the initial problem formulation that is of paramount importance. And one major route to flexibility is via an initial implementation incorporating a multitude of individual restrictions and/or assumptions (i.e., informed guessing mechanisms) which may be loosely utilized to obtain initial results rather than 'across the board' strictures which are exploited to the full in order to obtain the 'highest quality' initial results.

The third initial implementation criterion arises from the consideration that every implementation (that is, successive versions from initial to final, see figure 1) must continually produce potential 'solutions' despite unformalized 'holes' within the problem implemented and the prohibitions on the use of rigid, sweeping assumptions or restrictions. This final feature is that the mechanism of 'informed guessing' must have an active bias such that, however small the degree of information currently available to the individual mechanism, it will always contribute a decision to the total system function. In an extreme situation decisions must be arbitrarily produced (although not totally arbitrarily for if the problem or sub-problem has any form at all then potential solutions can always be somewhat circumscribed), For it is exactly these potential solutions that stem from largely unformalized problem features that pave the way for greater, if not total, formalization of the problem features which generated them.

The Iterative Development of an Adequate Formulation

Other things being equal, the most adequate implementation of a 'wicked' problem will be the one that utilizes the maximally informed guesses, that is, the one that incorporates a maximum of the formalizable facets that the problem in question contains. The question then is: can we systematically expose all of the formalizable facets of any given 'wicked' problem, and if so, how? The short answer would seem to be, no! But, what can be done is to systematically expose as many formalizable facets of our 'wicked' problem as is deemed necessary to produce an adequate implementation of our original problem. The procedure for accomplishing this relies upon the continued output of potential solutions from the sequence of inadequate implementations which are modified in the light of the major inadequacies that are apparent within these 'solutions'.

Points (2) and (3) given earlier (i.e., wicked problems have no stopping rule, and wicked problems have no solution in the absolute sense, respectively) can now be addressed.

'Wicked' problems are essentially real-world problems and as such any computer implementation of such a problem must potentially fulfill some real-world need. If we can say anything significant about some potential 'wicked' problem solution then we can say what is wrong with it in the light of its clashes with the real world. If we cannot say that anything is wrong with it, then the 'solution' is, in the strongest sense one could demand, adequate. This procedure is illustrated in figure 1. The consistent production of adequate solutions is the criterion of system adequacy. Hence we have a system evaluation procedure which incorporates a stopping 'rule'. It should be noted that

this procedure is more of a loose methodology than a definitive algorithm: for example, what is and what is not a major solution inadequacy can be strongly influenced by societal and personal bias. This then suggests, as one would expect, that the 'solution' adequacy procedure for 'wicked' problems is by no means a context-free feature. Design may still be "an argumentative process between those concerned" [2], but now the arguments are severely and explicitly constrained by the fact that they have to fit into the general framework of the current implementation and particular 'solutions'. A probably tedious and certainly time-consuming symbiotic type of relationship between experts within the particular problem domain and the automated problem would seem to be unavoidable. Which brings us onto the last 'wicked' problem point: (4) each 'wicked problem is unique. A point which perhaps implies the non-existence of a generalized structure which would be applicable to a wide range of 'wicked' problems. A viewpoint which has gained credence in the light of the failure of artificial intelligence to produce substantial and generally useful heuristics. Although as we shall later see this dearth of generalized heuristics may be attributable to a methodological failing rather than non-existence of generalized techniques. In support of the former view, we might note that leading artificial intelligence researchers, such as Minsky [8], are now espousing the view that intelligence is fundamentally a collection of sophisticated and specialized heuristics.

Informed guessing as a mechanism for problem solving can also be a process of expediency rather than necessity. Guessing is usually faster than following some formalized route to an answer, but unfortunately, guessing is also usually less reliable. So even when confronted with a totally formalized situation, informed guessing may be an optimal procedure. The game of chess, for example, is illustrative of a situation which can be totally described by a simple algorithm and yet the 'inefficiency' associated with the algorithmic route is so extreme as to effectively rule it out of consideration, informed guessing in the guise of 'heuristics' is the mechanism employed. It could be said that chess is a 'wicked' problem in practice, but not in principle. More typically the degree to which a decision is based upon guessing (or the degree to which a guess is 'informed') will represent some point of balance between efficiency and reliability.

'Wicked' Problems and the A.I. Paradigm

If there is a body of knowledge pertaining to computer implementation of 'wicked' problems, then the field of artificial intelligence is where we might most reasonably expect to find it; for complex, ill-structured problem 'solving' is the domain of human intelligence and so possession of this ability can be considered an important characteristic of intelligence. Typically 'heuristics' are employed to facilitate computer implementation of 'wicked' problems. Thus we might expect the implementation of individual problems to provide practical evaluations of the heuristics utilized, resulting in an accumulation of efficient and reliable heuristics each associated with some domain of applicability. Such has not been the case, as Spillers [18] notes with disappointment.

The artificial intelligence paradigm typically involves the employment of sweeping assumptions and 'across the board' strictures

to immediately reduce an initial 'wicked' problem to a well-structured problem (WSP). This WSP is then programmed utilizing heuristics as an efficiency expedient as well as for the circumvention of any residual ill-structure. The proof of the 'heuristic' (much like that of the pudding) resides largely in the results that it produces. A straightforward, here and now, empirical justification often leaves much to be desired, and so one is naturally loath to embark upon a course which appears to only offer the promise of an even more vulnerable immediate justification. The temptation is to move in the direction of particularising any heuristic for more dramatic here and now results as opposed to a more generally powerful version, less impressive here and now but carrying the promise of satisfactory performance on a significantly wider range of problems. Weinberg [19] notes, "heuristic devices don't tell you where to stop".

Now given the nature of 'heuristic' devices and given that the immediate problem of interest is the highly constrained WSP derived from some initial 'wicked' problem, there are essentially two possibilities open: (a) to employ generalized heuristics for less impressive immediate results but hopefully retaining some potential for coping with modified versions of the problem which move it in the direction of the original 'wicked' problem; and (b) to produce a very showy implementation, which entails the exploitation of the specific WSP structure by means of very particular heuristics. This second course of action would seem to offer little possibility for perturbing the current WSP implementation such that it moves effectively back towards the original 'wicked' problem. The former course of action embodies potential for accommodating relaxations of the current problem constraints but, being a non-realistic problem (a 'toy' or 'micro-world' version of the original 'wicked' problem) no guidelines are apparent for judging the 'goodness' or 'adequacy' of the 'heuristics' employed. It is difficult to judge when a 'heuristic' is striking the right balance between being sufficiently good for the problem at hand and yet maintaining the potential necessary for accommodating subsequent problem perturbations.

Artificial intelligence, insofar as it has viewed the derived WSP as a first cut toward implementing the original ISP, has generally run up against these problems. The bias has usually been toward the second course of action thus effectively making the derived WSP an end in itself by blocking the route back to the original 'wicked' problem. McDermott [11] in a scathing review of natural language artificial intelligence projects, said, "translation methods that worked on 'the big red block' will not succeed, and cannot be extended to succeed on ..." a given example which falls outside the strictures governing the 'toy' languages which such systems typically employ. The previously described approach to the computer implementation of 'wicked' problems utilizing 'informed guessing mechanisms' and the above described artificial intelligence paradigm are contrasted schematically in figure 2.

I believe that this rather sweeping characterization does contain substantial explanatory power for the "enigma" [18] that is, artificial intelligence. It is viewed as "the first step fallacy" by Dreyfus [5], and as "the Scylla of trivial first generation implementations" by Wilks [20] in a recent report to the National Institute of Education.

Within the same report Winograd [21] mentions the early success of linguistics which was based on a narrow concentration on problems of syntax (a relatively WSP), avoiding issues of meaning and communication in language use (the original 'wicked' problem); he then states that, "there is much current dissatisfaction with this restrictive paradigm, and many prominent linguists are beginning to reject a syntax-dominated approach and to look at these broader issues".

Recently, Winston [22] has tried to justify the micro-world approach in A.I. by offering the analogy that, "limited domains of discourse are the E. Coli of language research". The critical dimension along which this analogy fails concerns the fact that E. Coli is an existing, real world system that molecular biologists must deal with as best they can, while limited domains of discourse are artifacts that can be restructured or created at will to facilitate the application of any particular theory.

The Management of Complexity and Software Engineering

Compounded with the earlier problematic points given and discussed as particular to 'wicked' problems, there is the necessity to also manage the complexity of 'wicked' problems; but complexity is not, of course, a feature peculiar to 'wicked' problems. Software engineering is the automation, in the sense of computer program implementation, of complex problems. It can also be maintained that many of the problems are 'wicked' problems. Although the software engineers typically consider initial, complete and formal specification of the problem as an essential prerequisite of any actual implementation. This critical and problematic area, "this latest addition to the system technology field--requirements--in the form of requirements definition and requirements engineering" [15] was recently the subject of a special issue of the IEEE Transactions on Software Engineering [15]. In the Guest Editorial Ross gives a schematic illustration of the general problem [15, Figure 1] which is closely analogous to, but independent of, the figure 1 presented herein.

Now the feature that distinguishes software engineering from say, artificial intelligence, and makes it particularly relevant to the current proposals is: that the finished product invariably has real world applicability and use. In addition, software engineering as a discipline within the domain of 'wicked' problems is further distinguished from say, architectural design, by the fact that computer implementation is not just a modeling process but it is also construction of an automated version of the 'wicked' problem itself. Software products are not then usually open to human interpretation from what is essentially a model (of, say, the design of some edifice) to the real world situation modelled (the human design process and the edifice itself). Thus software engineering is not permitted to cop out. A more or less 'wicked' problem is often the initial 'given' and, although it is top priority to transform this problem into a formal analogue before proceeding further, the subsequent implementation swims or sinks in the light of real world application: that is, back in the domain of 'wicked' problems. Thus the final version of a system is a more or less adequately automated 'wicked' problem, rather than a model of the problem.

So within the battery of sophisticated complexity management techniques developed for software engineering one can also discern tid-bits for dealing with the inevitable 'wicked' problem intrusions as well as certain dimensions of the overt management of complexity techniques that possess particular applicability to 'wicked' problems. The problem of coming to grips with the ill-structure of the real world looms largest during initial formal problem specification and in the final user evaluation of the product: that is, the two major real world interfaces. The problem specifications are never exhaustively and unambiguously formalized and the final product never performs totally to the user's expectations. Without implying that 'wicked' problems features are solely to blame, for sheer complexity as well as human perversity have important parts to play, 'wicked' problem features do have a role and are accommodated by the techniques employed.

Flexibility or adaptability, for example, is often given high priority within the software design procedure. System adaptivity is considered desirable not only for accommodating future environmental changes and system enhancements, but also as a means of smoothing the two real world interface problems. It is widely recognized that many inadequacies of the automated system will only become apparent when the user can interact with the real system. The inherent adaptivity can, hopefully, be utilized to remove these inadequacies. Better still, the first system delivered to the user is not full-blown and virtually finished, but adaptable, it is a first phase implementation which embodies a maximum of gross structure and a minimum of commitment to finer detail. Thus real world application is utilized to elucidate and confirm the problem structure implemented in steps of successively finer detail.

The techniques of top-down design closely followed (say, only one refinement step behind) by top-down coding lead to this concept of top-down testing; perhaps a very useful set of concepts to be combined with the earlier methodology for computer implementation of 'wicked' problems. In the terminology of the earlier exposition: system inadequacies are systematically exposed in steps of successively more detail by means of real world interaction. No stress seems to be laid upon the difference between adequacy and inadequacy judgments. The real beauty of this scheme is that real world feedback can be utilized to modify gross features of the system structure before many of the relatively finer problems features are implemented.

Concluding Remarks

A general methodology for the automation of wicked problems was presented and it was demonstrated why the field of A.I. has contributed surprisingly little to our knowledge of how to program 'wicked' problems. A major dichotomy of A.I. methodologies was proposed and explained (figure 2). It was then suggested that the 'micro-world' option is the methodology usually adopted and that there are good reasons why little of relevance to 'wicked' problems is to be expected from this track. It is of relevance that some of the notable 'successes' in A.I. (e.g. DENDRAL [7] and MYCIN [16]) were constructed very much in accordance with the above philosophy and that a number of other particular examples of a similar philosophy are being pursued at MIT as was noted at the AISB Conference of 1974 [24].

Real world trial-and-error system development is an integral part of the methodology described; it may not always be feasible (e.g., a Ballistic Missile Defense System [6]). Simulation is then a necessity thus delimiting a class of extra-'wicked' problems. Brooks goes as far as to suggest, "plan to throw one [version of the system] away...you will anyway" [3]. Such a strategy smacks of a reversion to, or an absence of progress from, an earlier software design methodology, the "Wright brothers technique" [12]: "build a whole airplane, push it off a cliff, if it crashes start again". The important difference is that now the essentiality of real world interaction is recognized and it is harnessed to guide system design and development. A minimum of time and effort spent on, and commitment to, details means that the real world testing (pushing it off a cliff) is applied initially to the substantiation, rejection and general clarification of gross system features; a maximally informative real world test from a minimum commitment and risk of resources. In later stages of the system development finer system details are subjected to the rigors of reality.

There also appears to be a certain discordance between the pivotal position of testing in the above schema and a powerful current in programming methodology, viz: the movement to 'prove' the absence rather than test for the presence of errors. Again, there is an important difference when dealing with 'wicked' problems. We are not primarily attempting to implement some formalized system but trying to emulate to some acceptable degree a substantially more elusive and intangible 'object that is only known to exist in the real world (if it exists there).

References

[1] Alexander, C., Notes on the synthesis of form, Cambridge: Harvard University Press, 1964.

[2] Bazjanac, V., "Architectural design theory: Models of the design process", in W.R. Spillers, Ed., Basic Questions of Design Theory, Amsterdam, The Netherlands: North-Holland, 3-19, 1974.

[3] Brooks, F. P., The mythical man-month, Reading, Mass.: Addison-Wesley, 1975.

[4] Churchman, C. W., Wicked problems, Management Science, 4 No. 14, B-141 and B-142, 1967.

[5] Dreyfus, H. L., What computers can't do, New York: Harper & Row, 1972.

[6] Dreyfus, J. M., BMD-STP Project, TRW/University Workshop on Software Engineering, Redondo Beach, California, April 13-15, 1977.

[7] Feigenbaum, E. A., et al., Generality and problem solving: a case study using the DENDRAL program, Machine Intelligence 6, B. Meltzer and D. Michie (Eds.), N.Y.: Elsevier, 165-190.

[8] Goldstein, I., S. Papert and M. Minsky, "Artificial intelligence, language, and the study of knowledge", in Artificial intelligence and language comprehension, Washington, D.C.: National Institute of Education, 27-77, 1976.

[9] Hess, E. H., "Attitude and pupil size", Scientific American, 212, 46-54, Apr. 1965.

[10] James, E. B. and D. Partridge, "Tolerance to inaccuracy in computer programs", Comp. J., 19, 207-212, 1976.

[11] McDermott, "Artificial intelligence meets natural stupidity", SIGART Newsletter, No. 57, 4-9, April 1976.

[12] Naur, P. and B. Randell, Eds., Software Engineering, Brussels 39, Belgium: NATO Scientific Affairs Division, 1969.

[13] Reitman, W. R., Cognition and Thought, N.Y.: Wiley, 148-226, 1965.

[14] Rittel, H.W.J., On the planning crisis: systems analysis of the first and second generations, Bedriftsokonomen, No. 8, 390-396, 1972.

[15] Ross, D. T., "Guest Editorial: Reflections on Requirements", IEEE Transactions of Software Engineering, vol. SE-3, 1, January 1977.

[16] Shortliffe, E. H., Computer-Based Medical Consultations: MYCIN, N.Y.: Elsevier, 1975.

[17] Simon, H. A., The structure of ill-structured problems, Artificial Intelligence, 4, 181-201, 1973.

[18] Spillers, W. R., Design Theory, IEEE Transactions on Systems, Man, and Cybernetics, 201-204, March 1977.

[19] Weinberg, G. M., An introduction to general systems thinking, N.Y.: Wiley, 1975.

[20] Wilks, Y., "Natural language understanding systems within the AI Paradigm: a survey and some comparisons", in Artificial Intelligence and language comprehension, Washington, D.C.: National Institute of Education, 79-120, 1976.

[21] Winograd, "Artificial intelligence and language comprehension", Washington, D.C.: National Institute of Education, 1-25, 1976.
[22] Winston, P. H., Artificial Intelligence, Reading, Mass.: Addison-Wesley, 1977.
[23] Uhr, L., Pattern recognition, learning and thought, Englewood Cliffs, N.J.: Prentice-Hall, 268-284, 1973.
[24] "from our Experimental Psychology Correspondent", Bugs are beautiful, Nature 250, August 16, 1974, 533-534.

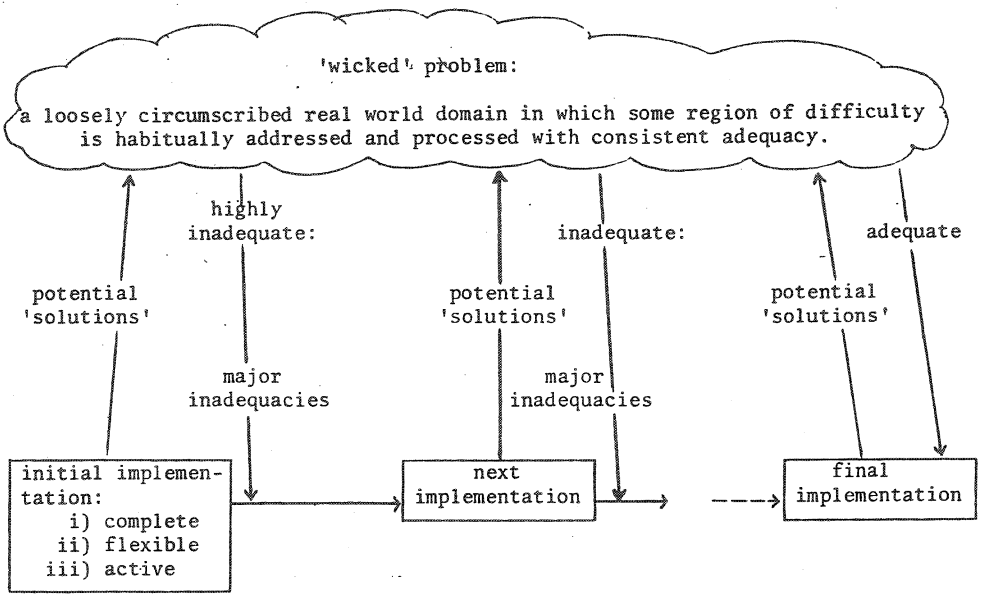


Figure 1 - a theory of 'wicked' problem design

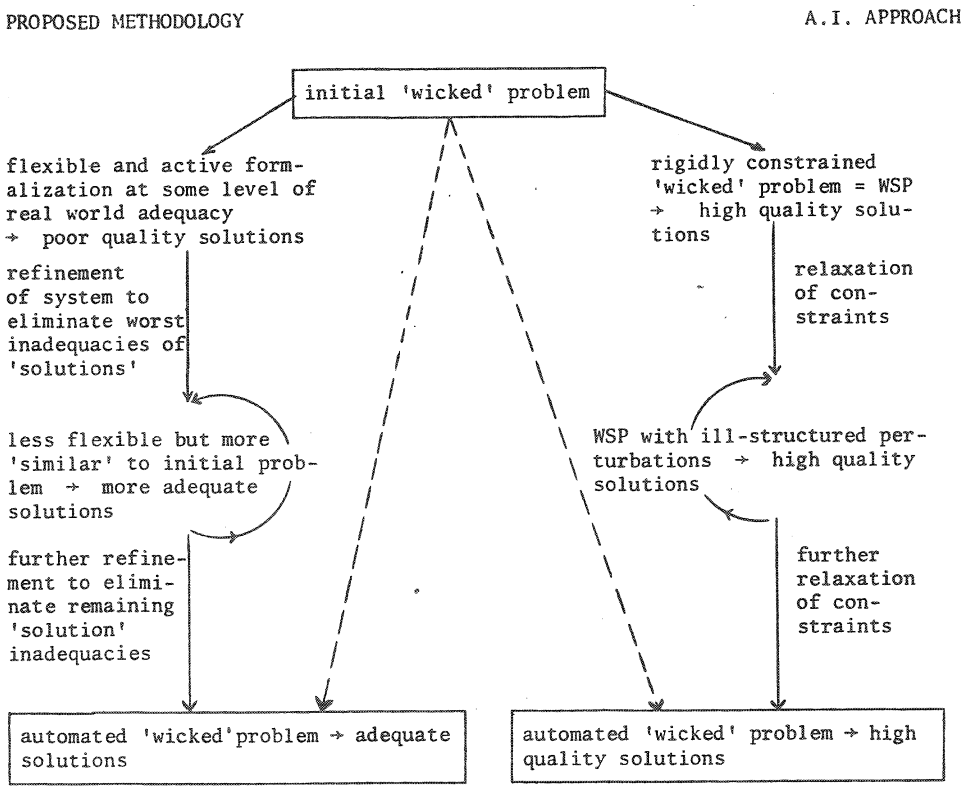


Figure 2 - a schematic representation of two different artificial intelligence methodologies.

A HUMANLIKE APPROACH FOR AUTOMATIC THEOREM PROVING

Dominique PASTRE
University PARIS VII
France

ABSTRACT

A program, proving theorems in set theory by the use of rather natural methods, was first written after observation of the author's behaviour.

Then, mathematicians were observed while resolving some problems of real analysis and set theory. Their argumentations were analysed in order to define new methods for automatic theorem proving by imitation of both human behaviour and brain organisation. Several points appeared to be important : the language used, knowledge and informations necessary in memory, working representations, methods and heuristics, and finally the possibility of finding "good ideas".

I - INTRODUCTION

Various methods of automatic theorem proving have been found and some programs have been written. Some methods are very simple and may be theoretically very efficient, but they are not efficient in practice because of the disastrous number of generated objects and of the difficulty of choices.

Although many strategies have been imagined to improve actual procedures, I believe that it is impossible to make them practically very efficient, and that it is necessary to search in quite opposite directions.

Since the performances of man are quite better, a possibility may be the following : imitate the human behaviour and copy his heuristics. Such a program would be complicated and would need a large memory space but it could be practically efficient.

I experimented this approach in writting a program, "DATTE", which behaves in a rather natural manner. Its results were very encouraging (150 theorems proved in Halmos(1960), Krivine(1969), Queysanne(1964) and Sigler(1966), some of them are difficult). So, I thought it would be interesting to continue in the same direction and it appeared to me that it would be useful to observe mathematicians "in action". Then, I observed and analysed the argumentations of some mathematical researchers resolving problems of real analysis (Choquet,1964) and set theory (Krivine,1969). (These argumentations are described in details in Pastre(1977).)

Here, after a brief paragraph about "DATTE", I shall mention some important points which appeared during the mathematicians' observations and which seemed to me useful for automatic theorem proving.

II - "DATTE",A PROGRAM WHICH PROVES THEOREMS IN SET THEORY

It is described in details in Pastre(1976,1978) and a comparison is made with Brown's (1978) program which works in the same domain.

Some representations, methods and heuristics are humanlike : the theorems and some rules that the program itself constructed are in the form "Hypotheses $\Rightarrow$ Conclusion", all the relations and some elements are drawn in a graph with which the program works. The main procedure is a natural procedure (all the properties which can be deduced from the actual hypotheses are added as new hypotheses until the conclusion is easily proved). The rest of the program is built around this procedure; different actions are tried according to a plan which was carefully elaborated by observing my own behaviour.

Besides the methods, the conception of the program was different from that of usual programs : there is no underlying theory ; but rather, a domain was defined, a large number of theorems was collected and the program was written step by step in a very empirical manner.

III - OBSERVATION OF MATHEMATICIANS

1°) The language

Indeed, nobody works formally. In general,"ideas", good or bad, are first expressed in a confused manner. It often happened that I, observer who knew how to solve the problem, "knew" that the researcher had found, but he had yet a large amount of precise work to do.

For example, in a problem about uniform continuity, cutting an interval was the "good idea". When one has this idea of cutting, the problem is almost solved. Yet, this cutting has to be done correctly. But the main point is the idea and a large part of the argumentation is done in natural language. I believe that it is necessary to define a computer theorem proving language which would not present the difficulties of natural language but which would be as far from the entirely formalised language as natural language is and with which the argumentations of the mathematicians can be expressed.

The language used by the researcher when he writes is precise and rigorous but ... illegible. This is true for the solutions given in books and also for a writting of a solution of one of the problems, given some time later by one of the observed researchers. One cannot hope that a program could immediately give solutions of this type. This formalised language is not useful for searching, neither for explaining, but only for communicating in a rigorous manner with experienced people.

2°) Informations - Knowledge

a) Each notion must be linked to a large number of informations. Knowing the formal definition of uniform continuity for instance is not enough. This notion must be related, on one hand, to some "ideas" — for example, a "small" interval may be interesting — , on the other hand, to some theorems which must be applied as reflexes. The notion of "closed and bounded" (it is not the conjunction of the two concepts of closed and bounded but a concept by itself) must be linked to the fact that examples of usual closed and bounded sets in $\mathbb{R}$ are closed intervals. The notion of compactness is the same, in $\mathbb{R}^n$, as the notion of "closed and bounded". This is a theorem, but it is applied as a definition.

That complex network of relationships which exists in the brain of the mathematician should be present in the memory of the computer. This will be redundant if compared with strict theoretical necessity,

but we work with redundant informations. And so, we can easily use the information which is in the better form.

b) Known theorems are used in various manners :
- They can be used as results. One tries to apply them if a situation looks like their hypothesis or if the goal to be proved looks like their conclusion. Some theorems are even applied by reflex without beeing expressed neither even mentionned, for example all the theorems about (uniformly) continuous functions. Other theorems are expressed but their application is also a reflex.
- Other known theorems are applied rather as tools. It is the case of Zorn's lemma : this result is not very interesting in itself, knowing its proof is useless, but using it sometimes makes proofs very easy.
- A last manner for a theorem to be useful is to give a type of proof. For example, knowing how the continuity of a norm can be proved helps to prove the continuity of a function which is not quite a norm.

3°) Representations

Mathematicians do not make formal deductions but rather they reason with their own representations of familiar objects. It is sometimes possible to put these representations in concrete form but they may remain purely mental. Some of them are universal, other are specific to each person.

Geometrical representations and plane drawings are very important. They may be only a support. But sometimes the argumentation is only geometrical and nobody would feel necessary to actually prove some intermediate results which are really obvious on the drawing. Moreover the mathematician frequently makes a drawing in the plan ($\mathbb{R}^2$), possibly imagines a volume in our usual space ($\mathbb{R}^3$), and has the intuition that what he says is also true in $\mathbb{R}^n$.

The visual geometry is an usual support for man. It would be necessary to find for computers, a similar, if not identical, representation.

The drawing is sometimes a concrete support but the mental image which is in the brain is more complex. For example, everybody "sees" that the following drawing represents an infinity of points converging to the limit 1 :



In one of the problems, sequences which converge to every point of the first sequence are built, and the construction is reiterated to order n and to infinity. Various diagrams are used to express, rather badly, the mental image which is clear in everyone's brain. Mathematicians have a mental representation of the "real line" in which the construction is easily made before the formalisation, which is delicate.

The computer must have its own representation of the "real line" in order to be able to work upon it, and to "see" accumulation points and isolated points as mathematicians do.

I believe it is indispensable to give to the computer good representations of all notions, which either imitate human ones or are computers'own. Anyhow it is necessary to deeply analyse human representations for each notion in particular.

Each mathematician may also have his own way of thinking which

depends on his speciality and his competences. I believe that computers must have in memory something analogous to the mathematician's "experience".

4°) Methods - Heuristics

a) Among observed methods, the most important and general one is undoubtedly the necessity of doing plans. This idea was successfully experimented in another domain by Pitrat(1977).

For some simple problems, after one has the "good idea", the remainder of the proof can be done mechanically enough. But as soon as a problem becomes rather complicated, it is necessary to do a plan, i.e. to decompose it into successive goals to reach. In effect, the immediate step to execute may be far away from the ultimate goal.

Sometimes even, several more or less detailed plans are to be done. When some decomposition level is reached, a man will admit some points as true. (This level depends on his own experience.) In general, it is quite legitimate, but it may also happen that the difficulty is overlooked. In fact, a certain mathematicians' consensus allows to judge if the proof is correct. It is difficult to estimate what a computer could admit as true. Anyway, even if, at last, every detail has to be proved, it must be merely either an easy verification work or an easy deduction by analogy with a known stored result.

b) Some other general methods were observed :
- Make use of your own mistakes or of what is not suitable ;
- Study particular cases, either as generalizable examples, either as counterexamples ;
- Separate difficulties and replace a problem by two or more independant problems.

c) It is possible, with some reflection effort to express heuristics which are, a priori, very intuitive. A good mathematician succeeded in two difficult situations. Other ideas, a priori very intuitive, could hopefully also be stated precisely. An indepth analysis seems necessary, preferably done by specialists in the subject.

5°) "Ideas"

a) A frequent phenomenon must first be underlined : the "good idea" which does not lead to the expected result, either because it is applied to a wrong object and the use of this object is never challenged, either because it is driven away by another idea — good or bad — . When knowing this phenomenon, it is possible to try and avoid that programs behave in this manner.

b) Now, it would be interesting to know how the "good ideas" are found, not only the really new and original ideas of the creative researchers, but also merely ideas that the average mathematician obtains after a little thought.

Many authors studied this question (de Bono(1972), Hadamard(1959), Polya(1945)), but no answer has been yet found.

It is very difficult to observe how "good ideas" are found because we have not access to the "programs which are in our brain". We can only examine their results and try to pinpoint the data which are fed

to them. These programs are gradually refined as the mathematician becomes more and more experienced.

Perhaps in the future one can expect that learning techniques would build in computers the same programs, or equivalent ones !

IV - CONCLUSION

But I believe that now, we must start by deeply analysing subjects in order to define important notions, organize the storage and adequate linking of all informations and set up a list of heuristics to be tried in certain situations.

Three other types of experiences can still be considered. It may be asked to a mathematician to express all that some notion calls up, and how he would explain (vulgarize) it. One can also observe a mathematician reading a new mathematical text in order to examine how he "understands" it and, little by little, absorbs new informations. Finally, a careful introspection about one's own representations could give types of reprentations which could be useful, although not universal.

REFERENCES

E.de Bono (1972), The use of lateral thinking, Penguin books, England

F.M.Brown (1978), Towards the automation of set theory ans its logic, to appear in Artificial Intelligence

G.Choquet (1964), Cours d'analyse, Tome II Topologie, Masson, Paris

G.Hadamard (1959), Essai sur la psychologie de l'invention mathématique, Blanchard, Paris, reed. Gauthiers-Villars, Paris(1977)

P.R.Halmos (1960), Naive set theory, Van Nostrand, Princeton

J.L.Krivine (1969), Théorie axiomatique des ensembles, PUF, Paris

D.Pastre (1976), Démonstration automatique de théorèmes en théorie des ensembles, Thèse de 3ème cycle, Paris VI

D.Pastre (1978), Automatic theorem proving in set theory, to appear in Artificial Intelligence

D.Pastre (1977), Observation du mathématicien : aide à l'enseignement et à la démonstration automatique de théorèmes, D.D.n°8 edited by the U.E.R. de Didactique des Disciplines at the University Paris VII, to appear in Educational Studies in Mathematics

J.Pitrat (1977), A chess combination program which uses plans, Artificial Intelligence 8

G.Polya (1945), How to solve it ? , Princeton University Press, Anchor Books (1957)

M.Queysanne (1964), Algèbre M.P. et Spéciales A A', Collection U, Armand Colin, Paris

L.E.Sigler (1966), Exercises in set theory, Van Nostrand, London

A NETWORK REPRESENTATION FOR PROCEDURAL KNOWLEDGE

Brian Phillips
University of Illinois at Chicago Circle
Chicago, Illinois 60680, U.S.A.

ABSTRACT

Two kinds of knowledge necessary for text understanding systems that are procedural, not declarative, are processes on the knowledge structure (e.g., path-tracing) and knowledge of concepts like "analogy" and "euphemism". The desirability of being able to manipulate procedures as data makes a case for having the same format for both declarative and procedural knowledge. This is achieved by writing procedures in a semantic net formalism. Executing such a procedure produces a network that explicitly contains the control structure as well as the data.

DESCRIPTORS: procedural knowledge, semantic networks.

1. Procedural and declarative knowledge

A general intelligent system must include procedural and declarative knowledge, which correspond to "knowing how" and "knowing that", respectively (Winograd, 1975). The format for these, and other, dimensions of knowledge could be totally distinct but Scragg (1975) makes a case for uniformity, citing the problems of consistency, redundancy, and learning that arise if separated. But additionally systems should be able to inspect and change procedural knowledge. In his planner, Sacerdoti (1975) has a "criticism phase" to turn unreliable plans into safe ones. Scragg (1975) observes that "It doesn't seem possible that people could in general answer detailed questions about an action but not be able to perform it" (p. 3), implying that question-answering systems should be able to access procedural knowledge. Consequently procedural knowledge does not exist solely to control the performing of actions but is also subject to some of the same processes as declarative knowledge.

Rieger (1976) and Scragg (1975) propose representations for knowledge that are deemed adequate for both linguistic comprehension as well as execution of a process. To test this duality there should be a controlled robot as well as linguistic analysis. Alas there is yet no robot in their systems. It may be intuitively easy to see that a description is inadequate: a procedure for driving a car may be "Twist the steering wheel and jiggle the pedals on the floor". But more complex descriptions need a formal test. Current genuine robots are linked to smaller worlds than are found in language processing systems, e.g., Ambler et al. (1975) have a robot to assemble blocks into toy cars; it has no language capabilities to describe its actions. At present no system could undertake to make Scragg's omelette or to operate a reverse flush toilet for Rieger. Thus there is an unproven gap in the formalisms for descriptions.

I want to represent some procedural aspects of knowledge that are used in language understanding but are not motor processes and so do not require a robot for testing. In section 2 this knowledge is presented. The format used to write the process descriptions is a

semantic network that has previously been used for declarative knowledge in discourse analysis (Phillips, 1977). In section 3 a variety of programs will be presented. Thus overall this is an experiment in seeking a testable uniform formalism for procedural and declarative knowledge.

2. What knowledge is procedural?

To understand one thinks; this is an internal manipulation of knowledge. This is procedural.

Additionally people have lexical concepts that are procedural. Consider the problem of accepting novel utterances. Some may reasonably have an explanation based on deduction from declarative knowledge. For example, "Marv gobbles the caviar" is accepted by knowing "animals ingest food", "Marv is an animal", "caviar is a food", and "gobble is a ingest". For an anomalous proposition, "Harry munches a spider", there is no knowledge that "spider is a food" (Phillips, 1977).

Compare this with judging the analogy "Humming birds are like butterflies". There is a specific implicit context: sucking nectar; growth to maturity is not valid. Deduction from a generalization "Birds are like insects in feeding" just doesn't work; it could produce an analogy between penguins and ants. Analogy is a relationship between concepts that exists when the concepts have common functionally relevant properties. The formal generalization can then be written as a procedure that matches properties. The qualification can be added that recent or oft used analogues may have declarative representations, e.g., "love" and "red roses". In a similar vein Winograd (1975) makes a case for second order knowledge--knowledge about knowledge--being procedural. In particular he mentions heuristics, which like analogy are context dependent.

3. Procedures in network form

To represent procedures in a network formalism it is necessary to show how a computation is defined and how its representation is interpreted. First in this section is an example of a mathematical procedure to acquaint the reader with the general approach. Thereafter come processes on network structure and procedural concepts.

3.1 Mathematical procedures

The smallest unit of processing involves an action and its arguments. Figure 1 contains "subtract", which causes the difference of its arguments (ARG1, ARG2) to be formed. "Subtract" is taken as a primitive. There will have to be a finite number of primitive processes in the system. They will be actions on the network, e.g., traverse an arc, and mathematical operations.

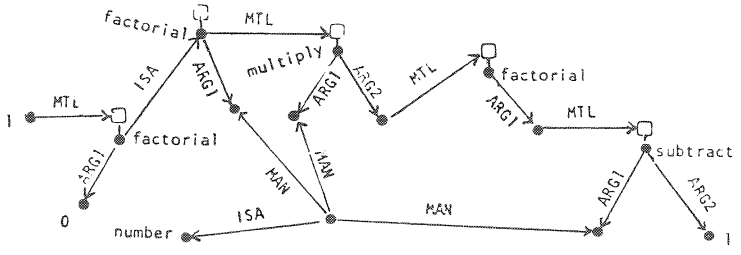


Figure 1 Definition of "factorial"

Modality nodes (represented by □) act as local controllers for the computation. When control reaches a modality, the associated computation is initiated. On completion, the value is associated with the process using a METALINGUAL (MTL) arc. Defined values are also stored using this device. In Figure 1 the value of 0! is shown stored in this manner.

Functions can be represented using metalingual relations. These associate a function name (or in fact any concept) with a net of actions that produces its value. In Figure 1, n! = n * (n-1)! is shown.

Arguments of actions are linked into taxonomies by ISA to show the kinds of valid arguments, e.g., that the argument of factorial should be a number, Figure 1. The MANIFESTATION (MAN) arcs are used in Figure 1 to show that it is the same value of the argument of factorial that appears at various points in the defining computation.

A recursive function is one in which the definiendum is also an action in the definiens, as with factorial in Figure 1.

Executing the procedure given in Figure 1 for 2! will show the potential for network construction. The first statement of the computation is shown in net A in Figure 2. As a value for this structure is not explicitly in the net, the metalingual definition is used, producing the net B in Figure 2. The net copying process should also be in network form and should be shown, but it is omitted for the sake of clarity. The "multiply" cannot be performed, one of its arguments has a definition containing "factorial" which must first be evaluated. But again there is a complex argument to be evaluated. This "subtract" can be performed, to give the answer 1, which is then also the argument of the "factorial". But 1! is not known so again the definition is used to give network C in Figure 2. In C the subtraction gives an argument 0 to the factorial; 0! is explicitly known (Figure 1); the multiplication yields 1 as the value of 1! The multiplication in B can now be carried out giving 2 as the value of 2!.

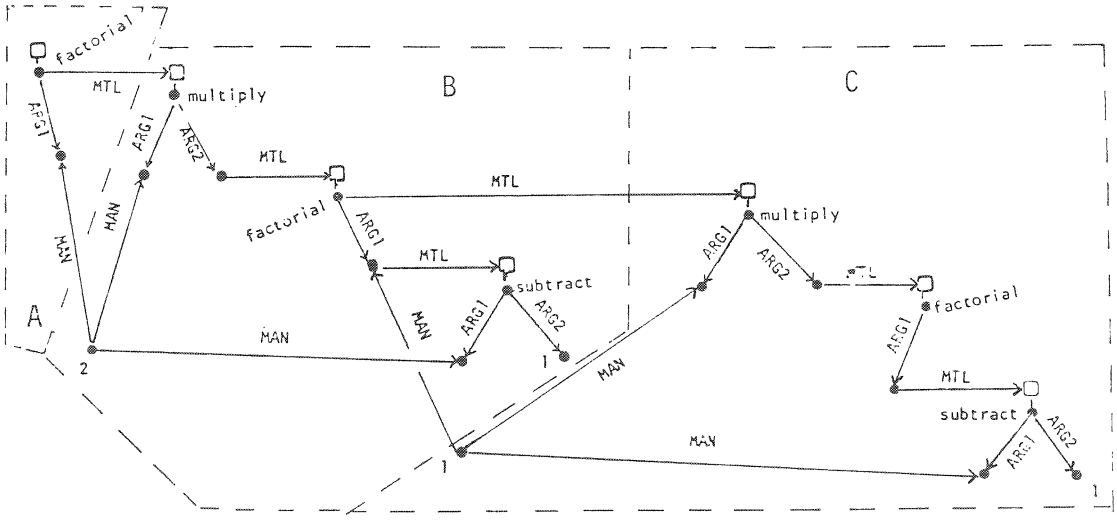


Figure 2 Evaluation of 2!

3.2 General processes on knowledge

It is hypothesised that there are two basic processes in using knowledge (Phillips, 1977). One is path-tracing, which is used in intersection techniques following Quillian (1969). The other is pattern-matching, which matches two subnets.

The primitive actions for path-tracing include "traverse" which

has two arguments, a node and an arctype. It returns a node that is connected to the argument node by the arc; if there is no such node the action returns FALSE. The action "flag" puts a marker on a node and "test" looks for the marker. Another action "locate" finds a node (through a dictionary) in the network that corresponds to a word.

A program is a partially ordered set of computations. Strict ordering is given by SEQUENCE (SEQ) relations between modalities; coroutines are linked by SIMULTANEOUS (SML) relations, Figure 3.

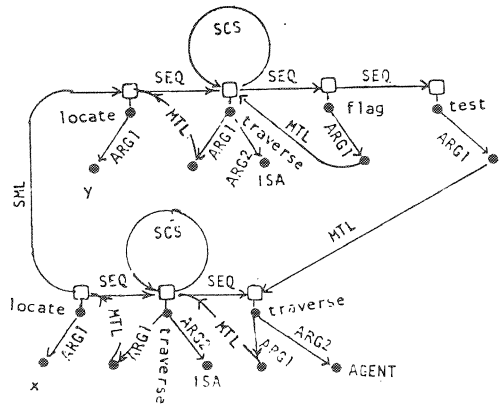


Figure 3 A path-tracing routine

The present thought on conditionals is to have control pass to a modality, indicated by a SUCCESS (SCS) arc, when an action succeeds: otherwise control goes to the next action indicated by the sequence structure, Figure 3.

In Figure 3 a program to show that it is possible for an object named by a word x to perform an action named by word y, by finding an intersection in a general proposition. In Figure 4 the procedure of Figure 3 is shown having been executed on an appropriate declarative structure. The argument arcs point to the actual values rather than to the formal ones of Figure 3. If there were no intersection then "test" and "flag" would not have the same node as argument.

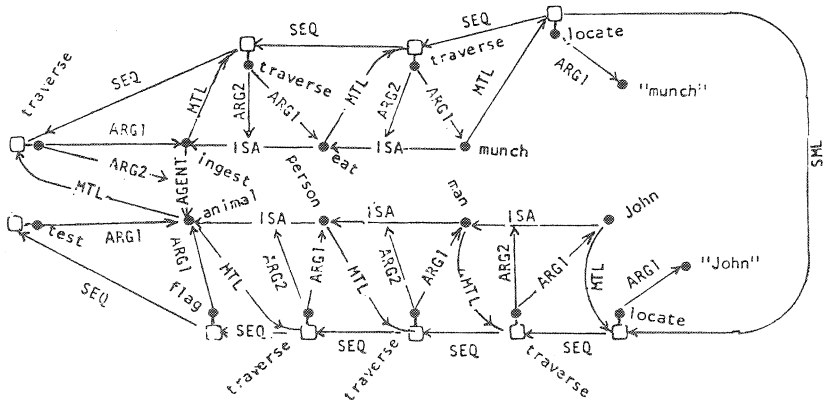


Figure 4 Application of a path-tracing routine

3.3 Procedural concepts

Analogy applies to items that have common attributes. The original choice of attributes is not part of the process but is determined by a higher level context. The attributes are characterized by a net, not a single node. The process is to find items that possess the attributes, using pattern-matching.

Or consider "euphemism" which is the substitution of an offensive word by a less derogatory one. Fortunately the ability to write functions enables the procedure to be written as a hierarchy of subfunctions rather than as a monolithic routine. The procedure for "euphemism" can be broken in three. First there is a search for a concept that has the bad connotations, then a less offensive equivalent is sought, and the original replaced. The sub-procedures can likewise be decomposed metalingually.

4. Discussion

At the time of writing only mathematical and path-tracing routines have been programmed.

The examples in Section 3 were preformed routines. A progessive step is to construct complex routines from basic actions. In selecting a plan one needs to know what it does to what. This is all present in the network; e.g., "traverse" would appear as in one of its instances in Figure 4 except for having general rather than specific arguments. By comparing the arguments of general actions with the specific problem, applicable actions can be found. This method can be extended to find complex plans. Such a plan can be compounded using the metalingual device. For example, a general form of Figure 4 can be treated in its entirety as a plan to establish agency. The initial and final situations can be characterized as, respectively, two disjoint nodes and then with an agent relation between them. In turn this agency plan could be part of a larger scheme.

The hierarchy enables a top-down approach to problem solving. A plan that covers the whole problem, but only in terms of less general plans not precisely, can be selected. Greater detail can be added by successively replacing general plans. If the plan cannot be completely formed this way, the gaps can be filled by a bottom-up use of primitive actions. Recall that these fragments are data structures during planning and can be manipulated to tie together variables to produce a plan in a format as given in Figure 3, say.

References

Ambler, A. P., Barrow, H. G., Brown, C. M., Burstall, R. M., & Popplestone, R. J. A versatile system for computer-controlled assembly. Artificial Intelligence, 1975, 6, 129-156.

Phillips, B. A representation for knowledge and its application to discourse analysis (Technical Report KSL-9). Chicago, IL: Knowledge Systems Laboratory, University of Illinois at Chicago Circle, 1977.

Quillian, M. R. The teachable language comprehender: A simulation program and theory of language. Comm. ACM., 1969, 12, 459-476.

Rieger, C. An organization of knowledge for problem solving and language comprehension. Artificial Intelligence, 1976, 7, 89-127.

Sacerdoti, E. D. A structure for plans and behavior (Technical Note 109). Menlo Park, CA: Artificial Intelligence Center, Stanford Research Institute, 1975.

Scragg, G. W. A structure for actions (Working Paper 20). Castagnola, Switzerland: Instituto per gli studi semantici e cognitivi, 1975.

Winograd, T. Frame representations and the declarative/procedural controversy. In D. G. Bobrow & A. Collins (Eds.), Representation and Understanding. New York: Academic Press, 1975.

A COMPUTATIONAL STUDY OF A PERIOD OF INFANT DEVELOPMENT

K. Prazdny
Computer Centre, Essex University, U.K.

Abstract

A period in the early human development ranging from about 12 to 20 weeks is described in terms of typical behaviours characteristic of this developmental stage. The behaviours are reassessed in terms of a computer model and it is argued that they are a result of the interaction between perceptual and conceptual levels of the system and the way in which a conflict between competing descriptions of the object attended to is resolved. As such, the rationale underlying the program suggests that it is the object and its event description which form the basic unit of analysis and not the infant's previous activity as argued by Piaget (Piaget, 1954), or the motion or places as argued by Bower (Bower, 1971).

Keywords

Object concept development, object identity, motion perception, developmental model.

Introduction

The nature of the object concept development is one of the most studied problems in the psychology of the infant. It takes a baby about 18 months to develop an adult-like concept of the object as being a permanent entity with an identity of its own. Piaget's work (Piaget, 1954) on object permanency has been replicated many times but his explanations of the phenomena were questioned frequently, especially by Gibson (Gibson, 1969).

Recently, Bower and his associates (Bower, 1971; Bower, 1975; Bower, 1977) performed a series of experiments which offer an opportunity to compare two successive development stages in computational terms because they are well documented (although not detailed enough) and centered around the transition between stage 2 and 3 of the object concept development. The concept of a developmental stage was introduced by Piaget and refers to a development period during which the development as reflected in the behaviour is stationary. The transition from stage 2 to stage 3 is an important one because it seems to result in the ability to identify objects by their features, a step of great significance which simplifies the perceptual world of the infant.

The experiments

The experiments and behaviours of infants in only Piaget's second stage of the sensorimotor period will be described because a model for the third stage has not been completed yet (although we anticipate it will be a straightforward modification of the model for the second stage). At the end of the second stage, the infants are between 12 and 20 weeks old, and the experimental situations are simple, involving at most 2 objects simultaneously. All movements are on planes with normals parallel to the view vector.

1. An object interesting to the infant (e.g., a toy) is seen together with a screen. The screen moves to occlude the object. Infants are surprised by the object's disappearance (after the screen had moved off), and do not show any sign of being disturbed or surprised by the object's reappearance.
2. An object is moving on a path (a straight line, or a circle). After a while the object stops in full view. The infants fixate the stationary

object for a while and then continue to follow the previously established path.

3. A moving object is occluded by a stationary screen. The infants continue to track it while it is occluded. They track the object when it reappears, and they follow its path when it does not reappear.
4. A moving object disappears suddenly. The infants continue to follow the path.
5. A moving object is mysteriously replaced by another object. The infants do not show any signs of surprise and continue to track the new object.
6. The same as in 5 but with a stationary object.
7. When the replacement in situations 5 and 6 is done while the object is invisible (behind a screen or in a dark interval) the infants are surprised and agitated (Gardner, 1971; Bower, 1977).
8. If one displaces a stationary object the infant will follow it and then look back to its original position.
9. A moving object disappears behind a screen. At the same moment another object appears at the point where the original object should reappear, and moves on the same path, with the same velocity. The infants are surprised and confused. Sometimes one can observe alternating glances between the reappeared object and the screen.
10. When a moving object suddenly changes its trajectory one may observe alternating glances between the object and the old path.

Bower (1975) proposed some hypotheses about the possible rules which may be used in interpreting the experimental situations described. He argues that an object's identity is determined by its place when it is stationary, and by its path of movement when it moves, and hence that all objects seen in the same place or on the same path of movement are the same object. A detailed analysis of his experiments shows, however, that this cannot be the whole story, and that no explanation which does not take the feature complex of an object into account can explain all the data. Consider for example, the discrepancy between observations 5, 6 and 7. When the replacement is done in full view the objects will apparently be regarded as one identical object. When the replacement is done out of view the identity will depend on the featural correspondence of the two objects. The identity of an object may thus be determined on two levels; on the perceptual level as in the situations 5 and 6, or on the more conceptual level, e.g., in a occlusion situation when the two occurences of the same object are separated spatio-temporally. When two spatio-temporally distinct objects are linked on a perceptual level they will be identified as one and the same object without a further check. When such a perceptual link is missing the identity attribution is an inferential process resting on the conceptual structures available.

A computer program that models the experimental findings

A working computer program has been constructed which models the behaviours described below. It is written in POPCORN (Hardy, 1975), a version of CONNIVER (McDermot and Sussman, 1972) using a pattern-structured data base. The inputs to the program are low-level symbolic descriptions of objects in a 'snapshot'. The set of successive 'snapshots' defines an experimental situation. The program's task is to interpret the situation, to incorporate a new snapshot into the description of situation formed so far, and to decide what to do after each snapshot has been integrated. The action repertoire of the program is not large. It consists of the ability to "track" an object, to be surprised, to "cry", and eventually to be confused beyond recovery.

The program consists of 3 "layers". The first layer is the low-level computation which distinguishes the fixed and moving objects in the program's "visual field". This is done by a "differencing" computation in which a new snapshot is compared with the interpretation of the situation so far. This is a self-contained module in which motion is computed in a shape-independent way. The computation is similar to the one advocated by, e.g., Lamontagne (1976). An object in the "visual field" is characterised by its position, its two-dimensional extension, and its feature list. The depth information is coded by the object's position on one of the two laminae separated in depth. The lamina separation is not quantified. An object so input to the program does not have an identity which relates it to the program's internal model of the situation. This is done by the motion computation and on the second level where an attempt is made to integrate assertions about an object with the relevant information in the data base. When an assertion is not used, or "attended to", its strength automatically decreases until it is eventually deleted from the data base. If an event fails to be assimilated within the second layer computation it will not be remembered.

The second layer computation is based upon the observation that it takes time for a concept to be developed. The intermediate-level assertions are monitored and when they are present for long enough a concept is developed describing an event in the world. The consequences of these actions are worked out at the third 'layer'. The whole can conveniently be viewed as computing descriptions of increasing abstraction. The fact that it takes time for a description to be developed accounts, for example, for the behaviour in situation 10. When the object attended to changes its path the system may notice the change (depending on its magnitude) and because the available description does not apply any more it will try to develop a new one. During this 'critical' period the decision about where to attend to is 'frozen' so that the system will attend to any object which inherits the identity (computed by the motion computation) of the original object. When the new description is developed two conflicting descriptions of the object will coexist in the data base. The system will try to resolve the conflict by investigating if both descriptions correspond to an object. This will result in an 'oscillatory' behaviour as the system alternates its 'attention' between points determined by both descriptions.

The program operates mainly by using a set offunctions (and demons) which are capable of expecting an event, i.e., which are capable of putting into the data base a condition-sensitive interrupt and of monitoring its activation. These expectations specify what to do if an event occurs within a specified time interval, and what to do if such an event fails to occur. When a demon embodying an expectation is activated it may be capable of putting another expectation into the data base, and so on. An expectation is implemented as a pair of one ifadded and one ifremoved method which may refer to each other directly (through a pointer), or indirectly (through a 'note' in the data base). The first method is the condition sensitive interrupt proper. The second one is triggered by the removal of the first one, i.e., by the failure of an occurence of the expected event within a certain time interval. Both methods are linked to an item in the data base which functions as a 'note' to the 'clock' mechanism. These 'notes' represent the 'age' of an expectation, i.e., time constraints on it. An expectation is activated (assembled and put into the data base) only when needed, i.e., its pattern and function body is determined by the context of the data base.

The small attentional capacity of the infant of this age is reflected

in the restriction that only one concept can be attended to at any one time, and that this object has to be in the centre of the "visual field". This means that only one description will be prevented from decaying. No check is performed on descriptions of objects specifying the same object with respect to the feature complex. This means that one and the same object may be represented as many non-identical objects, i.e., that many different descriptions of the same object may coexist in the data base at the same time.

The model offers insight into some 'mysterious' phenomena. For example, Bower (1967) reports that with high disappearance rates, the smaller the screen, the less likely the infants were to give permanence responses (in situation 1). He suggests that: "... as the dimension of the screen approached that of the desired object, the screen ceased to function as a screen and became instead a substitute object" (Bower, 1967). The model suggests why. Because of the high disappearance rate (velocity), the motion computation will pair the object with the screen in the 'differencing' computation involving a snapshot in which only the screen is visible as this pairing minimizes the overall 'perceptual' distances between objects in the two snapshots. Also, because of the lack of any partial occlusion information (due to high velocities involved), the system will not be able to infer occlusion. It will notice, however, that the screen disappeared and that the object changed its appearance.

Conclusion

It is perhaps apparent that the peculiar behaviour of the system is due to interactions between event descriptions and between the perceptual and conceptual information available. When a contradiction arises between two descriptions of an object the system discards information offered by the perceptual system in favour of established conceptual structures. Instead of explicitly updating its data base and merging the conflicting assertions the system prefers to treat the assertions as describing different objects, it prefers the conservative strategy of investigating if the old assertion does not apply to any object any more. It seems that in this stage the identical objects in two different 'snapshots' which are not linked by a perceptual activity or by an expectation become in fact two different objects.

The next stage seems to be characterized by the extensive use of the features for object identification. What are the structures characterizing this next stage? The Piagetian tradition identifies the transition from one developmental stage to another with the processes of accommodation. There is very little known about this process. New structures are formed out of the old ones, it maintains, but what structures, when and how is left unspecified.

The ultimate aim of this research is to characterize two successive developmental stages in terms of a computational model and to compare them. The basis of this approach is the view that the more precisely one can state the models of what the developmental stages are, the more precisely the transition process itself can be determined. It seems that while infants do have access to a wide range of information, they do not utilize it fully, and so, it is not the available information that determines the behaviour, but the cognitive structures by which reality is transformed.

Acknowledgement

My thanks to Drs P.J. Hayes and J.M. Brady for encouragement and discussion. This work was done while the author was a British Council Research Fellow.

References

Bower, T.G.R.; The development of object permanence, Perc. & Psychoph., 1967, 2, 411-418.

Bower, T.G.R.; The object in the world of the infant, Scientific Am., 1971, 30-38.

Bower, T.G.R.; Infant perception of the third dimension and object concept development, in Cohen & Salapatek (eds), Infant Perception (vol. 2), Acad. Press, 1975.

Bower, T.G.R.; A primer of infant development, Freeman, San Francisco, 1977.

Gardner, J.K.; The development of object identity in the first 6 months of human infancy, Soc. for research in child behaviour, Minesota, 1971.

Gibson, J.J.; The change from visible to invisible, Perc. & Psychphys., 1969, 5, 113-116.

Hardy, S.; The POPCORN reference manual, Essex University, 1975.

Lamontagne, C.; Visual motion detection, AISB conference proceedings, 1976.

Piaget, J.; The child's construction of reality, New York, Basic Books, 1954.

McDermot, D.V., Sussman, G.J.; The CONNIVER reference manual, AIM-259, MIT, 1972.

PARAMETERIZED REGION EXTRACTION FOR THE DESCRIPTION OF MOVING OBJECTS

Bernd Radig

Fachbereich Informatik, Universität Hamburg
Schlüterstrasse 70
D-2000 Hamburg 13, W.Germany

ABSTRACT

A sequence of TV-frames is recorded from a real-world scene. In order to determine motion attributes of objects the segmentation of these (digitized) TV-frames has to provide a reliable symbolic description for each image. The complexity of light intensity distribution along object surfaces in such scenes calls for a method which can extract regions according to various criteria. The algorithm presented here transforms the grayvalue matrix into a vector field, the vector components describing small local planes fitted to the grayvalue image. A three-level filter process uses parameterized constraints to remove inconsistent nodes and arcs from a graph which represents the vector field. Nodes are associated with vectors, arcs are initialized to connect adjacent vectors. A region is associated with an isolated graph component, its features are computed from the vectors in the component. Comparison of features establishes correspondence of regions originating from the same object surface in different frames. These region sequences serve as a basis for object description, especially its motion state.

INTRODUCTION

The techniques for single image interpretation - especially of artificial scenes [Winston 75] - are quite well understood. New problems arise if the interpretation goal requires to compare several images. Two images generated by binocular stereo vision may be used to get insight into the depth structure of the scene [Levine et al. 73, Hannah 74]. To build a three dimensional model of an object, it may be viewed from stepwise varying angles [Nevatia 76, Baker 77]. One may also compare images in order to detect changes in the optically perceivable environment [Price&Reddy 77]. Observation of moving objects in a sequence of TV-frames constrains such changes to systematic and continuous ones. To circumvent the complication of image segmentation while investigating the occlusion problem, some authors used artificial objects [Aggarwal&Duda 75, Chow&Aggarwal 77, Martin&Aggarwal 77]. Others omitted the segmentation phase and started from symbolic primitives [Badler 75, Tsotsos 76]. One may also delay segmentation until some motion attributes are computed at the grayvalue level [Potter 77, Jain&Nagel 77]. Separation of stationary and nonstationary image components has become sufficiently reliable for real world scenes, so it may serve as a powerful guide for further analysis [Nagel 77]. An opposite approach attempts to interpret each frame of a sequence independently and compares the symbolic descriptions at the most abstract level [Bertelsmeier&Radig 77]. A bottom-up analysis of the first few frames increases the accuracy of a scene model which specifies configuration and motion attributes of the objects observed in the evolving scene. As a predictive model it speeds up the top-down analysis of subsequent frames. Experiments with this system revealed that an unstable, unreliable segmentation and shape description burdens the interpretation process inadequately.

The segmentation technique described hereafter was developed to allow the comparison of low level symbolic image descriptions. The approach requires the segmentation to be sufficiently insensitive to disturbances like video-signal noise, scan jitter, changing illumination and shadowing, varying light reflexion, perspective distortion, partial occlusion, etc. Disregarding these difficulties, the light intensity distribution in natural scenes is complicated by itself. Horn and Woodham employed the rules of photometry to support analysis of object surfaces [Horn 77, Woodham 77]. Their results may explain only local intensity distributions in natural scenes, but gave the hint that segmenting an image into more or less homogeneous regions, or regions which are bordered by strong grayvalue discontinuities does not seem to be always an adequate tool. Some algorithms are able to extract regions having an internal grayvalue distribution of nonzero slope [Yakimovsky&Cunningham 76, Somerville&Mundy 76]. Others examine direction and magnitude of the gradient [Yoda et al. 75, Chien&Jacobus 76]. In my algorithm not the gradient but small planes fitted to the pixel matrix (see also [Holdermann&Kazmierczak 72]) serve as a basis for region extraction. Since planes are described by mean grayvalue, direction and magnitude of slope, a considerable amount of intrinsic structure may be incorporated into region descriptions. The segmentation results allow to compare regions from different frames without relying on disturbable shape description or other geometric properties. By choosing different segmentation parameters one can extract homogeneous regions as well as regions which are characterized by a strong gradient or a systematic behaviour of the gradient direction.

PREPROCESSING

Our digitization apparatus samples the video signal coming from a commercial black&white camera in a matrix of 573 lines by 512 columns of 8 bit bytes. Overlapping submatrices (5 by 4) are used for a least-square fit of a plane. Each plane represents a rectangle of 3 by 2 pixels. The plane normal vectors are collected in an array of 191 lines and 256 columns. Since a TV-frame represents an aspect ratio of 3 by 4 (standard CCIR norm), each vector represents a square of the projected image. The resulting vectors are coded by (fig. 1)

- G, the mean grayvalue (8 bit)
- I, the inclination (degrees) against the grayvalue axis (3 bit), 0 for horizontal planes describing constant intensity
- D, the angle (degrees) between the projection of the normal vector onto the coordinate plane and the scan direction (5 bit).

The data volume of each image is reduced from 573*512 ≃ 300000 bytes to 191*256*2 ≃ 100000 bytes. The CPU-time for this reduction is below 100sec on a DEC-KI10 processor.

Fig. 2 shows a car scene and fig. 3 the vector inclination component coded dark. Fig. 4 shows the vector representation of a section containing the right wheel. Each vector is plotted as a little square coloured with the grayvalue G. Each square is intersected by a stripe the darkness of which is related to the inclination I (white=zero inclination). The stripe orientation corresponds to the vector direction, rotated by 90° so that the stripes line up along contours. (The vectors themselves are directed from dark to light.)

CONSTRAINT FILTER

The vector field is transformed into a graph where the vectors constitute the nodes, and arcs are initialized as the (undirected binary) adjacency relation.

$$\text{NODES} = \{\ V = (Y,X,D,I,G)\ \}$$
$$\text{ARCS} = \{\ (V_1,V_2)\ |\ 0<(Y_1-Y_2)^2+(X_1-X_2)^2<3\ \}$$

A set of pameterized constraints controls the removal of nodes and arcs from the graph. Simple constraints may be defined by restricting the range for the magnitudes of vector components. After the first filter pass only those vectors remain in the graph whose component values are in the allowed range.

$$\text{NODES} = \{\ V\ |\ \overline{\forall}_C\ f_C(V)=1\ \},\quad f_C(V) = \begin{cases} 1 & <=>\ C\in[Cmin,Cmax] \\ 0 & \text{otherwise} \end{cases}$$
$$"C"\in\{Y,X,D,I,G\}$$

The discriminating functions f_C may be replaced by others depending on single components or combinations thereof; the product may be replaced by some fuzzy-logic threshold operation. The removal of nodes is accomplished by cutting all arcs leading to other nodes, singular nodes are discarded. The graph resulting from this first pass is called node-consistent. Fig. 5 shows a graph covering the same image section as fig. 4. The constraints are $Y\in[115,163]$, $X\in[65,128]$, $I\in[63,90]$.

The next filter pass establishes arc-consistency (see [Mackworth 77]). Here arcs are removed if they connect vectors whose component values do not fulfill the conditions of a parameterized binary relation. The binary relation reflects component differences:

$D_G = |G_1-G_2|$, is the grayvalue difference.

$D_D = \min(|D_1-D_2|,\ 360^o-|D_1-D_2|)$, is the difference of direction, max 180^o.

Let $L'=\arctan(|Y_1-Y_2|/|X_1-X_2|)$ be the angle of the line connecting the coordinate positions of two vectors and $L_i=\min(|D_i-L'|,\ 360^o-|D_i-L'|)$, $i=1,2$ the difference between that angle and the direction of vector V_i, then is

$D_M = \min_i(L_i,\ 180^o-L_i)$, $i=1,2$ the smallest angle between the vectors and the line defined by L'.

$D_S = \min(L_1,\ 180^o-L_1) + \min(L_2,\ 180^o-L_2)$ is the sum of both angles.

Using again a simple discriminator function we get

$$\text{ARCS} = \{\ (V_1,V_2)\ |\ \overline{\forall}_D\ f_D(V_1,V_2)=1\ \},\ \text{with}$$

$$f_D(V_1,V_2) = \begin{cases} 1 & <=>\ D\in[Dmin,Dmax] \\ 0 & \text{otherwise} \end{cases}$$

$$"D"\in\{D_G,D_D,D_M,D_S\}$$

Some examples illustrate the influence of interval limits for a graph covering the same image section as fig. 5.

Fig.6: $G\in[0,255]$, $D\in[0,360]$, $I\in[45,90]$,
$D_G\in[0,255]$, $D_D\in[0,45]$, $D_M\in[0,45]$, $D_S\in[0,56]$

Whereas the arcs in fig. 6 have directions following edges bordered by contrasting areas, perpendicular arc directions are shown in

Fig.7: $G\Theta[0,255]$, $D\Theta[0,360]$, $I\Theta[45,90]$,
$D_G\Theta[0,255]$, $D_D\Theta[0,34]$, $D_M\Theta[34,90]$, $D_S\Theta[67.5,180]$

Regions of irregular vector orientation are extracted by

Fig.8: $G\Theta[0,255]$, $D\Theta[0,360]$, $I\Theta[0,90]$,
$D_G\Theta[0,255]$, $D_D\Theta[34,180]$, $D_M\Theta[34,90]$, $D_S\Theta[45,180]$

The second phase ends with regions consisting of isolated subgraphs. A fast, non-recursive algorithm visits each vector of the subgraph and evaluates region features from vector component values. The most important features are:

- AREA = number of vectors
- ARCS = number of arcs (ARCS/AREA=arc density)
- $\sum X$, $\sum Y$ = coordinate sums (center of mass)
- $\sum G$, $\sum G^2$ = grayvalue (mean and variance)
- $\sum I$, $\sum I^2$ = inclination (mean and variance)

GHIST[0..31] = grayvalue histogram (condensed to 32 intervalls)
DHIST[0..31] = direction histogram (11.25° resolution)

The third filter pass uses AREA and both histograms to modify the region result establishing something like global consistency. An AREA interval is given so that regions too small or too large may be suppressed (by deleting all arcs in the corresponding subgraphs). In figs. 6 to 8 subgraphs having less than 5, 10, 10 nodes, respectively, are already suppressed. Two other constraints may concern the shapes of the grayvalue and direction distributions. In the following examples the ratio of the highest to the lowest (>0) histogram value is restricted. All grayvalues or direction values where the histogram frequencies are below the threshold determined by the histogram peak and the ratio are marked as forbidden. The regions where such forbidden values occur are visited again, and all arcs between two vectors one of which has a forbidden and the other an allowed component are cut. That may separate the region into two or more isolated subgraphs. The features are reevaluated for each subgraph, which subsequently has to undergo the third filter pass, too. In fig. 5 the only restriction was $I\Theta[63,90]$, in fig. 9 the ratio for both histograms is 5 to 1 and in fig. 10 it is 2.5 to 1.

The CPU-time used in the filter process including evaluation of region features does not exceed 1msec per vector, yielding a total of less than 3sec per frame for the demonstrated examples.

MOTION DETECTION

To test the applicability of this segmentation technique to motion description, traffic scenes with moving cars have been recorded and processed. The following example deals with 66 TV-frames covering 2.6sec of real time. With appropriately chosen parameters (not altered within the sequence) they are segmented into about ten regions per frame which describe the moving car and parts of the stationary background. The

region descriptions of all frame pairs adjacent in time are compared in order to combine similar regions to sequences. The similarity measure is obtained by comparing the region features one by one, and combining the results - numbers between 0 and 1 - by means of a fuzzy-AND, here the minimum. Fig. 11 shows the black center-of-mass traces of sequences of similar regions. The picture is composed of the first and the last video frame showing the car at the start and the end position. Subsequent processing connects incomplete region sequences by measuring the similarity between head and tail which may be several frames apart. The resulting connections are plotted as white lines. This way the gap in the car-shadow trace caused by the occluding tree-top is bridged.

CONCLUSION

A three level constraint filter has been proposed which modifies a graph associated with a vector field transformation of an image. It uses 3 parameter pairs at level 1, 4 at level 2, and 3 at level 3. All parameters have an intuitively understandable meaning which is advantageous for their adjustment. Various examples have been presented to demonstrate the influence of parameter settings.

The usefulness of direction information is shown in the last example (fig. 12). There two regions are extracted in a subsection of the image containing the car and its shadow. The segmentation parameters have been chosen to filter regions of collinear vectors with strong inclination. Due to the grayvalue slope which increases towards the bottom of the car and decreases towards the outer shadow where it merges into the street, car side and shadow are perfectly separated. The wheels are not included in either region. The separation succeeds in all 66 frames of the moving-car example. The segmentation is reliable enough to enable forming of region sequences in a simple way, even allowing to bridge gaps in time when an object (-part) is not visible due to occlusion.

ACKNOWLEDGEMENT

I would like to thank H.-H. Nagel and B. Neumann for their useful suggestions. B. Neumann helped with his comments on the manuscript. Hardware and software supporting this work have resulted from the combined efforts of R. Bertelsmeier, P. Cord, I. Heer, H. Kemen, H.-H. Nagel and B. Radig.

REFERENCES

[Aggarwal&Duda 75] J. K. Aggarwal, R. O. Duda: Computer Analysis of Moving Polygonal Images; IEEE Trans. Comp. C-24 (1975) 966-976

[Badler 75] N. I. Badler: Temporal Scene Analysis: Conceptual Descriptions of Object Movements; Dissertation, Report No. 80, Univ. Toronto, Febr. 1975

[Baker 77] H. H. Baker: Three-Dimensional Modelling; IJCAI-77, 649-655

[Bertelsmeier&Radig 77] R. Bertelsmeier, B. Radig: Context Guided Analysis of Scenes With Moving Objects; Bericht IfI-HH-B-41/77 Institut für Informatik, Univ. Hamburg, Febr. 1977

[Chow&Aggarwal 77] W. K. Chow, J. K. Aggarwal: Computer Analysis of Planar Curvilinear Moving Images; IEEE Trans. Comp. C-26 (1977) 179-185

[Chien&Jacobus 76] R. T. Chien, C. Jacobus: Directional Derivatives in Computer Image Processing; IJCPR-76, 684-688

[Hannah 74] M. J. Hannah: Computer Matching of Areas in Stereo Images;

Dissertation, Memo AIM 239, Stanford Univ., July 1974
[Holdermann&Kazmierczak 72] F. Holdermann, H. Kazmierczak: Preprocessing of Gray-Scale Pictures; Comp. Graphics and Image Processing 1 (1972) 66-80
[Horn 77] B. K. P. Horn: Understanding Image Intensities; Artificial Intelligence 8 (1977) 201-231
[Jain&Nagel 77] R. Jain, H.-H. Nagel: Analysing a Real World Scene Sequence Using Fuzziness; IEEE Conference on Decision and Control, New Orleans, Dec. 7-9, 1977
[Levine et al. 73] M. D. Levine, D. A. O'Handley, G. M. Yagi: Computer Determination of Depth Maps; Computer Graphics and Image Processing 2 (1973) 131-150
[Mackworth 77] A. K. Mackworth: Consistency in Networks of Relations; Artificial Intelligence 8 (1977) 99-118
[Martin&Aggarwal 77] W. N. Martin, J. K. Aggarwal: Dynamic Scene Analysis: The Study of Moving Images; Techn. Report 184, Inf. Syst. Research Lab., The Univ. of Texas at Austin, Austin, Texas, Jan. 1977
[Nagel 77] H.-H. Nagel: Analyzing Sequences of TV-Frames: System Design Considerations; Bericht IfI-HH-B-33/77, Institut für Informatik, Univ. Hamburg, March 1977, & IJCAI-77, 626
[Nevatia 76] R. Nevatia: Depth Measurement by Motion Stereo; Computer Graphics and Image Processing 5 (1976) 203-214
[Potter 77] J. L. Potter: Scene Segmentation Using Motion Information; Computer Graphics and Image Processing 6 (1977) 558-581
[Price&Reddy 77] K. E. Price, R. Reddy: Change Detection and Analysis in Multispectral Images; IJCAI-77, 619-625
[Somerville&Mundy 76] C. Somerville, J. L. Mundy: One Pass Contouring of Images through Planar Approximation; IJCPR-76, 745-748
[Tsotsos 76] J. K. Tsotsos: A Prototype Motion Understanding System; Master Thesis, Dept. Comp. Sc., Techn. Report TR 93, Univ. Toronto, June 1976
[Winston 75] P. H. Winston (ed.): The Psychology of Computer Vision; McGraw-Hill, New York, 1975
[Woodham 77] R. J. Woodham: A Cooperative Algorithm for Determining Surface Orientation from a Single View; IJCAI-77, 635-641
[Yakimovsky&Cunningham 76] Y. Yakimovsky, R. Cunningham: On the Problem of Embedding Picture Elements in Regions; Techn. Memo 33-774, Jet Propulsion Lab., Cal. Inst. of Technologie, Pasadena/Cal., 1976
[Yoda et al. 75] H. Yoda, J. Motoike, M. Ejiri: Direction Coding Method and its Application to Scene Analysis; IJCAI-75, 620-627

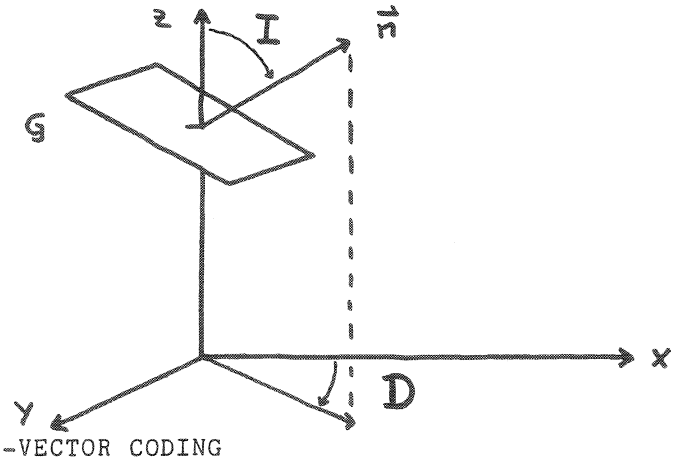


FIG. 1 PLANE-VECTOR CODING

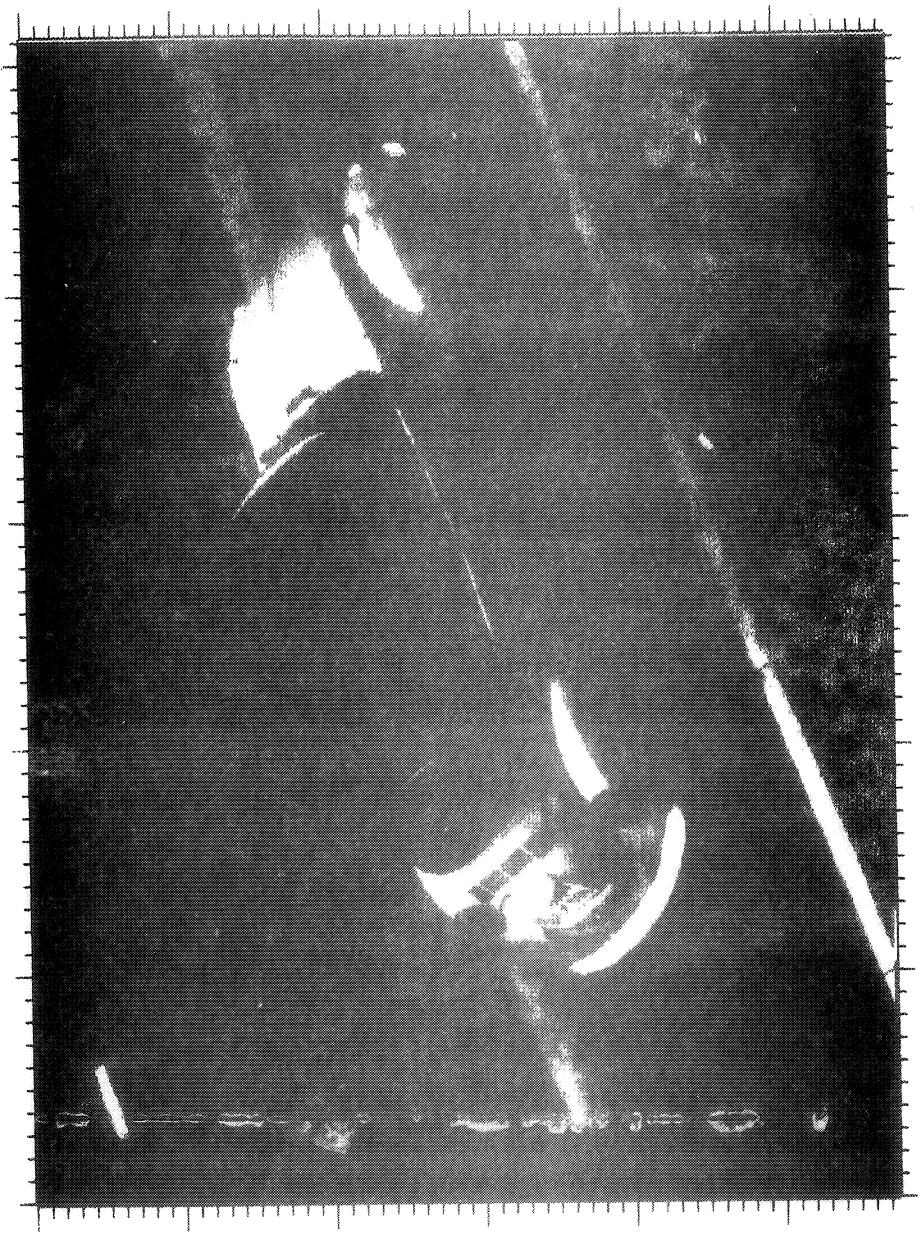
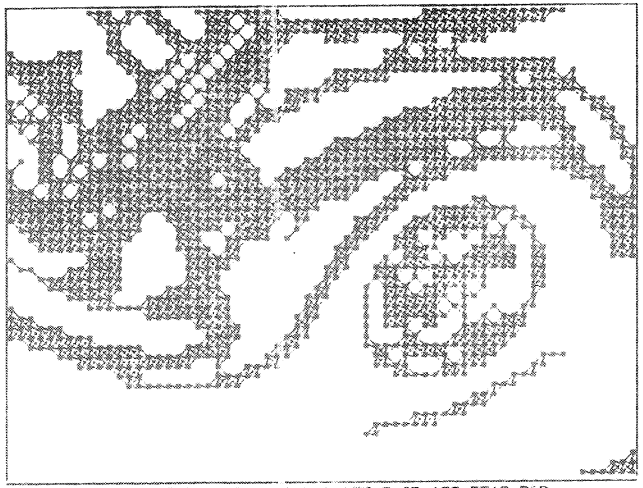
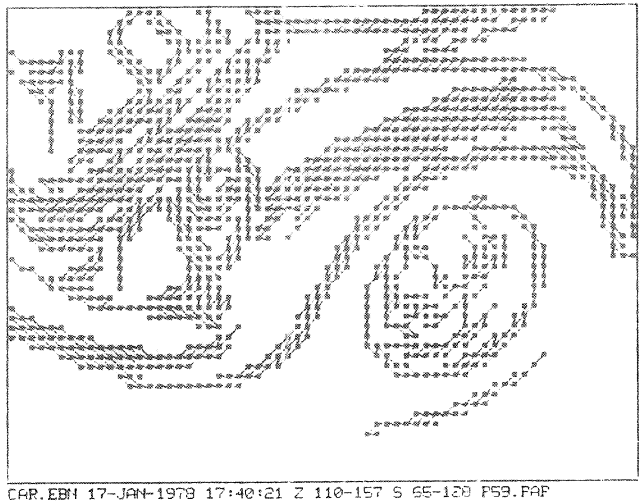


FIG.2 CAR SCENE



CAR.EBN 17-JAN-1978 18:8:31 Z 110-157 S 65-128 PS12.PAP

FIG.5 GRAPH OF VECTORS WITH STRONG INCLINATION



CAR.EBN 17-JAN-1978 17:40:21 Z 110-157 S 95-120 PS8.PAP

FIG.6 GRAPH OF (MODERATE) COLLINEAR VECTORS

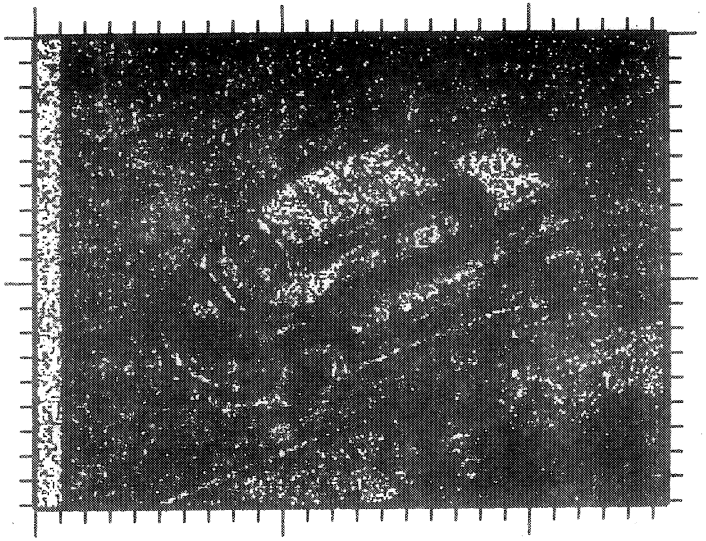


FIG.3 GRAYVALUE CODING OF INCLINATION (DARK=STRONG)

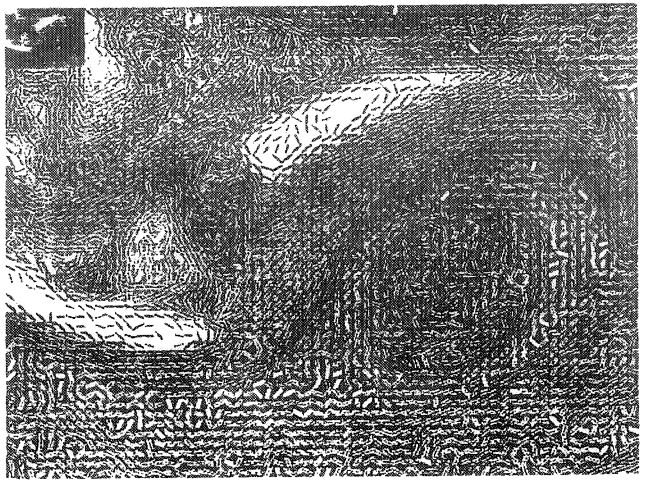
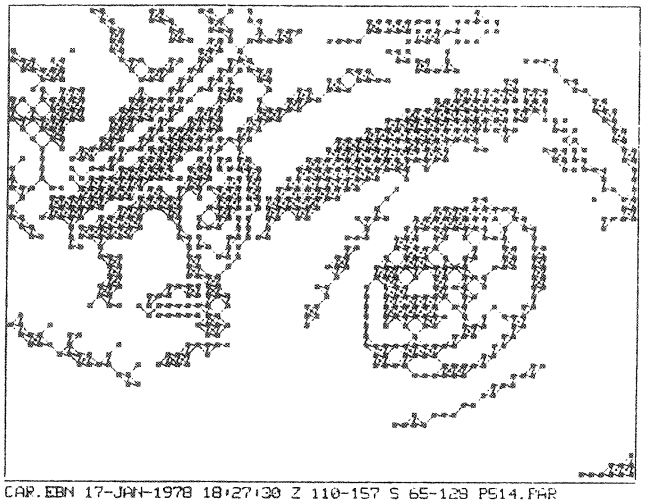
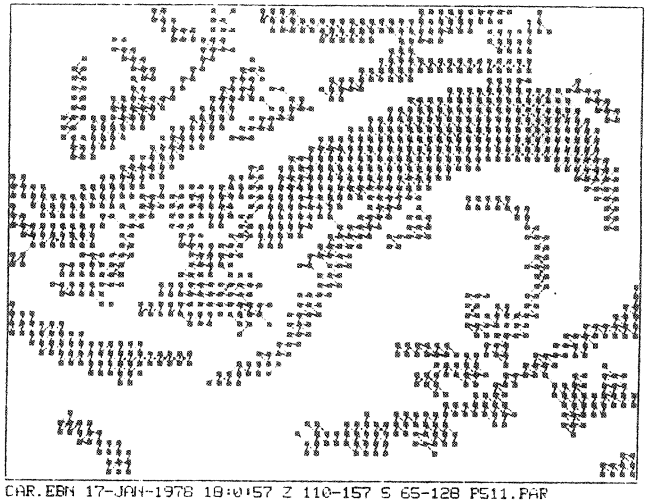


FIG.4 VECTOR REPPESENTATION OF AN IMAGE SUBSECTION



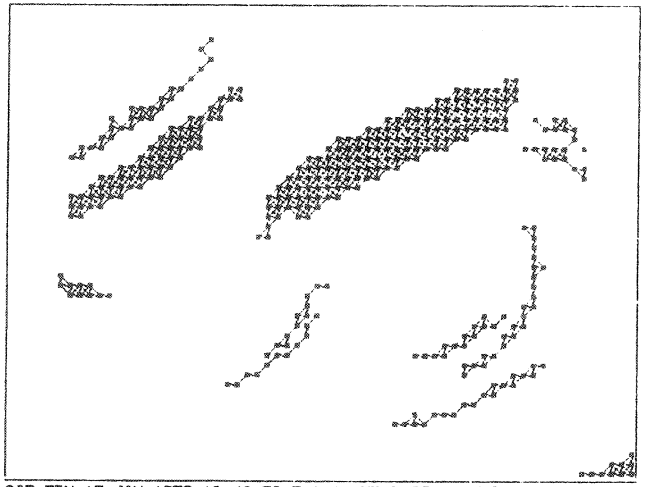
CAR.EBN 17-JAN-1978 18:27:30 Z 110-157 S 65-128 PS14.PAR

FIG.9 FREQUENCY RATIO MAX. 5/1 IN HISTOGRAMS



CAR.EBN 17-JAN-1978 18:0:57 Z 110-157 S 65-128 PS11.PAR

FIG.7 GRAPH OF (MODERATE) PARALLEL VECTORS



CAR.EBN 17-JAN-1978 18:46:58 Z 110-157 S 65-128 PS16.PAR

FIG.10 FREQUENCY RATIO MAX. 2.5/1 IN HISTOGRAMS

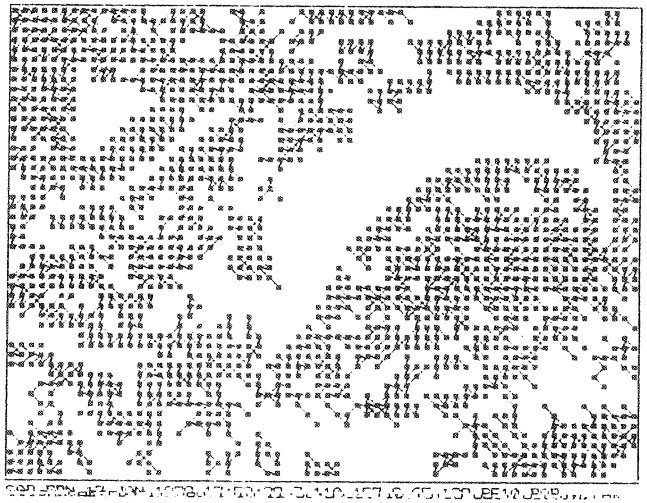


FIG.8 GRAPH OF RANDOMLY ORIENTED VECTORS

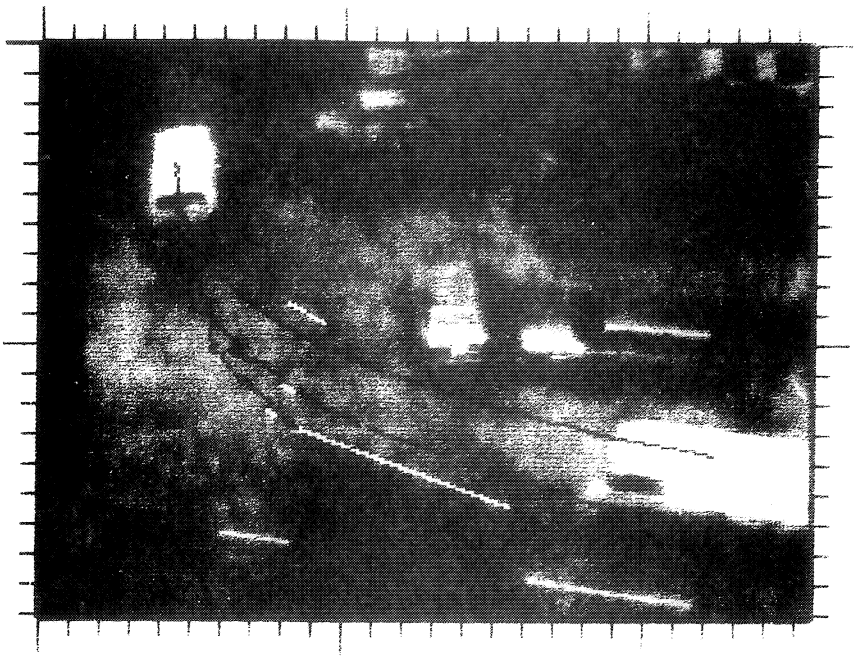
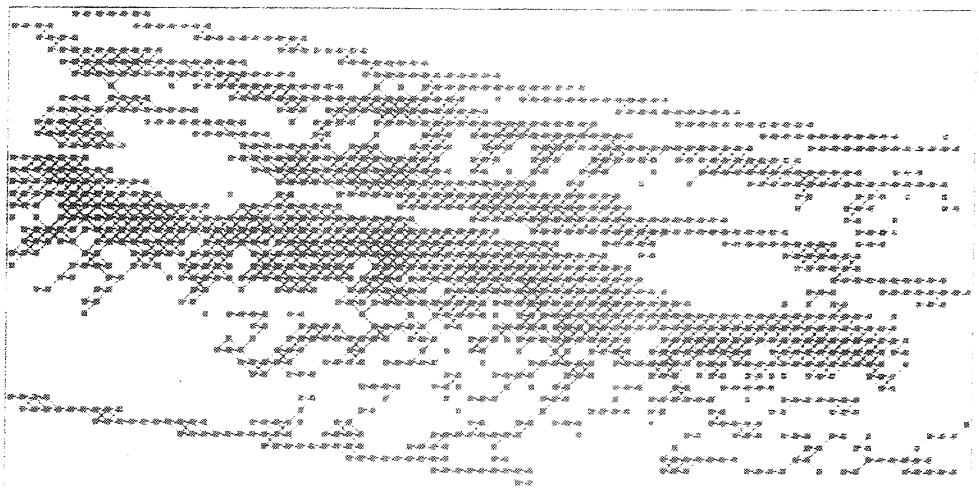


FIG. 11 IMAGES OF MOVING CAR AT [illegible]



WUBUS3.EBN 17-JAN-1979 [illegible]

FIG. 12 SEPARATION OF CAR AND ITS SHADOW

PREDICTIONS AND PROCEDURES IN SEMANTICALLY-BASED GRAMMAR

Graeme D Ritchie

Department of Computer Studies, Napier College of Commerce and Technology, Edinburgh, Scotland

ABSTRACT

It has been suggested that grammars for analysing English sentences should be based on semantic constructs. We examine what this might mean, focussing on the questions of semantic predictions and of processing procedures. There seems to be no general way of organising processing units around semantic categories, or of relating semantic predictions to specific processing units. We conclude that processing rules cannot be expressed in terms of referential semantics, and that top-down predictions can do little to drive the actual processing of words.

DESCRIPTIVE TERMS

Computational linguistics, syntax and semantics, grammatical categories, sentence processing.

I : Preliminaries

Most of the work done in natural-language processing in the 1960s was based on the idea of a wholly syntactic grammar, with little attention being paid to semantic theories of language. This reflected a similar trend within transformational linguistics during the first half of the decade. In the 1970s both linguistics and artificial intelligence have concentrated to a much greater extent on theories of meaning, and traditional syntax has been forced into a secondary position. In this paper, I wish to suggest that there has been an excessive over-reaction in this area, and that certain currently fashionable notions of 'semantically-based grammar" are not workable in the long term. This is not an argument in favour of traditional syntax, but rather an attempt to prevent an inadequate system (autonomous syntax) being replaced with an equally unrealistic extreme (autocratic semantics).

A few warnings are in order concerning the arguments I shall present here. I may sometimes assert that a particular arrangement "is not possible" within some system, or that certain devices "cannot" carry out some task. These assertions are subject to certain basic assumptions concerning theories of language, as follows. Every computational model has a level at which it can properly be regarded as a theoretical proposal, as distinct from a set of (largely irrelevant) details of the particular implementation. Also, there is some common, but informal, standard of "elegance" or "cleanliness" to which we should adhere when assessing the potential of any mechanism. Hence, when I say that mechanism X "cannot" perform some task, I tacitly assume that we are considering the appropriate level of theoretical description for mechanism X, and that we are ruling out devious ad hoc manoeuvres. These provisions are essential at present, since most computational

models are based on particular programs (and hence there may be confusion between hypothesis and implementation), and most of these models can allow arbitrary computations (unless we rule out those which are "clumsy").

The arguments presented here may seem trite, but they are not directed at straw men. In order to particularise the issues, I shall, in Section III, illustrate my points using proposals which have actually been made within AI circles in recent years. The central issues involve the use of semantic constructs in specifying a sentence analyser, and the question of how to use high-level, semantic (or "conceptual") information to aid the analyser in its task of processing the words of a sentence. I hope to persuade the reader that, while AI models have tended to suggest that these questions are easy to solve, the problems are very difficult, or even impossible. This is not to suggest that semantic information cannot, or should not, be used profitably during sentence analysis; semantic interpretation and inference should be deployed wherever this can aid the processing. The two questions considered here are: what linguistic categories are appropriate for specifying a grammar for sentence-analysis, and how can semantic predictions assist in the task of actually processing words?

II : Categories and Levels

The various classes and levels of classification used in a linguistic model are not merely notational conveniences - they define the skeleton of the model, and indicate what generalisations are being made concerning otherwise unrelated items. It is important in developing a general theory of language processing to ensure that the structure of the model is clear, and that some specification is given of how the categories are to relate to each other.

We can distinguish (at least) four different kinds of linguistic information which may be used within a sentence-analyser, as follows:

(A) Concrete Syntax: This describes the surface syntactic configuration, in terms of sequences of concatenated words or phrases, that forms an item. Traditional phrase-structure rules like (1) (i) and (ii) describe the concrete syntax of their left-hand sides.

(1) (i) <sentence> : := <noun phrase> + <verb phrase>
(ii) <noun phrase> : := <determiner> + <common noun>

(B) Referential Semantics: This describes what sort of thing the phrase refers to, and is not normally applicable to all kinds of phrase (unless some generalised form of "extension" is defined, as in Montague (1970)). In this classification, categories like "ANIMATE", "HUMAN", etc., are appropriate. For example, (2) (i) and (ii) might both be classified as referring to ANIMATE things, and (3) (i) and (ii) as referring to "physical objects".

(2) (i) The tall man who we met last night.
(ii) Your hamster.

(3) (i) The thing you broke.
(ii) His pencil.

(C) Abstract Syntax: This kind of description is on the boundary of syntax and semantics, since it classifies the constituents of linguistic items, and the way that they fit together into a semantic structure. The "templates" of Wilks (1973, 1975) and the "cases" of Fillmore (1968) make statements at this level, by defining how semantic items may combine to form larger semantic units.

Notice that this level gives rise to two ways of describing any linguistic item in its context. We can categorise the internal structure of the item; for example, a "proposition" might be described as being formed by a "predicate" being applied to a "term". Alternatively, we can discuss the external structure surrounding the item; for example, the underlined phrase in (4) might be classed as acting as a "subject" to the rest of the sentence.

(4) The dog leapt into the street.

(D) Type of Semantic Structure: This describes the semantic properties of the meaning itself (as opposed to what it refers to), thus defining how it can be processed semantically. For example, the semantic structures for the phrases in (2) above could be classed as "terms" or "definite referring expressions". To state that an item was a "relation", or a "predicate" would be a classification of this kind.

There are two levels, within a possible model of sentence processing, that are particularly relevant here, since they are directly involved in the organisation of the grammar. These are the level at which an analyser can, on semantic/conceptual grounds, predict that some kind of item is imminent in the input, and the level at which particular analysing procedures process the actual words in the input. Although these two levels are logically distinct, few computational models allow for this distinction explicitly, and most researchers seem to assume (tacitly, perhaps unconsciously) that both functions (high-level predictions and low-level procedures) can be handled at a single level. This will be illustrated, in Section III below, by considering a few of the possibilities embodied in recent proposals.

III : Predictions and Procedures

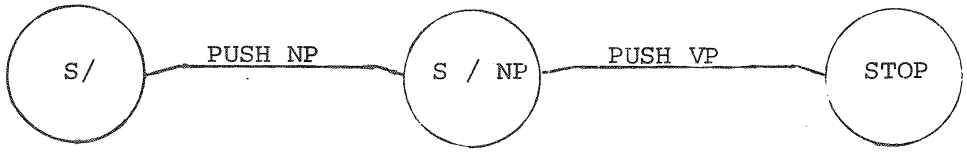
Let us consider whether any single one of the classificatory levels given, in Section II above, is adequate for describing both semantic predictions and processing procedures. To avoid the impression of attacking imaginary opponents, I will, wherever possible, cite actual proposals of models which seem to involve the hypotheses under consideration.

(A) Concrete Syntax

This is the level at which processing procedures must operate - actually examining input words in order, and sorting them into constituents in some way. It is rare for an English grammar to generate a prediction of a particular kind of item and of the words that it will include. Traditional analyses described most phenomena at the level of concrete

syntax, and this was carried over to AI models in various ways, including the "augmented transition network" (ATN) grammars (e.g. Woods et al. (1969), Bobrow and Fraser (1969), Kaplan (1971)). ATN grammars tended to include networks like figure 1.

Fig. 1



An arc like (PUSH NP) simultaneously expresses a prediction (that a particular kind of phrase will be found in the input) and specifies a procedure (the "NP" network) which is to be used to process that phrase (see Ritchie (1978) for a fuller discussion). This style of grammar survived for some time, partly because so many disparate items were forced into these established categories. A "noun phrase" may be a category at the level of concrete syntax, but it is not a very illuminating one. A noun phrase can take a variety of forms, as illustrated in (5).

(5) (i) John left.
(ii) Eating wombats is wrong.
(iii) To eat wombats is wrong.
(iv) More than ten of his many antique hot water bottles were stolen.
(v) That he spoke to you is surprising.
(vi) What you want to do is difficult.

This means that, although "PUSH NP" gives the sentence-analyser a starting point from which to process the constituent, this starting point is merely a label for a disjoint collection of quite varied options. As I have argued elsewhere (Ritchie (1977, Chapter III), a grammar which simply itemises all the options for the purposes of exhaustive search is theoretically undesirable. (Compare the comments by Lakoff (1971) on the uninteresting nature of a disjunction in the statement of a grammatical rule). We can continue to organise our processing units round such heterogeneous concrete syntactic categories as "NP", but in the long term something neater is needed.

(B) Referential Semantics

This level seems to be the closest approximation to the "conceptual" descriptions of Schank (1972), Riesbeck (1973, 1974, 1975). Riesbeck regards "conceptual" information (as opposed to "syntactic" information) as the main substance of a sentence analyser, and he seems to suggest that most of the problems of language understanding can be overcome by appropriate use of "context". While it may be tenable that conceptual information is useful (but not necessarily sufficient) in resolving certain potentially ambiguous constructions, it is definitely not true that conceptual information (whether contextual or within the sentence) is sufficient to guide the analyser through actual English words.

Let us take a simple, or even trivial, example. The "context", to talk in terms of Riesbeck (1973), is that of being in a bar, drinking. The input utterance starts as in (6).

(6) I like...

This utterance might continue in any one of several ways, for example those in (7). The analyser would have to process vastly different sequences of words in these cases, and the "context" does not help it much in this task.

(7) (i) ...this pub.
(ii) ...peanuts.
(iii) ...McEwans Export.
(iv) ...that girl we met last night.

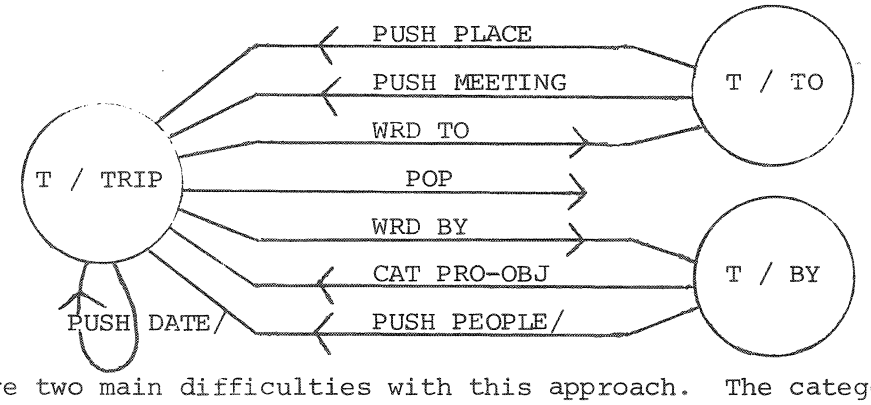
Even if the "context" were much narrower, say, the consumption of peanuts, the analyser would be given little help in working its way through the surface strings in (8), even though some of them have very similar meanings.

(8) (i) ...eating peanuts.
(ii) ...to eat peanuts.
(iii) ...these peanuts.
(iv) ...crisps as well as peanuts.

Conceptual predictions may be generated at the level of referential semantics, but the actual words used are a matter of concrete syntax. English has no obvious patterns relating referential semantic classes to concrete syntactic categories; there are not, for example, different syntactic constructions corresponding to "ANIMATE" and "INANIMATE" items. Riesbeck does not suggest how this gap may be bridged.

Woods et al. (1976) do make certain suggestions which seem to be based on the use of referential semantic categories for organising the analyser. They give examples of ATN grammars like Figure 2, where old-fashioned syntactic categories (e.g. NP) have been replaced by referential semantic categories.

Fig. 2



There are two main difficulties with this approach. The categories are carefully chosen for the subject matter of the English to be analysed, and hence are not generally applicable (as Woods et al. admit). If we wish to adopt grammars of this style for all of English, there would be as many different analysing grammars as there were domains of discourse.

Also categories like "MEETING" or "TRIP" do not, as observed above, correspond to particular concrete syntactic patterns, and so cannot index precise procedures for analysing the input.

Despite these attempts by Woods et al. to force referential semantics into use for both predictions and procedures, the technique appears to be ad hoc.

(C) Abstract Syntax

1. Internal.

There seem to be few, if any, examples in English where there is a prediction of a particular internal semantic structuring (notice the contrast with (C)2. below). It is rare for grammatical predictions to constrain the way that the incoming item is to be internally composed, even if they constrain the resulting meaning. The only suggestion to this effect seems to have come from Wilks (1976), who proposed adopting the ATN formalism to state preference semantic grammars. That is, the "PUSH" construct (which expresses a prediction and a procedure) would specify a "template" instead of a syntactic or semantic category. Wilks gave no examples where this would be appropriate, so it remains to be seen whether English has constructions which are amenable to this approach.

2. External

This is exactly the kind of information that generates predictions within sentence analysis. The sentence-initial prediction often expressed in ATN grammars as "PUSH NP" is created by the need for a subject in a clause, and the "PUSH VP" that follows is to provide a complement (or predicate) which can combine with the subject to form a clause. Similarly, the prediction generated by a "transitive" main verb is derived from the fact that an object is needed to combine with the verb to form a predicate. Nevertheless, knowing the role that a predicted item is to play (with respect to the surrounding structure) rarely constrains the form that it will take (except in terms of its overall semantic properties - see (D) below). We could introduce processing constructs like "FIND-SUBJECT" (or "PUSH SUBJECT", in ATN notation), but these would not specify processing procedures, since "SUBJECT" describes a way of using an item, not a kind of item or a way of processing one (with certain provisos, as in (D), below).

(D) Type of Semantic Structure

The predictions during sentence analysis (generated at level (C)2.) might well be expressible in terms of the semantic type involved. Certain abstract syntactic requirements (e.g. for the subject in a proposition) may need a particular kind of semantic structure (e.g. a "term"). For example, an English clause might be describable as a "term" followed by a "predicate", rather than as a "noun phrase" followed by a "verb phrase".

The nearest proposal to this that I know of came from Johnson-Laird (1975), who suggested that processing procedures could be organised so that there is a "REFERENT" routine which is invoked whenever a "referent" is expected in the input. As stated, this seems to be liable to the same criticism as in (B) above - English has no particular construction corresponding to "referent" (except perhaps the heterogeneous, if

syntactically precise, "NP"), and so this label could not specify any particular processing procedure. It is even dubious whether predictions of referents occur in English; some more general class, such as "terms" or "expressions" is needed.

(It should be noted that these predictions can, in any particular instance, be refined further using referential semantic classes (level (B) information). This is merely the old observation that, for example, certain verb-meanings often require certain semantic types of subject, object and indirect object (cf. Katz and Fodor (1963), McCawley (1968)). For example, (9) (i) requires a HUMAN object, and (9) (ii) a LIQUID (or (FLOW STUFF) ?) object.

(9) (i) He married...
(ii) John poured...

However, we cannot write particular referential classes into our grammar, unless we are following Woods et al. in making a grammar that is specific to one range of subject matter).

IV : Relationships between levels ?

The position sketched in Section III above is roughly as follows. Any high-level predictions concerning constituents within sentence analysis are generated by requirements at level (C) (abstract syntax), and may be expressible in terms of certain classes at level (D), semantic types (plus, in some cases, certain information from level (B), referential semantics). The question of processing procedures is less clear. Actual word strings do not always fall into neat classes, all capable of being processed by a single procedure. This difficulty has been covered up, to a large extent, by the adoption of traditional syntactic categories such as "NP", which have served both as a (rather haphazard) interface with semantic notions like "term" or "referent" and as a name for a list of separate types of surface strings that may express these semantic categories. The whole notion of "noun phrase" is rather hard to define, owing to its multiple roles in grammatical theory (cf. Ritchie (1977, Section III.11).

At this point in an argument, it is normal for a writer to wheel on to the stage his own shining new invention which renders all previous contraptions obsolete. Unfortunately, I cannot fulfil this dramatic requirement. Although I do have a model (Ritchie (1977)) which makes most of the necessary category distinctions indicated above, it does not throw light on how to organise a grammar around semantic categories, or how semantic predictions may influence processing procedures. I have come to the conclusion that this is, quite simply, because there is no such influence. Certain syntactic sequences may create specific syntactic predictions (for example, the occurrence of the word "the" constrains what may follow), and some higher level abstract syntactic constructions (and perhaps referential semantic constraints) may produce extremely general indications of what is to come (e.g. "a phrase or clause referring to animate beings"), but the variety of English grammar ensures that the latter cannot be translated reliably into the former.

This is not to say that levels (B), (D) and (E) cannot be used during sentence processing to aid the analysis procedures; for example, once a constituent has been analysed, its suitability for fitting into the wider linguistic context can be checked using the semantic levels of information, but it should be noted that a large amount of processing may have to be done on the constituent before it is in a suitable form for its higher level attributes to be assessed (particularly in the case of referential semantics - see Ritchie (1976)). However, there seems to be no way that the information generated at these higher levels can be used to select (in a top-down fashion) actual processing procedures at level (A).

Thus we arrive back at a very traditional approach - using higher semantic information merely to check the results of lower level syntactic processing. We also resurrect the old problem - how can we translate syntactically built structures systematically into semantic items? This is one crucial area where much work is needed.

It is worth mentioning in passing a related issue (which is not, however, central to the argument presented here). Various facts (e.g. the diversity of forms of "NP", commented on above) have led some people in artificial intelligence to discuss, over the past few years, "middle-out parsing". The idea is that using a combination of top-down and bottom-up techniques is the best approach (cf. e.g. Marcus (1975)). (This is not to be confused with the terminology of Woods et al. (1976), who use "middle-out" to refer to the direction in which the input string is scanned, contrasting with "left-to-right"). This seems very plausible, and it acknowledges that the notion of top-down processing is often inappropriate for actual words. What the arguments in this paper aim to establish is that the bottom-up processing of words would have to be organised around syntactic units, even if the top-down predictions were expressed semantically.

V : What Now ?

So far I have been developing very negative conclusions. This has been necessary, since I believe that it is not constructive to ignore certain basic questions of language research, or, worse still, to talk as if the questions have been solved. There seems to be a prevailing feeling within artificial intelligence that English grammars can be organised around semantic entities (usually, referential semantic ones), and that semantic predictions will greatly simplify the process of sentence analysis. I believe that the first of these claims is false, and that the second is over-stated. Any models which are based on these foundations may be interesting as reductio ad absurdum refutations of the premises, but they will not themselves be useful theories.

What we must do is tighten up our own methodological standards. Hypotheses must be based more on the actual patterns within language, rather than on current dogmas. We should be more explicit about what we mean by terms like "syntax" and "semantics", and state exactly what constructs we are defining within these headings. Above all, we must aim for theoretical generality, rather than concentrating on the quick profit of writing one-off programs with no wider consequences.

Acknowledgements

I would like to thank Alan Bundy, Chris Mellish, Martha Palmer, Arnold Smith and the conference referees for their comments on various earlier versions of this paper.

References

Bach, E. and Harms, R. (1968) "Universals in Linguistic Theory". Holt Rinehart and Winston, London.

Bobrow, D.G. and Fraser, J.B. (1969) "An Augmented State Transition Network Analysis Procedure". First International Joint Conference on Artificial Intelligence, Washington, D.C. Mitre Corp., Bedford, Mass.

Fillmore, C.J. (1968) "The Case for Case". In Bach and Harms (1968).

Johnson-Laird, P.N. (1975) Lecture at the School of Epistemics, University of Edinburgh, Edinburgh.

Kaplan, R.N. (1971) "Augmented Transition Networks as Psychological Models of Sentence Comprehension". Second International Joint Conference on Artificial Intelligence, London, British Computer Society, London.

Katz, J.J. and Fodor, J.A. (1963) "The Structure of a Semantic Theory". Language 39.

Lakoff, G. (1971) "On Generative Semantics". In "Semantics", ed. Steinberg, D.D. and Jakobovits, L.A., Cambridge University Press, Cambridge.

Marcus, M. (1975) "Diagnosis as a Notion of Grammar". In Schank and Nash-Webber (1975).

McCawley, J.D. (1968) "The Role of Semantics in a Grammar". In Bach and Harms (1968).

Montague, R. (1970) "Universal Grammar". Theoria 36.

Riesbeck, C. (1973) "Expectation as a Basic Mechanism of Language Comprehension". International Conference on Computational Linguistics, Pisa, September 1973.

" (1974) "Computational Understanding: Analysis of Sentences and Context". Ph.D. Thesis, Computer Science Department, Stanford University.

" (1975) "Computational Understanding". In Schank and Nash-Webber (1975).

Ritchie, G.D.(1976) "Problems in Local Semantic Processing". Proceedings of AISB Conference, July 1976, Edinburgh. Department of Artificial Intelligence, University of Edinburgh, Edinburgh.

" (1977) "Computer Modelling of English Grammar". Ph.D. Thesis CST-1-77, Department of Computer Science, University of Edinburgh, Edinburgh.

" (1978) "Augmented Transition Network Grammars and Semantic Processing". Internal Report CSR-20-78, Department of Computer Science, University of Edinburgh, Edinburgh.

Schank, R.C. (1972) "Conceptual Dependency; a Theory of Natural Language Understanding". Cognitive Psychology Vol.3 No.4.

Schank, R.C. and Nash-Webber, B.L. (1975) Proceedings of Workshop on Theoretical Issues in Natural Language Processing, MIT, June 1975. Association for Computational Linguistics, Arlington VA.

Wilks, Y. (1973) "The Stanford Machine Translation Project". In "Natural Language Processing", ed. R. Rustin , Algorithmics Press, New York.

" (1975) "Preference Semantics". In "Formal Semantics of Natural Language" ed. E.L. Keenan, Cambridge University Press, Cambridge.

" (1976) Lecture at AISB Conference, Edinburgh, July 1976. (In AISB Newsletter, November 1976).

Woods, W.A. et al. (1969) "Augmented Transition Networks for Natural Language Analysis". Report CS-1 to the NSF, Computation Laboratory, Harvard Univeristy.

" (1976) "Speech Understanding Systems, Final Report; Vol. IV: Syntax and Semantics". BBN Report No.3438, Bolt Beranek and Newman, Cambridge, Mass.

WHAT'S IN A SOCIAL ACTION FOR U?

M. Rosner

University of Essex

April 1978.

ABSTRACT

What is a social action, and what would it be for a social actions to be incorporated into a natural language understanding system? A simple notation for talking about the goals of the participants is introduced and it is suggested that given a sufficiently precise semantics, this notation could efficiently describe the knowledge needed by a system which could use, and understand the use by others, of a small subset of social actions.

KEYWORDS action, social action, conversation, politeness, dialogues, notation.

I. *INTRODUCTION*

The discussion below is to be viewed in the context of a project whose aim is to write a program called U which is capable of participating in the kind of Natural Language conversations which take place between th users and operators of a computing installation. Assume that U has knowledge of a small repertoire of actions which it is both capable of using, and understanding when used or requested by its conversational partner. The social actions, including, for example, thanking, apologising, complaining etc., depend for their use on what Bruce [1975] has called 'belief systems knowledge', that is, knowledge of 'institutional facts' [Searle 1969], knowledge of other's knowledge [Minsky 1968], and knowledge about intentions. The physical actions are those which have their effects on the underlying model of the computer system, such as allocating and deallocating resources, mounting tapes, and so on.

This division of actions into 2 classes may not ultimately prove tenable, but by making the distinction I can at least pose the question: What is involved in getting U to understand the social actions in its repertoire? For it is with actions like thanking, apologising, complaining, requesting, and the like that we shall mainly be concerned here.

Let us begin by considering some aspects of our own understanding of social actions.

II. *EXPLAINING ACTIONS*

The explanation of action A by agent X answers the question: why did X do A? There seems, however, to be a commonsense distinction between two types of explanation consistent with this definition, which I shall call (following Wilks [1977]) REASON and CAUSE explanations.

Suppose, for instance, that I ask for an explanation of X waving to me across the street. A paradigmatic reason-type explanation would be in terms of X's reasons: he wanted to greet me; he wanted to let me know that he was there; he wanted to show that he felt positively toward me. A paradigmatic cause-type explanation would be in terms of the movement of his muscles; because of the magnets attached to his arms and so on.

At an intuitive level, the difference between these kinds of explanation lies in the element of conscious choice that underlies reason-type explanations. With cause-type explanations, the notion of an underlying intention does not arise for the action is conceptualised as having taken place inevitably and irresistably, just as one billiard ball causes another to move through impact. Stating the difference in this way I make no claims to philosophical sophistication, for there is at least one sophisticated philosopher (Davidson [1963]) who maintains that reason-type explanations are merely disguised forms of cause-type explanations. However, I emphasise that it is not my purpose to give a philosophical account of the distinction; rather it is to exhibit that there are two kinds of explanation of action which may be acceptable as such under different circumstances. Consider how the explanation changes when what we thought was an intentional action turns out not to be, like the 'wave' of the wax man at Madame Tussaud's.

In understanding social actions, we are mostly interested in giving reason-type explanations, because we are dealing with actions that are performed on purpose, and not those which are the product of mere circumstances. It is only by treating them as such that a program like U might be able to respond with an appropriate social counteraction, and I therefore consider that the first step towards understanding a social action is the construction of a reason-type explanation.

III. *LEVELS OF DESCRIPTION*

Any system which purports to understand social actions takes as input descriptions of those actions. But the problem is that the same action can be described in many different ways. X waving at me across the street, for example, could be described as having performed a complex kind of arm movement, a wave, a greeting, and the suitability of each of these descriptions depends on the use to which they might be put.

The discussion above, and indeed, much of the discussion in this paper, presupposes a high level input description of the social actions involved, such as "X greet Y", "X thank Y", "X apologiseto Y" and so on. This is not intended to convey any lack of interest in the interactions between descriptions at different levels, and in section VI a plausible mechanism for building high level descriptions is briefly discussed. The briefness reflects my feeling that at present there is much to be said about the logic of social actions at this high level which advances our own understanding of the inferences that they cause to take place. It is crucial that such understanding should precede programming. The question of how a particular social action is 'implemented': for a greeting, say, whether it is performed by waving, or saying "Hi!", seems independent (it seems of me) to the question of formalizing what takes place in the belief system of a potential greeter or greetee.

IV. *GOALS*

One way of conceptualising intentional behaviour is in terms of goals.

In this paper, as throughout most of the AI literature, the concept of goal is closely connected with the notion of a state of the world which consists essentially of an infinite conjunction of propositions. For example, the command "open the door!" is associated with the goal proposition: the door is open, so that any state of the world which contains it is a candidate goal state which the commander wants to be brought about. Because infinite conjunctions of propositions cannot be represented extensionally, an alternative and more tractable way to deal with states treats each contained proposition as a predicate which is true when applied to the state in question. The proposition "open(door)" then becomes the predicate

lambda s.open(door,s).

These are essentially the propositional fluents of McCarthy [1963].

This conceptualisation of goal is adequate to explain the simplest kind of intentional behaviour. The issue of the above command, for instance, can be explained along the following lines: issuing a command is a legitimate method for conveying to a hearer that one has a goal. Therefore by assuming the commander has a goal, his utterance is explained as a step in a known plan for getting the hearer to know that one has a goal. Such goal oriented explanations are not new in NL understanding systems. SHRDLU [Winograd 1972], for example, inspected its goal tree to answer 'why' questions about its own actions. The intrinsic connection between explanations, goals and plans has also been exploited for the generation of social actions by Perrault and Cohen [1977]. The algorithm for satisfying goals conceived in this way goes

something like this: find the action which yields a state to which the goal predicate applies, then infer that the owner of the goal desires the performance of that action.

A peripheral issue which deserves mention is that NL allows the expression of many intentions which cannot be captured by this simple relationship between 'naive' goals and actions. For example, compared to the unproblematic a.

a. Open the door!

the description of what the speaker wants in b, c and d

b. Keep the door open!
c. Leave the door open!
d. Let the door open!
(these examples due to Pat Hayes)

seem to require more than a simple state of the world. In b and c it is something like a period of time with certain characteristics, whilst with a and d it is a certain kind of event. Orthogonal to this 'temporal scope' of the goal is the 'potency' of the world presupposed: in the absence of further action, the door changes state spontaneously in b and d, whilst in a and c it stays put. Each of these axes is related to the problem of deciding whether the commander desires explicit intervention, as opposed to the mere cooperation of a hearer who is attempting to satisfy the goal.

Unfortunately, space prevents the full consideration that these issues deserve, but they will be given due attention in a forthcoming paper. For the remainder of this discussion I will assume that a goal can always be achieved by the performance of some action (conceived as a state to state transformation).

V. *GOALS AND SEQUENCES OF SOCIAL ACTION*

One useful property of 'goal talk' is that it is easy to define a concept of 'goal ownership', that is, that someone has a goal, without committing oneself to an exact specification of its nature. The utility of this may not be immediately obvious, suffice it to say that one can give a surprisingly simple account of such social actions as thanking, apologising, and requesting using this concept and the related concepts of 'goal achieving' and 'goal disachieving', which I shall now attempt to illustrate. The key idea is that in social interaction we are not limited to pursuing our own goals, but also the goals of others.

Consider the following instance of thanking:

A> Have you got a light?
B> Here.
A> Thanks.

What I want to suggest is that whenever X owns a goal and Y does something which achieves that goal, then X thanks Y. In this case, A's first utterance can be construed as communicating to B that he has a goal. Specifically, that he wants A to give him a light. B then gives him the light, achieving his goal, with the predicted result that

A thanks B. Notice that it is possible to state this scenario about thanking without mentioning what the goals are. This can be made explicit with a simple piece of notation: Let <X:gY+> indicate an action by X which achieves some goal which is owned by Y. Then a consistent sequence of actions is

(1) <B:gA+> <A thanks B>

Now consider, with the aid of this notation, another possible sequence:

(2) <A ? B> <B:gA+>

What possible social action by A to B is consistent with the situation before B achieves one of A's goals? The answer, I suggest, is something close to a REQUEST.

The notation can be extended by allowing <X:gY-> to stand for an action which by X which disachieves one of Y's goals. This notion of disachieving is hard to define exactly, but for the time being it will be sufficient to say that A's goal is disachieved just after B's 'no' in

A> Can I have a light?
B> No.

Here I suggest that what makes B's utterance into an abrupt refusal rather than a polite one is the lack of apology, or to be exact, pre-apology [Goffman 1971]. (cf B> Sorry, no.)

This transaction would be represented in the notation by

(3) <A preapologiseto B> <A:gB->

Note the distinction here between preapology and apology, which would be represented by

(4) <A:gB-> <A apologiseto B>

In preapology, the undesirable action (in this case utterance of the word 'no') is in the future with respect to the preapology. In apology proper, it is in the past (this is reflected in our understanding of another possible reply by B, B> No, sorry.)

Now clearly, by enumerating all the combinations of variables in the notation as it stands, we can guess at plausible interpretations for the social actions involved, for example by switching the variables in the second component of (4)

(5) <A:gB-> <B ? A>

we get B's action to A after A has disachieved one of B's goals. A social action such as COMPLAINING would fit this scenario. Other combinations suggest other social actions eg

(6) <A warn B > <B:gB->
(7) <A advise B> <B:gB+>
(8) <B:gB-> <A sympathisewith B >
(9) <B:gB+> <A congratulate B>

I don't want to argue that there is anything magical about this particular notation, but what is interesting is that the achieving and disachieving of just one goal seems to be a powerful constraint on the kinds of social actions that surround it. The fact that this notation happens to represent this economically is no more than a happy, if

somewhat contrived, coincidence.

An obvious objection to this claim is that the meaning of thanking, and related social actions, could not possibly be elucidated with a theory of this simplicity, for there is no provision within it to mention the relationship of one person's goals to the other; to state, say , that B wants his goal to be achieved but does not want A to achieve it. To some extent this problem is an artifact of the notation, for I have said nothing about the process by which the achieving or disachieving of a single goal is determined. This could be arbitrarily complex, and certainly could involve a proposition of the form: B's goal is such that all actions by A will not achieve it.

Another point which needs clarification is the status of expressions like

(10) <B:gA-> <A thanks B>

(11) <A preapologise to B> <A:gB+>

which are semantically odd given our normal assumptions about the participants, and the concepts of thanking and preapologising.

The trouble is that, up till now, I've been using the notation to indicate consistent sequences of action, given the normal assumptions. Thus by merely writing (10) or (11), I am implying that they, too, are consistent which they clearly are not. The way out of this paradox is to make these normal assumptions explicit, so that (10) and (11), thus fully stated, would come out as inconsistent, causing the rejection of one such assumption. Which one to choose is, of course, a problem, but in this case any system should be more prepared to reject an assumption about one of the participants than about its knowledge of thanking or preapologising.

For the time being let us ignore such shortcomings and ask ourselves what possible use the knowledge which we have been making explicit could be to a NL understanding system. To do this we must first of all make certain assumptions about such a system.

A1. That it should be able to use, and recognise the use by others, of social actions such as thanking, apologising, etc.

A2. That it should have independent criteria for deciding whether a goal has been achieved or disachieved.

Now the question is, what could an expression like (1)

(1) <3:gA+> <A thanks B>

MEAN for such a program? I will try to answer this by trying to state the general character of inferences which could be made from it. To begin with, I will make the point of view of the program explicit by using 'self' and 'other' instead of 'A' and 'B'. This generates 2 configurations for (1):

(12) <Self:gOther+> <Other thanks Self>

(13) <Other:gSelf+> <Self thanks Other>

Essentially, each expression states that a given sequence of actions is 'consistent' in a sense which could be used

(a) To provide the system with EXPECTATIONS, eg if Self has just achieved Other's goal, it should expect to be thanked, (from 12)
(b) To provide SUGGESTIONS for 'what to do next', eg if Other has just achieved Self's goal, then a suggestion for what to do now is for Self to thank Other, (from 13)
(c) To provide EXPLANATIONS of actions by Other, eg if Other thanks Self, then this could be explained by the existence in memory of a past action by Self which achieved the goals of Other, (from 12).

I could write down similar analyses for the other expressions listed without any difficulty. A much more substantial problem is giving a complete account of the meaning of such expressions, and this would involve stating exactly how the system handled the concepts of suggestion, expectation, and explanation. Saying, for example, what happens when an expectation is thwarted, when an assumed explanation fails to tally with the facts, or when a suggestion should not be complied with. These are complex matters which relate in a non trivial way to the ontological committments in the system's representation of such 'entities' as a state of the world, other persons, and the relationships that can exist between them (see [Schmidt & d'Addamio 1973] for an interesting example of a complete system which employs a sophistocated model of persons to explain their actions). The plausibility of (10), say, might be restored given the knowledge that A is a masochist, and the suggestion to thank Other might best be ignored if it were known that she was rude in the first place.

To give a complete account of the meaning of an expression is to state the kind of thing it refers to. In this case, to say that each expression denotes well formed sequence of actions of some particular kind. At this stage, I am unable to do this, but common sense, as well as a desire to avoid committing the sin of Tablets of Stone [Hayes 1975] dictates that we ignore this duty at our peril. I have tried to give a preliminary account in the absence of further details of the system by saying how these expressions are used, and for the present this will have to suffice.

VI. *LINKING THE NOTATION WITH THE TEXT*

For a NL system to use the consistencies which I have been discussing, it would need ultimately to have some way of relating social actions to the input utterances.

In some cases, this is trivial, since the occurrence of a single word can itself be sufficient evidence that a particular social action has taken place. The explicit performatives [Austin 1962] are a case in point. But generally speaking, the process will not be a direct one and may involve a considerable amount of inference. The

utterance

'Can ?x ?y'

is three ways ambiguous between a request for action, a request for information, and a request for permission depending on the kind of pronoun ?x is, and the kind of action ?y is (a fuller discussion of this example appears in [Hayes & Rosner 1976]). Clearly any mechanism which has to decide between these alternatives must be able to reason freely with both pragmatic and syntactic categories.

A possible handle into this kind of inferencing is a phrasal lexicon of the type mentioned by Becker [Becker 1975], and at the moment I am experimenting with a pilot implementation of this which matches lexical items consisting of complex patterns with restricted variables against input utterances. The output is a list of plausible hypotheses about (amongst other things) the syntactic and semantic categories of the variables, and the social actions being performed by the participants. These are expressed in predicate calculus. Of course, there is no reason in principle why the same mechanism should not be used 'in reverse' for response generation though at present no work has been done in that direction.

VII. *PROBLEMS WITH ACTION DESCRIPTIONS*

Note that expressions in the notation are of two types

Type 1: <X thank Y> (say)

Type 2: <X:gY+>

and they each denote actions by their respective agents. But they differ in the following way. Type 1 expressions are naming particular actions such as thank, forgive, etc, whereas type 2 expressions characterise the action by the effect it has on one of the participant's goals. So what we have, in effect, are 2 partial descriptions of actions, with each focussing on a different aspect. Or do we?

As far as a system using this notation is concerned, this raises an important question about the way in which which actions are to be represented, and that is, should type 1 and type 2 expressions be understood as different partial descriptions of actions of the same kind, or should there be an explicit ontological distinction between the kinds of action referred to by type 1 and type 2 expressions?

To adopt the latter position is to say that reasoning about (say) thanking is essentially separable from reasoning about the action in virtue of which the thanking is made, that the system's descriptions of the concept 'thank' are at the level of type 2 expressions, and no deeper. This view is almost certainly a gross oversimplification, but I believe that such descriptions may be adequate for a certain class of social actions, Thanking, apologising etc. seem to illustrate this quite well. If they are, then I am justified in using a notation

like this and making assumptions like A2 for that class. Much of the reasoning in this paper has been conducted from this point of view.

The problem is with social actions that cannot be handled in this simple way. Consider 'greeting', for example. Here I cannot conceive of any expression of the form

<X greet Y> <Y: gX???>

which successfully reflects our intuitions unless ??? expresses something in addition to mere achieving or disachieving of X's goal. Essentially, what is required is information about its contents, and this takes us way beyond any possible semantics for the simple notation I've been considering.

The former position solves the problem by refusing to admit any difference in type between <Y:gX+> and <Y:g???>, treating each as a different kind of partial description. Unfortunately this presupposes a general set of axioms which is adequate to cope with ALL the different partial descriptions of action which might arise, and I am certainly not in position to do this.

Each position has its advantages, although I believe that at this stage, it is more fruitful to impose distinctions on different types of action if by so doing we can produce useable descriptions of even one such type.

VIII. *CONCLUSION*

I began by asking what U needs to understand social actions, and have proposed that for a small subset of these, a simple formulation based on the concepts of goal ownership, achieving, and disachieving will suffice. I have employed a notation to illustrate the kind of consistencies which U will be working with. I have also tried to stress the limited use to which such a notation can be put without any clearly stated semantic theory to guide and constrain the significance to be attached to it. It is of the utmost importance, therefore, to determine U's ontological commitments in order to find out just how useful the ideas presented in this paper will turn out to be.

----------:----------

IX. *Acknowledgements*

I would like to thank Tony Cohn, Mike Brady, and especially Pat Hayes for their helpful comments, criticisms and suggestions.

X. *REFERENCES*

Austin J.L. How to do things with words, Oxford 1962.

Becker J.D. The Phrasal Lexicon.
Proc TINLAP Conference, Cambridge, Massachussetts. (1975)

Bruce B. Belief Systems and Language Understanding.
BBN Report No. 2973, January 1975.

Davidson D. Actions, Reasons and Causes.
Journal of Philosophy vol 60 685-718 (1963)

Goffman E. Relations in Public, Penguin 1971.

Hayes Pat Nine Deadly Sins.
AISB Newsletter, July 1975.

Hayes P. & Rosner M. ULLY, a program for handling conversations.
Proc AISB Conference, July 1976.

McCarthy J. Situations, Actions, and Causal Laws
Stanford AI memo no. 2, July 1963.

Minsky M. Matter, Mind, and Models.
Semantic Information Processing,
MIT Press, 1968.

Perrault C. & Cohen P. Planning Speech Acts.
AI memo 77-1 (1977)
Dept Computer Science, University of Toronto

Searle J. Speech Acts,
Cambridge University Press, 1969.

Schmidt C. & D'Addamio J. A Model of the Common Sense theory of Intention and Personal Causation.
Proceedings of the 3rd IJCAI

Wilks Y. What Sort of Taxonomy of Causation do we need for Language Understanding?
Cognitive Science 1, 235-264 (1977)

Winograd T. Understanding Natural Language,

A CONVERSATIONAL SYSTEM WHICH UNDERSTANDS SHORT STORIES

G. SABAH, Groupe de Recherche 22 du CNRS
4, Place Jussieu 75230 Paris France

ABSTRACT

We present a conversational system which can learn short stories in a natural language and perform some reasoning about them.
In our model, a sentence is represented by a purely semantic case structure ("deep structure") with embedding properties.
The vocabulary is a tree of word representative elements, each of which having its sense precised by some semantic features (pointers to other words). To understand a sentence within a given story, we reduce the information it carries. This reduction gives way to the application of a few rules for character behaviour, whose aim is to connect the sentence to another one by establishing some causal relationship between desires and actions. The system can then answer "why" questions and questions about some facts not explicitly described in the story.
The natural language used is French, the program is written in SIMULA

I - INTRODUCTION

The system here described is designed to converse with people in a natural language (French). It carries out a kind of understanding which relies upon knowledge of level intermediate between purely semantic and pragmatic. Short "children" stories are told to the system. Then we ask questions about these stories in order to test what has been "understood" by the system.
Our three basic ideas are:
1/- The "deep structure" of a sentence is a relational structure we build on the model of case-grammars (FILLMORE [5] CHAFE [3] , CELCE-MURCIA [2]). In this model, a sentence is viewed as a set of noun phrases, each of them being related to the verb by a particular connexion which expresses the semantic relation it has to this verb.
In order to deal with some syntactic facts, like active/passive forms, this structure is merged with another one, representing each sentence as a predicate applied to a topic (one privilegied NP).
2/- In order to memorize the set of informations we call vocabulary, we drew our inspiration from the psychologists recent thoughts about the organisation of human memory: specially the notion of semantic feature (OSGOOD [7] "atoms of sense"). A word is connected to a finite set of such features, each one of them defining a particular aspect, or "use", of its sense. The representation we chosed for these features is not too memory-consuming (presently 300 concepts in a 5K storage) and it is compatible with a learning of new concepts or new relations of the vocabulary.
3/- Rules for character behaviour are used to establish a causal order among desires and actions met in the story. Our aim was to achieve real comprehension of each sentence of a given story, which means understanding of its relations with the context. We consider that correct understanding is reached when the program has inferred the causal relationship for the sentence. Our "rules for behaviour" are derived from common sense, introspection and elementary freudian results.

In order to get sentences easily handled by these rules, we have to extract information about desires, actions, perceptions and relations of each character. We thus constructed the notion of "Abrégé". The parameters of an abrégé are his type (one of the fundamental class of facts listed above), and a list of characters who play a role in the sentence [or set of sentences] concerned. A story is then represented as a structure of abrégés bound by causal relations which are calculated by means of applying the behaviour rules.
So the flow of treatment of a sentence will be:
. traduction from French into case structure
. generation from this structure of abrégés
. search for possible causes of these abrégés
. search for such causes among the already known facts.
Details on the complete system are to be found in SABAH [8] . We shall only insist now on the construction of causal relationship within a story.

II - THE STRUCTURE OF A STORY

II.1.- Characters

When we are told a story, the name and level of definition of a concept are not enough to determine who or what a noun phrase means: we wish to make a distinction between two different objects identically named, or to recognize a single one under various appellations. So we need a new identifier for the concept, or proper name of an object being a particular occurence of a more general class. A *character* is then defined by his name, i.e. the class it belongs to, and his particular characteristics, i.e. informations allowing us to distinguish it from others representants of the same class.

II.2.- The Abrégés

Having identified characters, our structure now has to express connections between the sentences concerning them. To do so the program operates a certain reduction of the information each sentence contains: it creates one or more "Abrégés" which are meant to describe the fundamental relations between characters. This agrees with GREIMAS [6] who exhibits, when building models for sense, the two phases he calls reduction and structuration.
We have chosen five kinds of "Abrégés":
Desire, Intention, Action, Perception, Relation.
The order in which the characters appear in an abrégé depends on the role they play in the situation described.

Let us give some examples of translation into "abrégés":

"The fox is hungry" → *(Desire; fox, *edible)*
(the star indicates that edible is a semantic feature,not a character)
"the fox wants the fishes" → *(desire; fox, fishes)*
"the fox wants the raven to let go his cheese" →
(Intention; fox,raven,cheese)
(the first character(fox)wants to modify a relation between the second and the third ones (raven and cheese))

A sentence can also generate more than one "abrégé":

"The fox sees a raven who is holding a cheese" →
(Perception; fox,raven) and (relation; raven,cheese)

To create an "abrégé" from the deep structure of the sentence, we examine the semantic features of the verb, which gives us the kind of "abrégé", and then, according to this result and to the semantic relations expressed(or not)in the deep structure, we find out which characters are involved.

II.3.-The character behaviour rules

Seven behaviour rules are a priori given to the program;each of them consists in two members, the left one being a possible cause of the right one.Their elaboration involved a mainly intuitive process now under study.These rules are:
(naming x,y,z...character variables,Q a semantic feature variable)

1. *(Desire; x, Q)* *(Perception; x, y, Q)* → *(Desire; x, y)*
2. *(Desire;x, y)* *(Relation;x ,y)* → *(Action; x,y)*
3. *(Desire; x, y)* → *(Intention; x, x, y)*
4. *(Desire; x, y)* *(Relation; z, y)* → *(Intention; x, z, y)*
5. *(Intention; x, y, z)* → *(Action; x, y)*
6. *(Intention; x, y, z)* → *(Action; x, z)*
7. *(Action; x, y)* → *(Intention; y, y, x)*

The first and the fourth, for instance, may be paraphrased as below:
1:if a character x wants something with the quality Q and if he perceives y with this quality , x may desire y.
2:if a character x wants an object y and if a character z is related to this object, x may intend to act upon the relation between z and y.

II.4.-Building causal relationships

These rules being given, we want the system to find an explanation for the new "abrégés" that appear as the story runs.By explanation of an abrégé A_0, we mean an abrégé A_1 already met in the story, and a construction such that

A_1 A_2 ... A_0 R_n being a behaviour rule

For instance, from the sentence:
(1) "A hungry fox sees a raven who is holding a cheese"
the program derives: *(Desire; fox, *edible)* (a)
(Perception; fox, raven) (b)
(Relation; raven, cheese) (c)

Then if we say: *"The fox speaks to the raven"* , the "abrégé":
(Action; fox, raven) (d) has to get explained, being an action.

Rule 5 shows that it may be caused by: *(Intention; fox, raven, x)* (e)

which in his turn, following rule 4 , may issue from:

(Desire; fox, x) *(f)* and *(Relation; raven, x)* *(g)*

And now, we are lead to replace x by cheese so that (g) identifies with (c) and (f) comes from (a) by rule 1,as cheese has the feature edible.
We consider that the second sentence is explained by the conjunction of sentence (1) ,rule 1,rule 4 and rule 5.

That construction might resemble to a frame (cf ABELSON(1),CHARNIAK(1) SCHANK(9)) "speaking to somebody who is holding something we wish to have" but, here, it xas specially constructed for the given situation.

Recapitulation of the different steps of the understanding process:

-Identify the "abrégé" to be explained with the right member of a rule i.e.consider the characters of the abrégé as values for the variables.
-Generate the corresponding left member (inference of an abrégé).
-Search among the abrégés already understood:
.if the inferred one is found, the process terminates and the sequence of causes is stored.
.if not we repeat the process with the different rules, and if it still fails, we try to understand each of the inferred abrégés by the same operations, so building a tree of possible causes, which by construction is finite.

Processing of a story is given below in form of question-answering by the program.

Program:I'm listening
Man:listen to a story
P:name of the story?
M:the cat and the mice
P:I'm listening
M:a hungry cat comes into a house .
the mice of the house then go and hide .
the cat goes into the attic .
he goes down into the cellar .
he does not see any mice .
he gets hidden .
the mice who do not see the cat any longer , go out of their hole .
then the cat catches them .

M:where does the cat go ?
P:into a house
into the attic
into the cellar
M:what does the cat want ?
P:the cat wants to eat the mice
M:why does the cat go into the attic .
P:because he wants to create a relation between him and the mice
M:why does the cat get hidden ?
P:because he wants the mice to go out of their hole
M:why do the mice go and hide ?
P:because they don't want the cat to eat them

III -CONCLUSION

The aim of our work is to develop an automatic process of comprehension and precisely totest what could be "understanding" for a machine . Our program is to said to understand a fact when it can exhibit a cause for this fact or a reason why it happens whithin a given story . This is already sufficient to get non-trivial answers as shown by the results.

Some improvements can be made in the program . The most important one should be to provide the system with as many "comprehension structures" as there are conscious characters in the story . This would allow us to deal whith inequal knowledge situations , such as lies or misunderstanding.

For future research , we suggest comparison between presently used "methods of understanding" , in view of developing parallel systems which could obtain a lifelike comprehension .

BIBLIOGRAPHY

[1] R.P. ABELSON, 1975 .
Representing mundane reality in plans .
In representation and understanding. BOBROW & COLLINS (Eds.)
New York:Academic press

[2] M. CELCE MURCIA, 1972 .
English comparatives .
Ph D. Thesis. Los Angeles:UCLA Dept. of Linguistics.

[3] W.L. CHAFE, 1970 .
Meaning and the structure of language .
Chicago:University of Chicago press .

[4] E. CHARNIAK, 1976 .
Inference and knowledge . In Computational semantics. E. CHARNIAK &Y. WILKS (Eds.).North holland publishing company, Amsterdam .

[5] C.J. FILLMORE, 1968 .
The case for case . In universals in linguistic theory .
Bach &Harms (Eds) Chicago:Holt, Rinehart & Winston, Inc.

[6] A.J. GREIMAS, 1966 .
Semantique structurale . Langue et langage, Ed. Larousse, Paris .

[7] C.E. OSGOOD, 1963 .
On understanding and creating sentences .
In American psychologist. 18, 735-751.

[8] G. SABAH, 1978 .
Contribution à la comprehension effective d'un récit .
Thèse de doctorat d'état, Janvier 1978, Paris VI .

[9] R.C. SCHANK, 1975 .
The structure of episode in memory .
In Representation and understanding , Bobrow & Collins (Eds)
New York: Academic press .

WHAT STRUCTURE DOES A LISP PROGRAM HAVE - BESIDES BEING A SET OF PROCEDURES?

Erik Sandewall
Informatics Laboratory
Linköping University
Linköping,Sweden

Abstract: The hierarchical structure of algol-like languages is well known. In procedure-oriented languages such as APL or LISP, a program consists of a large number of procedures, each of which is usually small in size. For large programs, consisting of hundreds of procedures, it is necessary to have some higher-level structure to organize the procedures. The present paper describes such a higher-level structure. A programmer's-assistant-type program which supports the model has been implemented, and the experience in using this program gives some support for the model.

Caveat: Because of the space limitations at this conference, the following is a condensed version of the paper. The full paper is available as a departmental report with the same title. Introductory material (necessary for readers who are not active users of incremental programming systems such as LISP systems) as well as a description of the working system and a comparison with current program models, has been left out in the present version.

Background and purpose of this work

Work on program structure is usually prescriptive: it specifies structures that its author feels should be used for organizing programs. The present work is instead descriptive: it attempts to identify structures that actually exist in programs that people write. It is therefore hoped that the paper will give the reader a feeling of déjà-vu: "yes, I write my programs that way". Surprise is not an intended purpose of the present paper.

The work of the paper is based on the LISP language and programming system, and would only be (practically) possible in an incremental system such as a LISP system. LISP systems store programs in their data base, and support an easily extensible data base. This encourages the user to augment the data base with additional information that make the structure of the program explicit, and to let this structure grow and improve gradually. In conventional programming languages, program structure is frozen into the language.

The basic program structure of LISP systems (built into the basic system) is that a program is a set of procedures, together with initial values for the data base. On a syntactic level, this implies a simple procedure-call structure on a set of procedures, with arrows from caller to callees.

Current LISP systems such as INTERLISP (1) contain tools for program maintenance which support the procedure-call structure as well as a grouping structure where a set of related procedures are collected into a file. These tools are helpful but still insufficient for describing and managing large programs.

In some previous work (2,3) I have suggested the use of data-driven procedures, organized by a data base management system which operates on a data base with embedded program fragments, as a useful structuring principle. This has sometimes been useful but (like all prescribed models) sometimes it does not help much for structuring the application.

Thus we formulated the question: in an average LISP program with a lot of procedures but no declarative framework, which structures can be identified on an a posteriori basis. The question was addressed by studying existing programs in our group, and by developing a system which supported the program model and which itself was adequately described by the model.

The model

A program is a set of procedures. These procedures are first grouped on three levels: operative procedures, auxiliary procedures, and elementary procedures.

Operative procedures are intended to be called by the user during his interaction with the program. They can be viewed as commands to the program, and are most naturally understood by the programmer in problem oriented terms, i.e. the operation they perform. Procedures for loading data from external files, for editing, and for printout of specified sections of the data base are typical examples of such procedures.

Elementary procedures are procedures which service specific data structures, or constructs in the data base. They are typically the result of a bottom-up design where a data structure has been defined first, and procedures to service it arise as corollaries. They correspond in some ways to the procedures associated with data types in CLU (4). Typically the programmer understands them in a two-dimensional way as a certain operation on a certain data type; more dimensions will be suggested below.

Auxiliary procedures are intermediate between the other two groups: they are understood by the programmer by what the do, rather than what data structure they support, but the operation is internal to the system and cannot be understood in problem-oriented terms. Auxiliary procedures have been less common than the other two kinds in the examples I have looked at.

This dimension in the structuring of the program will be referred to as the three levels of the program. Inside each level, procedures are grouped together into families of related procedures. The members of the family have different roles, which keep the family together. The roles have been arrived at empirically and are as follows:

- the head or main procedure is the one which does the major job, and which is located at the top or close to the top in the procedure-call hierarchy of the family. The name of the head procedure becomes the family name.

- capping procedures: often it is not convenient for the user to call the head procedure of an operative procedure family directly e.g. from the terminal, and there may then be a small procedure which is called by the user and which calls the head. The capping procedure may e.g. set up the right environment for the head, or allow a more convenient and flexible argument structure.

  For example, in the INTERLISP system itself, editl is the head of the editor, and edite is a capping procedure.

- servant procedures are the ones called by the head as its immediate auxiliaries. Unlike the level of auxiliary proceures which was described above, servant procedures are not understood by the programmer in terms of a separate purpose; they must always be understood together with their head. Otherwise speaking the procedure call from the head to the servant is a control structure very similar to an iterative statement inside the head,(and often serves the same purpose). Local variables in the head are very often used by the servants as well. There may of course be several levels of servants.

- handle procedures are called by a family member in a data-driven way.

- modified procedures have been formed from another family member by a small variation, for example to perform some extra side-effect or take some special case into account.

In this fashion, we distinguish families of procedures on all three levels. Most families have a few members, many have just one, and some have had more than ten in the examples I have looked at.

On the operational and elementary levels, which is where most procedures go, families are grouped in the following ways:

On the operational level, the model uses a network of data states which are related by operations. This could be called a data-flow model or information precedence model of the top level of the program.

Each of the operations may be implemented by one or more procedure families on the operational level. On the elementary level, we obtain structure by using instead hierarchies of data types and access methods, each of which is associated with one or more procedure families that support it. In a second, orthogonal dimension, each family is associated with an operation chosen among the following:

For access methods (addional explanations in the full paper):

operation	example among standard INTERLISP functions
store	putprop, putd
retrieve	getprop, getd
delete	remprop
modify	addprop, advise
print loadable	printfns

etc.

For manipulation of data types, i.e. composite expressions (lists, association-lists, forms, the structures which appear as values to INTERLISP's fileCOMS, etc.) a four-dimensional classification is possible based on

a) the data type at hand
b) the operation on the instance of that data type
c) whether the operation is addressed by content or by position
c) whether the operation uses side-effects or not

The following table gives the major operations plus examples of standard LISP functions for the four possible combinations of the last two dimensions:

operation	reference by position		reference by content	
	without side-effects	with side-effects	without side-effects	with side-effects
combine	cons append	nconc	union	
extract/ delete	car cdr	gnc	memb assoc remove	dremove
change		rplaca	subst	dsubst
rearrange	reverse flatten	dreverse	sort	

plus the following operations for which the four columns do not apply:

operation	example
measure	length
classify	atom typeof
convert	fix unpack
read	read ratom
print	print prettyprint

Notice that the LISP functions in the tables are given as examples only. A classification of existing LISP functions is in itself not such a useful exercise; our point is that the categories in the tables have been useful for structuring collections of procedures in application programs.

In summary, the model organizes a set of procedures as follows:

1) distinguish the three levels of operational, auxiliary, and elementary procedures
2) distinguish several operations on the top level and a range of access methods and data types on the lowest level. (All programmer defined)
3) distinguish several operations on each data type and access method, together with the four combinations of binary choices for some of the operations
4) distinguish families of procedures, associated with top-level operations, with data-type/operation combinations on the lowest level, or just located on the middle level
5) each procedure family may consist of several procedures which play various roles. (The range of roles is semi-fixed).

Discussion

The model proposed here has some features in common with program structures used in conventional languages, but there are also significant differences. This is discussed in more detail in the full paper.

The evidence for the model, besides common-sense reasons and a synthesis of previous experience, is that it was able to accomodate one non-trivial program (≈120 functions) in a natural way. Program listings organized according to the model by an implemented support system, were estimated to be significantly superior to conventional listings.

The major limitations of the model are that global data base assignments (including global variables) and contributions to handles provided by the surrounding system, are not modelled.

References

1. W. Teitelman et al.
INTERLISP Reference Manual
Xerox PARC, Palo Alto, Calif., 1975

2. E. Sandewall
Some observations on conceptual programming in Machine Intelligence 8, E.W. Elcock and Q Michie (eds) John Wiley & Sons, 1977

3. E Sandewall
Ideas about management of LISP data bases
Proc. Fourth International Joint Conference on Artificial Intelligence, 1975

4. B. Liskov and S. Zilles
Programming with abstract data types
SIGPLAN Notices Vol. 9, Nr 4, 1974

REPRESENTING ACTIONS BY STATE LOGIC

Camilla B. Schwind
Technische Universität München
Institut für Informatik
8 München 2, Postfach 202420, FRG

Abstract

The following paper presents a formal calculus interpreted by Kripke-type semantics for the description of the actions of an intelligent system. The frame problem is reduced to the logical problem of proving consistency.

The Logical Theory Z

The language of Z, L(Z), is an extension of classical predicate logic by the state operators +, - (all immediately following and preceding states) and $\bar{F}$, $\bar{P}$ (all following and preceding states) and the modal operator $\Box$. Terms and formulae are defined as usual; and if A is a formula then +A, -A, $\bar{F}A$, $\bar{P}A$, and $\Box A$ is. A Kripke-Structure K for Z is given by a set of classical structures $\{K_s : s \in S\}$, where $S \neq \emptyset$ is the set of the states of K, together with a binary relation R_o on S. The transitive and reflexive closure of R_o is denoted by R. A truth function T^K assigns to every formula A and every state $s \in S$ a truth value t or f dependently on K_s. The state and modal operators are interpreted by means of the Kripke relations R_o and R as follows:

$T^K(+A,s)=t$ iff $T^K(A,s')=t$ for every s' such that $s\ R_o s'$
$T^K(-A,s)=t$ iff $T^K(A,s')=t$ for every s' such that $s'\ R_o s$
$T^K(\bar{F}A,s)=t$ iff $T^K(A,s')=t$ for every s' such that $s\ R\ s'$
$T^K(\bar{P}A,s)=t$ iff $T^K(A,s')=t$ for every s' such that $s'\ R\ s$

We define the additional operators:

$\Diamond A \longleftrightarrow \neg \Box \neg A$
$\dashv A \longleftrightarrow \neg + \neg A$

We have proven the completeness of the following logical axioms and inference rules:

A1 All valid formulae of classical predicate logic
A2 a. $\bar{F}A \longrightarrow A$ b. $\bar{P}A \longrightarrow A$
A3 a. $\bar{F}A \longrightarrow \bar{F}\bar{F}A$ b. $\bar{P}A \longrightarrow \bar{P}\bar{P}A$

A4 a. $\bar{F}A \rightarrow +A$ b. $\bar{P}A \rightarrow -A$

A5 a. $\neg+\neg-A \rightarrow A$ b. $\neg-\neg+A \rightarrow A$

A6 a. $+A \rightarrow \neg+\neg A$ b. $-A \rightarrow \neg-\neg A$

A7 a. $+(A \rightarrow B) \rightarrow +A \rightarrow +B$ b. $-(A \rightarrow B) \rightarrow -A \rightarrow -B$

A8 a. $\Box A \rightarrow \bar{F}A$ b. $\Box A \rightarrow \bar{P}A$

A9 a. $+\Box A \rightarrow \Box A$ b. $-\Box A \rightarrow \Box A$

A10 $\Box(A \rightarrow B) \rightarrow \Box A \rightarrow \Box B$

A11 $\Box A \vee \Box \neg \Box A$

R1. Cut rule: from $A \rightarrow B$ and A infer B

R2. Substitution rule: from $A \rightarrow B$ infer $\exists xA \rightarrow B$ provided x is not free in B

R3. from A infer $\Box A$

R4a. from $B \rightarrow A$, $B \rightarrow +A$, $B \rightarrow ++A$, ... infer $B \rightarrow \bar{F}A$

R4b. from $B \rightarrow A$, $B \rightarrow -A$, $B \rightarrow --A$, ... infer $B \rightarrow \bar{P}A$

Characterization of worlds and actions by Z

A world can be described by nonlogical symbols of Z together with physical (or empirical) laws expressed by formulae of Z. Actions are represented by action predicates. State changes are caused when actions are executed and they correspond to the relation R_o. In order to describe world changes we must formulate action axioms which describe

1. What are the preconditions of an action to be carried out.
2. What are the consequences of an action if it has been carried out.

For example, consider the blocks world in figure 1 whose physical laws are $HOLDxy \rightarrow \neg\exists z ONyz$, $\neg ONxx$. Let w be the conjunction of these laws. w is not a nonlogical axiom of our theory because we have to use R3 within deductions. Hence in deductions only logically true formulae should be used. And w is not logically true.

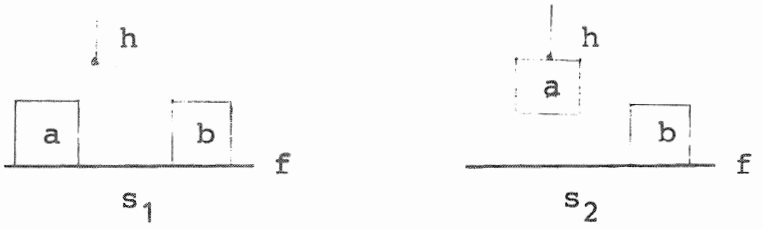


Figure 1

s_1 is described by $ONaf \wedge ONbf \wedge \neg\exists x HOLDhx \wedge \neg\exists x ONxa$

s_2 is described by $ONbf \wedge HOLDha$

Assume that there is the action "take". In this very simplified model the precondition of "take", namely CANTAKExy should be $\neg\exists z ONzy \wedge \neg\exists z HOLDxz$, and the result of "take" is HOLDxy. But we also need to prove that ONaf does not hold any more and that ONbf still holds. Hence we need an axiom for "take" which allows us to derive that if x can take y in one state then there is an immediately following state where x holds y and nothing else has changed; more generally if an action is executed in one state then there is a following state where its results hold and laws of things which are not affected by that action have not changed and still hold. We therefore suggest the following frame axiom for take:

ET $CANTAKExy \wedge p \wedge \Diamond(RESULTTAKExy \wedge p) \rightarrow \neg(p \wedge RESULTTAKExy)$

meaning that if x can take y and some proposition p holds and if p is consistent with the result of the take-action then there is a following state where p and the result of take hold. The reader should realize that $\Diamond$ means logically consistent not physically or empirically consistent: every formula which is not logically false, i.e. equivalent to $\neg p \wedge p$, is logically consistent. In particular physically inconsistent formulae like ONxx or $ATxz \wedge ATxy \wedge y \neq z$ are logically consistent. Naturally the question arises how one could prove that something is consistent or is not consistent. The idea of doing such a proof is the following: We need to add to our system axioms of the form $\Diamond A$ where A is not logically false. With respect to our example language we should add axioms like $\Diamond ONaf$, $\Diamond ONbf$, $\Diamond \neg ONbf$, $\Diamond(ONaf \wedge HOLDhb)$, $\Diamond(ONaf \wedge ONbf)$, etc. In Brown (1978) a finite axiomization of consistency axioms for a finite domain of objects is presented. It should be noted that the problems we are here concerned with have finite domains. Applying ET to s_1 we can derive that there is an immediately following state where ONbf still holds after a was taken.

$CANTAKEha \wedge ONbf \wedge \Diamond(HOLDha \wedge ONbf) \rightarrow \neg(HOLDha \wedge ONbf)$

$\Diamond(HOLDha \wedge ONbf)$ holds by the consistency axioms. On the other hand, if we set p to $w \wedge ONaf$ then $\neg\Diamond(p \wedge HOLDha)$ is derivable by

(a) $w \rightarrow (HOLDha \rightarrow \neg ONaf)$ A1

(b) $\neg(w \wedge HOLDha \wedge ONaf)$

(c) $\Box \neg(w \wedge HOLDha \wedge ONaf)$ R3

(d) $\neg \Diamond(w \wedge HOLDha \wedge ONaf)$

Hence ONaf is "deleted" ($s_1 \longrightarrow ONbf \wedge CANTAKEha$) before ET is used (i.e. ONaf cannot be one of the conjuncts substituted for p in ET.). In the following example (figure 2) by Hayes (1978) we shall discuss how the execution of one action to one object can affect a property of another object.

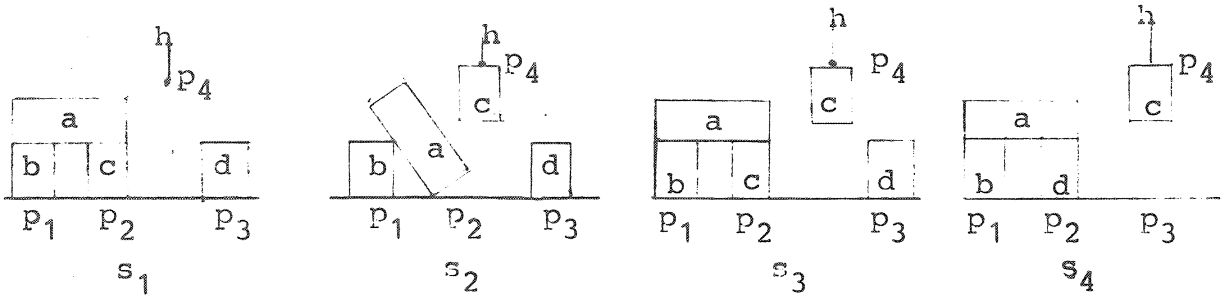


Figure 2

Let us now assume a slightly different precondition for take, namely $CANTAKExy \leftrightarrow \neg \exists z HOLDxz$ and an additional predicate SUPPORTED. The physical laws are as before plus conditions for places (p_1, p_2, p_3, p_4) stating that an object can only have one place and for SUPPORTED: namely an object is supported if it is on other objects and a long object is only supported if it is supported by at least two blocks each being under one end of that object. We shall not formalize in detail these -intuitively clear- conditions. $\neg \Diamond(SUPPORTEDa \wedge HOLDhc \wedge ATdp_3 \wedge ATbp_1 \wedge w)$ is derivable (The proof is trivial but very long). SUPPORTEDa contradicts the other statements because only b has its place under a. Now we can derive:

(1) $s_1 \longrightarrow CANTAKEhc \wedge (ATbp_1 \wedge ATdp_3 \wedge ONab)$

(2) $CANTAKEhc \wedge q \wedge \Diamond(q \wedge HOLDhc) \rightarrow \neq (q \wedge HOLDhc)$ ET

where $q \leftrightarrow (ATbp_1 \wedge ATdp_3 \wedge ONab)$

(3) $\Diamond(q \wedge HOLDhc)$ consistency axiom

(4) $s_1 \rightarrow \neq (q \wedge HOLDhc)$ (1) and (2) and (3)

Hence we have shown $(s_1 \longrightarrow \neq s_2)$

One problem with ET is that, because only logical consistency is required, we can also do derivations $s_1 \rightarrow \neq s_3$ where s_3 is

a physically inconsistent world. Since $\Diamond(ATcp_2 \wedge HOLDhc \wedge ATcp_4)$ the world s_3 is also one of the following worlds. ET does not exclude them. In order to produce only physically possible following states we could modify ET:

ET' $CANACTION \wedge p \wedge \Diamond(p \wedge RESULTACTION \wedge w) \rightarrow \neg(p \wedge RESULTACTION \wedge w)$

ET' would exclude s_3.

Hayes (1978) pointed out that s_4 is also a possible following state of s_1 because $\Diamond(SUPPORTEDa \wedge HOLDhc \wedge w)$. But s_4 follows only if d moves under a when c is taken. Hence s_4 follows only from s_1 when two actions, take h c and move d p_3p_2 take place. We cannot prove that s_4 is not a following state of s_1. But we cannot prove either that it is a following state. Hence neither $s_1 \longrightarrow \neg s_4$ nor $\neg(s_1 \longrightarrow \neg s_4)$ is derivable by our frame axiom.

Acknowledgement

I thank Frank Brown for many fruitful discussions and Pat Hayes for useful criticisms.

References

Brown, F.M. (1978) "A Sequent Calculus for Modal Quantificational Logic" 3rd AISB/GI Conference Proceedings, Hamburg.

Hayes, P.J. (1970) "Robotologic" Machine Intelligence 5.

Hayes, P.J. (1971a) "A Logic of Actions" Machine Intelligence 6.

Hayes, P.J. (1971b) "The frame Problem and related Problems on Artificial Intelligence" Stanford Artificial Intelligence Project Memo AIM-153.

Hayes, P.J. (1978) Personal Communication.

McCarthy, J. and Hayes, P.J. (1969) "Some Philosophical Problems from the standpoint of Artificial Intelligence" Machine Intelligence 4.

Rescher, N. and Urquart, A. (1971) "Temporal Logic" Springer Verlag Wien.

Sandewall, E. (1972) "An Approach to the Frame Problem and its Implementation" Machine Intelligence 7.

Schwind C.B. (1977) "AState Logic for the Representation of Natural Language based Intelligent Systems" IJCAI 5 Conference Proceedings, MIT.

REPRESENTATION AND CONTROL IN VISION(*)

by

Aaron Sloman, David Owen, Geoffrey Hinton, Frank Birch, Frank O'Gorman
Cognitive Studies Programme,
School of Social Sciences,
University of Sussex,
Brighton, U.K.

(*) The Project is funded by the U.K. Science Research Council on grant no B-RG-8688-7. Frank O'Gorman has now left the group.

Is AI vision research making good progress?

Vision work in AI has made progress with relatively small problems. We are not aware of any system in which many different kinds of knowledge co-operate. Often there is essentially one kind of structure, e.g. a network of lines or regions, and the problem is simply to segment it, and/or to label parts of it. Sometimes models of known objects are used to guide the analysis and interpretation of an image, as in the work of Roberts (1965), but usually there are few such models, and there isn't a very deep hierarchy of objects composed of objects composed of objects ... By contrast, recent speech understanding systems, like HEARSAY (Lesser 1977, Hayes-Roth 1977), deal with more complex kinds of interactions between different sorts of knowledge. They are still not very impressive compared with people, but there are some solid achievements. Is the lack of similar success in vision due to inherently more difficult problems?

Some vision work has explored interactions between different kinds of knowledge, including the Essex coding-sheet project (Brady, Bornat 1976) based on the assumption that provision for multiple co-existing processes would make the tasks much easier. However, more concrete and specific ideas are required for sensible control of a complex system, and a great deal of domain-specific descriptive know-how has to be explicitly provided for many different sub-domains.

POPEYE and HEARSAY

The POPEYE project is an attempt to study ways of putting different kinds of visual knowledge together in one system. Our philosophy has much in common with HEARSAY, including:

(a) using strategies which may not find optimal solutions, but behave sensibly in a non-trivial set of tasks, in a "friendly" world,
(b) trying to find a good global interpretation without necessarily analysing and recognising all the relevant substructures,
(c) working outwards from the "best" (e.g. biggest, least ambiguous) fragments, at any level,
(d) commitment to distributed processing, since expertise often depends

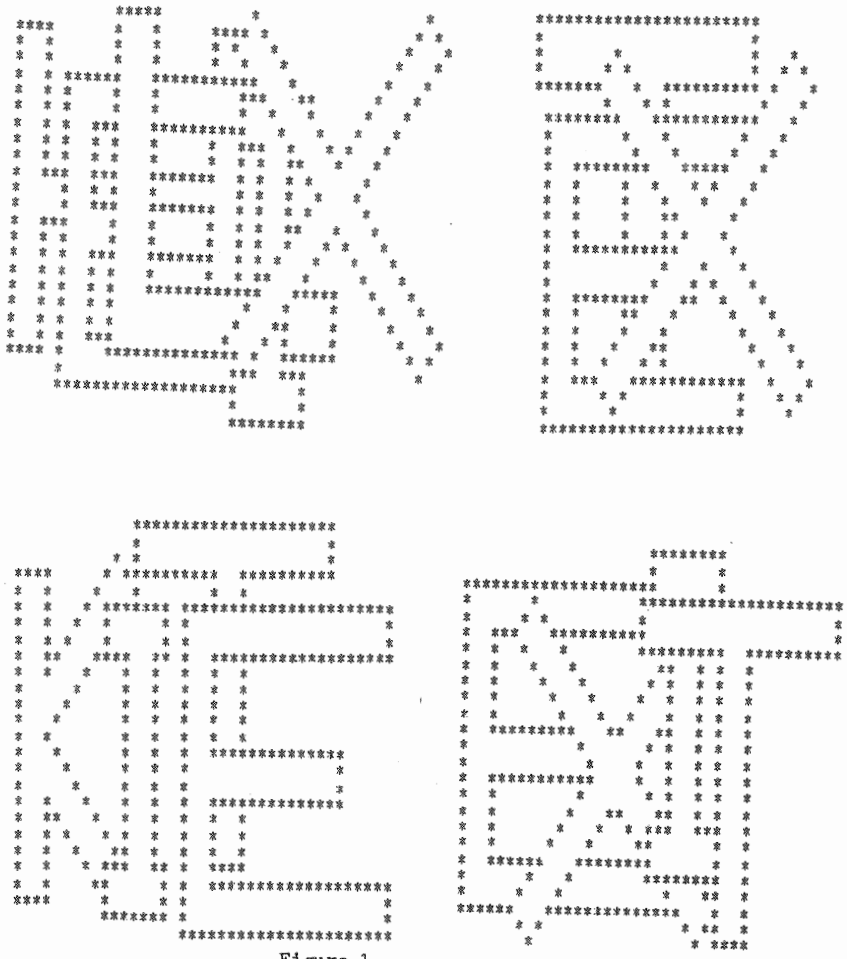
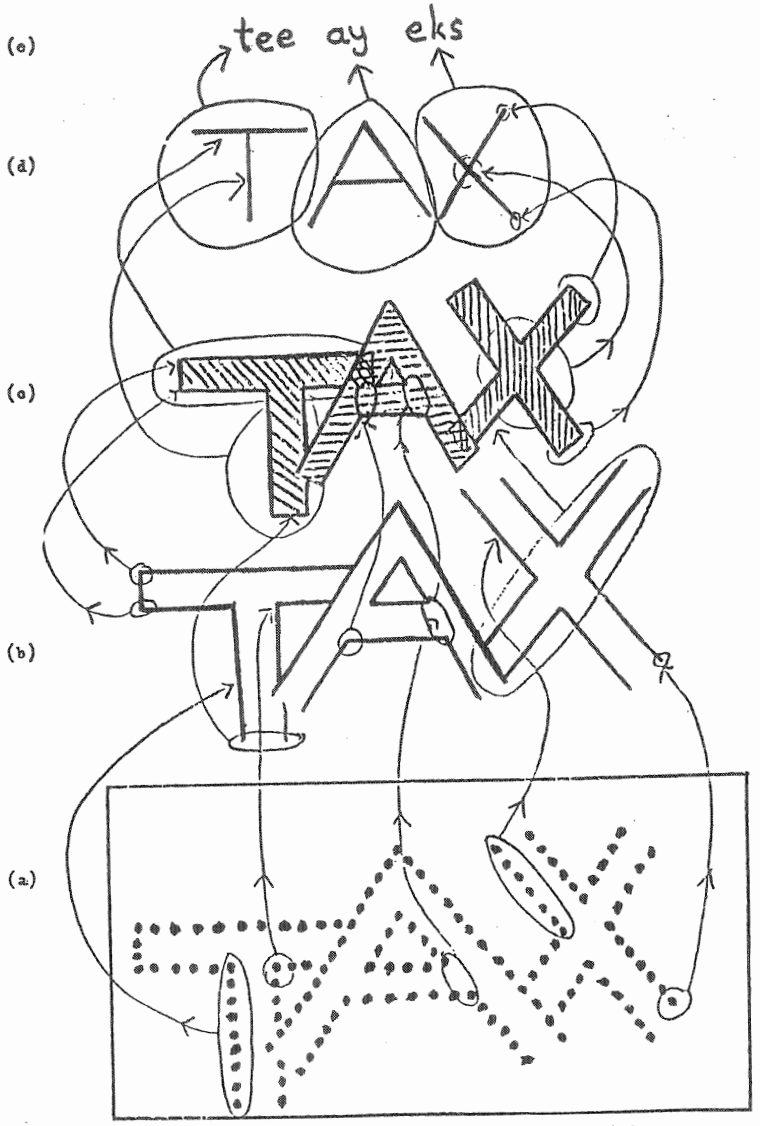


Figure 1

Examples of the pictures POPEYE analyses input as 2-D binary arrays. (Positive and negative noise can be added.) The more obvious lines, bars, bar-junctions and letters are found by the program. Current performance will be illustrated with slides at the conference.



Domains of interpretation of POPEYE pictures: - (a) configurations of dots, spaces, dot-strips etc., (b) configurations of line-segments, gaps, junctions, etc., (c) configurations of plates involving bars, bar-junctions, overlaps, edges of bars, etc., (d) stroke configurations, (e) letter sequences.

on conducting parallel searches in different domains, with partial results in one transforming the search-space in others,

(e) ordering different domains, and associated processes in a priority hierarchy, for scheduling scarce resources,

But POPEYE differs from HEARSAY. Where HEARSAY uses a general-purpose database (called a "black-board") with monitors to activate sub-systems when the contents are changed, POPEYE builds a variety of different kinds of networks (including 2-D arrays and other "analogical" representations), tailored to suit what they represent, so that, for example, searches are easily constrained by using the structure of the representation.

HEARSAY's modularity depends on uniform database procedures, whereas we achieve adequate modularity by making all sub-systems which access a particular type of sub-network use a procedure which knows how to deal with that kind of structure and knows which other sub-systems to invoke as a result. Much of the modularity of fixed-format data-base assertions can be achieved using fixed format procedure calls. Dynamic changes in such procedures are easily programmed by making them access special lists. Dynamic binding of function variables in POP2 also helps (Anderson 1976).

HEARSAY subsystems are supposed to know very little about one another, whereas, in POPEYE, messages (functions to be executed - Sloman & Hardy 1976) are sent explicitly from one sub-system to another (usually via a scheduler). We factor procedures according to whether they are concerned only with one kind of sub-system, or whether they are concerned with a message from one to another. Contrast the "frames" approach, i.e.: attach each chunk of procedural knowledge to one class of frames.

HEARSAY schedules tasks according to a numerical evaluation function combining many different measures, whereas our scheduling is based on a simple qualitative partition of sub-tasks into priority ordered classes, each with its own scheduling strategy, defaulting to a simple queue. E.g. when trying to find a word in pictures like figure 1, order tasks according to whether they are concerned with a word, a letter, a junction between bars, a bar, a line-segment, scanning for dots, etc. Choose the longest first, among line-segment jobs. Use a different ordering for a different goal, like find "all the vertical bars". So the mapping between priority levels and conceptual levels is represented by the order of a list of job-categories, and is therefore very easily altered. It is not obvious how task-dependent re-ordering of the levels in HEARSAY is achieved. Similarly a new category of sub-tasks is very easily spliced into POPEYE's priority list. The scheduling criteria used within different sub-categories need not be comparable. Long lines are (usually) more important than short ones, but the system never has to weigh length of lines against size of letters. This is consistent with what might happen in a parallel implementation.

Thus, the main scheduler selects the best jobtype, then its manager selects the best job. This makes it relatively easy to produce some major qualitative changes in performance without having to assess the interactions of several different weighting calculations. Other changes are harder to achieve without a numerical evaluation system, e.g. making the system give high priority to finding very short horizontal lines near isolated dots and large diagonal bars. But it is not clear that people can easily cope with arbitrary and complex attention-focussing instructions either.

Top-down processing in HEARSAY uses goals added to the data-base which trigger relevant procedures. In POPEYE (as in people?) much top-down processing involves a higher-level process directly sending a request to a lower-level to do its stuff in a particular region of the image space, with some of its criteria modified (e.g. if a horizontal line is wanted, then the dot-scanner should be less fussy about accepting evidence for horizontal lines in the specified region.) Again, HEARSAY allows more sensitive scheduling, whereas POPEYE is simpler and cheaper to do. The simplicity makes it more likely that a program will one day be able to do task-dependent re-scheduling. It is not obvious what the trade-offs are. Much depends on the kind of world the program is expected to cope with, on the tasks it is to perform in that world, and the costs of various kinds of errors and delays.

HEARSAY uses searching mechanisms able to cope with co-existing competing hypotheses, whereas our philosophy is to assume that when one cannot choose between alternatives one should describe what is common to them (cf. discrimination nets), or ignore them, then work on some other sub-task, hoping that emerging context will resolve the ambiguity. E.g. if relations between ambiguous limb-like shapes may unambiguously identify a human figure, then don't waste resources generating both leg and arm hypotheses for all the shapes (Paul 1976). Simpler searching is traded against more elaborate structural descriptions, in more domains (figure 2). This approach cannot cope with really difficult situations e.g. puzzle pictures. Where more problem-solving power is needed, and not merely domain-specific expertise, we are experimenting with the mechanism described by Birch and a relaxation mechanism for combinatorial search controlled by "preferences". (Hinton 1976, 1977).

Domain-specific representations

Because of the huge amounts of data in visual images and their interpretations, specialised representations are needed, so that access to information is rapid, and mutually relevant fragments may be linked to form significant cues without enormous combinatorial searches.

E.g. on finding a dot in the array, POPEYE can quickly check whether the local context is messy and ambiguous, since neighbouring dots are readily accessible. Similarly, the picture is mapped into an array of "zones" each of which knows about important image or scene fragments located there. Every "orientation" stores known ("infinite") lines in that orientation, in an ordered chain, so that each can quickly find its neighbours, and POPEYE can quickly examine all lines between two given lines. Each line lists the important line-segments and gaps which lie along it, ordered as they occur in the picture. A line-segment knows about bar-walls hypothesised as lying on it. Two bars may share a segment, one on each side. Each bar wall knows about the bar it is a wall of, and about the segment depicting it. A bar knows about its walls, its major axis, the junctions with other bars with which it merges, etc. Many image and scene structures have an intrinsic direction-system associated with them, defining a front end and a back end, a left-side and a right-side, etc. This is used by orientation-independent programs for manipulating them.

Problems of noticing significant substructures whilst such a network grows are non-trivial, and so far we have only partial solutions. Birch

(1978) describes one approach, combining recognition and segmentation, in a self-improving discrimination net.

Different categories of sub-processes distinguished by the scheduler correspond roughly to a variety of "intermediate" representations, storing partial results of processing, minimising the need for backtracking, i.e. allowing structure-sharing between different searches. (Marr (1976) calls this 'The principle of least commitment'. Cf. Sloman and Hardy 1976 p 252.) We have to examine many image and scene-fragments to discover the significant kinds of structures and relations. The hardest task is finding good descriptions, especially descriptions capable of adequately representing incompletely analysed structures, the "intermediate" representations. This study of the structure of the task domain is analogous to linguistic studies of the grammar and semantics of languages. The need to cope with partial information in a performing system leads to a richer ontology than a study of "competence".

Choosing a domain for studying vision

We are studying a variety of different sorts of images and scenes. We started with pictures like those in figure 1 for various reasons, including:

(a) People can interpret them (and can get better with practice), so there is a relevant human ability to be explained.
(b) Progressively more difficult examples can be produced, with positive or negative noise and confusing occlusions and juxtapositions, to test the program's ability to degrade gracefully.
(c) As shown in figure 2, sensible interpretation of the pictures requires knowledge about structures in different domains.
(d) Some of the image structures also arise in pictures of 3-D blocks-world scenes.
(e) Information flowing across domains enables redundancy to be used to find a good global interpretation without processing all details.
(f) Computer-generated pictures enable us to do some useful work despite the lack of TV equipment and a shortage of space in our computer.

What is heterarchy?

POPEYE is a step towards a heterarchic system which could be implemented on distributed processors. Some take "heterarchy" to refer to the use of fancy control structures. We think it is a negatively defined concept, contrasted with hierarchy. A hierarchical perceptual system is one which can be represented as a linear pipeline of processes thus:

Input ---> P1 ---> P2 ---> P3 ---> P4 ---> Output

Where arrows represent flow of data and, in a serial processor, control. A simple-minded alternative is to permit feed-back loops between sub-processes, so that not all the arrows go one way. A more complex alternative is to postulate several parallel pipe-lines through which information can flow, with additional routes for possible feed-back, feed-across, and feed-forward, as in figure 3 (optional inputs represent prior expectations). Each Pij may have several sub-processes active at different locations in the picture - (See figure 2).

In POPEYE there is a general drift of information along each pipeline and from lower pipelines to higher ones, but with occasional short-circuits. Each layer is concerned with a different domain (figure 2). Within a layer, or pipeline, information tends, on the whole, to flow from smaller to larger (higher-priority) structures. But this is only a default, over-ridden by detection of significant cues or the use of prior expectations and higher-level knowledge, causing hypotheses or requests to be formulated which can flow up or down between layers, or skip backwards or forward within a layer.

So recognition of some fragment can make the system jump to a more global hypothesis about a larger whole containing it, or an interpretation in a different domain. A high-level cue may invoke the correct global interpretation while much lower-level processing is still incomplete. The effects of such knowledge-based jumps in processing are, we believe, more important than the advantages gained from using general-purpose mathematically complete or even optimal search strategies within any one problem-space.

Short Bibliography

Anderson, Bruce 'A brief critique of LISP' in Proc. AISB Conf. 1976.

Birch, Frank et al, 'A (self adapting) network for recognition of visual structures' AISB 1978.

Bornat, R. 'Reasoning about hand printed FORTRAN programs' in AISB 1976.

Bornat, R. & J.M. Brady, 'Finding blobs of writing in the FORTRAN coding-sheets project' Proc. AISB Conf. 1976.

Brady, J.M. & B.J. Weilinga 'Seeing a pattern as a character', in Proc. AISB Conf. 1976.

Hayes-Roth F, & V.R. Lesser, 'Focus of Attention in the HEARSAY-II Speech Understanding System', in 5th IJCAI, 1977.

Hinton G. 'Using relaxation to find a puppet', in Proc. AISB Conf. 1976.

Hinton G. PhD thesis, Edinburgh University AI Dept. 1977.

Lesser V.R. & L.D. Erman, 'A retrospective view of the HEARSAY-II Architecture', 5th IJCAI 1977.

Marr, D. 'Early processing of visual information', in Philosophical transactions of the Royal Society of London, pp 483-519 1976.

Paul J.L. 'Seeing Puppets quickly', Proc. AISB Conf. 1976.

Sloman A. & S. Hardy 'Giving a computer Gestalt Experiences', AISB 1976.

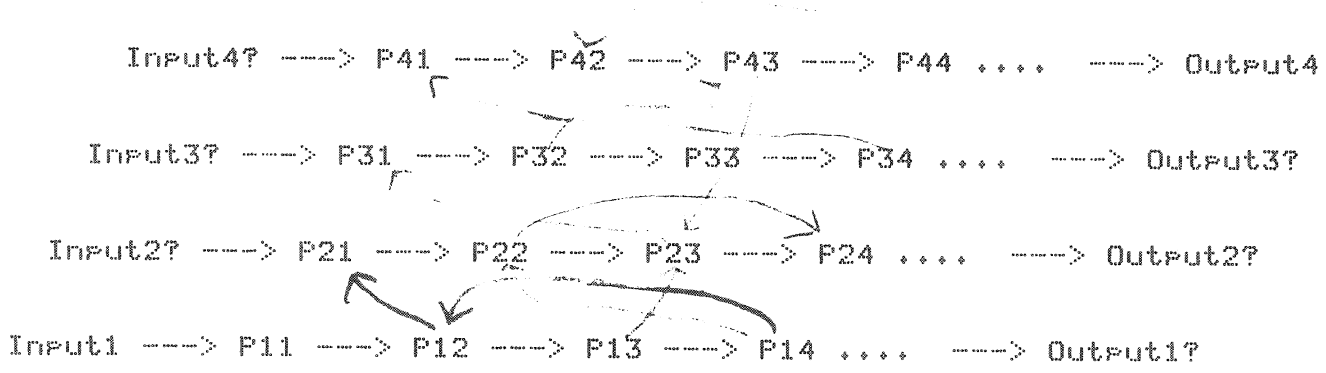


Figure 3

DESIGN FOR A PLAN HYPOTHESIZER

N.S. Sridharan and D. Smith
Department of Computer Science
Rutgers University, New Brunswick NJ 08903

A fundamental problem common to both cognitive psychology and social psychology concerns how people perceive interpersonal events. In our approach to explaining how people understand the actions they observe others performing, we assume that deriving the causal structure of the actions is the principal challenge. The nature of the causal relations is not that of physical causality but the intentional form of it, often termed "agency" or "personal causation". We have claimed [Ref 2, Ref 3] that memory for observed actions is in the form of a plan structure which consists of a set of plan units partially ordered by causal relations with well-defined conditions marking the beginning and end of plan structures. Physical causality, by contrast, yields linear temporal orderings with no bounds on the scope of beginning and end.

A plan structure representing the result of understanding actions should

a) be structurally well-formed; it should contain an orderly structure of plan units designed to accomplish the goal.

b) be attributable to the actor, i.e. be executable according to the actor's beliefs about the world.

and c) be compatible with the current state of the world.

In developing an Artificial Intelligence system to model the process of plan recognition from input consisting of a sequence of observed actions, we utilize the following three components:

a) a hypothesized plan structure, H.

b) a person model, PM, which is a set of statements the system attributes to the actor.

and c) a world model, WM, which is a set of statements the system believes are true in the current situation.

We adopt a paradigm of "Hypothesize and Revise" in an attempt to capture certain aspects of the human process of plan recognition [Ref 4, Ref 5]. This paradigm emphasizes the role of a single or very few hypotheses in guiding the interpretation of subsequent input. Specifically the hypothesis H allows the determination of the "next expected" action and the next observed action is matched to this expectation. If the match is successful, the hypothesis may get customized or refined by adding further detail. On the other hand, if the attempted match fails an analysis of the relationships among the most recent action, the hypothesis H, and the models WM and PM provide suggestions for revising the hypothesis.

This research was supported by Grant RR-643 to the Rutgers Research Resource on Computers in Biomedicine from the BRP, Division of Research Resource, NIH.

In this paper we consider one component of the plan recognition process, that of hypothesizing a plan. The reader should refer to [Ref 4,5] for a description of expectation matching and detailed specification of rules for revising the hypothesis. It is important to realize that the recognition of plan structure and the hypothesizing of a plan structure are not separable activities within our system design and that they are intimately interdependent. This article focuses on the process of hypothesizing while providing certain critical links to the recognition process.

Design Criteria for a Plan Hypothesizer

The problem posed to the plan hypothesizer is to develop a hypothetical plan structure to achieve a specified goal that may be attributed to the actor. The hypothesized plan will be used by the recognizer to guide the interpretation of incoming action descriptions.

1. The actor normally has choice in setting his goals and can select a variety of means for achieving them. Further, he can change his mind after partially executing a plan. Thus, since there is no correct plan hypothesis, there is no criterion of correctness. We only require that the plan produced be a reasonable one in the sense explained above.

2. The plan hypothesized need not be optimal. Since the plan is to be used to understand actions, the task of hypothesizing a plan should provide structures which are easily recognized. Thus optimizing the plan structure is not crucial.

3. The hypothesized plan should be attributable to the actor. Thus, the person model, PM, should be used to guide the plan hypothesizer. The world model plays a secondary role and serves to suggest and constrain the choice of objects, locations, etc. that enter into the plan structure. When the recognizer determines that certain propositions used in the current plan are true in the PM but false in the WM, it can anticipate the manner in which the actor will fail.

In view of these criteria, the plan hypothesizer specified will be similar to Sacerdoti's [Ref 6] but will not employ all of the "critics" which are found in his work. Specifically, the "USE EXISTING OBJECT" critic will not be applicable since this critic is involved with selections of specific objects so as to minimize the actions required to carry out the plan. On the other hand, the "RESOLVE CONFLICTS" critic will be applied since the actor will be prevented from carrying out actions which have been denied by preceeding actions (it should be noted that this critic will differ in its form slightly from that of Sacerdoti).

At this stage the plan hypothesizer does not use the "ELIMINATE REDUNDANT PRECONDTIONS" critic. This critic is problematic since most action sequences observed will usually take advantage of existing preconditions. However, since persons are not optimal plan generators they may not take advantage of every such possibility. We have opted to relegate this critic to that part of the plan recognition process that decides what the "next expected" plan unit will be.

Tate's work [Ref 6] also has similarities to the plan hypothesizer presented here since both systems utilize an explicit goal structure for planning and monitoring: there is, however, one essential difference. Tate's system functions in a domain which has access only to knowledge contained in a World Model. For this reason, Tate's system will generate plans applicable to the state of the world whereas our system will attribute hypothetical plans to an actor based on both the state of the world [WM] and the beliefs about that world attributed to the actor [PM].

The Process of Hypothesizing Plans

Before defining a process for hypothesizing a plan, it is necessary to specify the plan units which comprise a hypothesis and the relations between these units.

Plan Unit

a. Opportunities (OPP)
   A set of propositions which must be true in the Person Model (PM) for the plan to be undertaken.

b. Outcomes (OUT)
   A set of propositions which will be true upon completion of the plan.

c. Goals (GOAL)
   A single proposition selected from the outcome set. This unit helps give focus to the reason for the plan.

d. In-Order-To (IOT)
   A set of pairs in which the first element is the goal and the second element is a plan whose opportunities require that the goal be true. This set helps to give focus to the partial ordering of plans.

e. Not-Between (NB)
f. Not-Next (NN)

   The last three units are constructed as the hypothesis is generated. They act as Sacerdoti's conflict critic and impose a minimal partial ordering on the plans which they link together.

The process of hypothesizing a plan is performed in a "backwards" (goal directed) manner. Its inputs are a goal set which is to be realized, a proposition set which represents the Person Model (PM) attributable prior to the execution of any action in the plan, and a set of plan schemata. This process uses the goal set to select a set of plans which can accomplish the specific goals. Once this step has been performed the opportunites for the plans selected are used as goals for another application of the same step. The decision of what plans to select for satisfying the required goals is effected by looking up the outcomes and goals within the plan schemata. The selection of this set of plans determines the In-Order-To relations after which the Conflict Critic is called to compute the Not-Between sets and the Not-Next sets.

This "backwards" planning repeats the previously outlined steps until the opportunities for the plans which have not been "backed up" are a subset of the Person Model. When the opportunities meet this requirement, the plan is said to be complete and the tree structure generated is made available to the recognition process. It should be noted that the plans which now comprise the hypothesis are partially ordered and that this ordering is the most general possible (the plan has been kept as "parallel" as possible). This process of hypothesizing will produce a structure which will enable the recognizer to accept any valid sequence of actions while rejecting any sequence which violates the agreement between the opportunities and the Person Model.

The Conflict Critic

The Conflict Critic works with one a priori assumption. This assumption is motivated by the belief that once a goal has been achieved by the execution of a plan, that goal will not be reversed until it has been used for the purpose designated by the hypothesis. The formal form of the assumption is : if plan A's goal is G, plan B's opportunity is G, and plan A has been performed "in-order-to" satisfy plan B then any plan which is between the execution of plan A and B may not deny the proposition G. In other words, a plan's goal may not be undone until it has been used for the purpose intended. Note that this is a requirement on only the plan and it permits an execution sequence to achieve and re-achieve any proposition.

Once one of the steps of the process of plan hypothesizing has been performed, the conflict critic will be invoked to establish the restrictions on the temporal order of the plans which now comprise the hypothesis. The calculation of these restrictions is explained below in both English and a logical notation. The logical notation is presented to hint at the procedures for implementing this critic.

Conventions for Logical notation

OPP(Pi,x) - x is a proposition which must be true if Plan Pi is to be performed.

OUT(Pi,x) - x is a proposition which will be true immediately after Plan Pi is performed.

GOAL(Pi,x) - x is a proposition which serves to focus the reason for using Plan Pi.

$$GOAL(Pi,x) \Rightarrow OUT(Pi,x)$$

IOT(Pi,x,Pj) - Plan Pi is performed In-Order-To provide the opportunity to perform Plan Pj. Proposition x is the focus which drives the planning.

$$IOT(Pi,x,Pj) \Rightarrow GOAL(Pi,x) \wedge OPP(Pj,x)$$

NN(Pi,Pj) - Plan Pi cannot be followed by Pj.

NB(Pi,Pj,Pk) - Plan Pj may not be performed after Pi and before Pk.

Establishing the Not-Next Set

The Not-Next set is formed by considering two types of issues which affect the sequencing of actions. The first has to do with the denial of opportunity for a plan to be executed and the second pertains to the discovery of superfluous plans.

If the negation of a member of the opportunity set of Plan Pj is contained in the outcome set of Pi, then Pj cannot immediately follow Pi.

$$\{\exists\ x \mid OUT(Pi,x) \wedge OPP(Pj,\tilde{}x)\} \Rightarrow NN(Pi,Pj)$$

If an outcome of Plan Pi is itself a goal of Plan Pj then Pj cannot immediately follow Pi.

$$\{\exists\ x \mid OUT(Pi,x) \wedge GOAL(Pj,x)\} \Rightarrow NN(Pi,Pj)$$

Establishing the Not-Between Set

This set is built so that the assumption of maintenance of focus can be attained. This set is defined by a disjunction of two conditions. The first is: If the goal of Pi is an opportunity for Pk (note that this goal must remain intact until Pk uses it) and this goal denies an opportunity for Pj, then Pj cannot be performed between Pi and Pk. The second is: If the goal of Pi is an opportunity for Pk and this goal is reversed by an outcome of Pj, then Pj cannot be performed between Pi and Pk.

$$\{ \exists\ x \mid IOT(Pi,x,Pk) \wedge [OPP(Pj,\sim x) \vee OUT(Pj,\sim x)]\} \Rightarrow NB(Pi,Pj,Pk)$$

It should be noted that the Not-Between set is not dependent on the outcome set of a Plan but only on the goal set. Thus, as stated previously, the focus is maintained while at the same time allowing for subsets of opportunities to be established and denied and reestablished before the execution of a plan which required those opportunities.

Example

In order to demonstrate the construction of the Not-Between and Not-Next sets the following example will be hand simulated. This example is one of hypothesizing a plan for hanging a picture on a wall; the plans available for this hypothesizer are listed in Figure 1. By convention, the goal is listed as the first proposition in each OUTcome set.

As previously stated, the hypothesizer constructs the plan as it proceeds backwards from the given goal. For this example the goal is "Picture hung at L*". Figure 2 shows the successive stages that are encountered as this goal is backed up and the relations between the selected plans that are formed.

Conventions for Figure 2

Propositions are grouped in sections to signify OPPortunity sets for the plans selected.

Arrows with label indicate plans which have been selected to satisfy one of the propositions in the GOAL set currently being backed up.

The arrow originates from the OPPortunity set of the plan and terminates at the GOAL of the plan.

The labels on the arrows indicated which type of plan is being used.

SP-i	:	String Picture
NW-i	:	Nail in Wall
HP-i	:	Hang Picture at L*
MO-i	:	Move Object
MP-i	:	Move Person

Step 1

The first step deals with the propositions given in the problem specification. Since only one plan type (Hang Picture) satisfies this goal an instance of HP forms the hypothesis: the opportunities for Hang Picture is used to backup the proposition set. Once this is done, there are five propositions which must be satisfied at the next step.

Since only one plan is involved in the hypothesis at this time no entries may be made to the Not-Next (NN) or Not-Between (NB) sets; however, the appropriate instantiated forms of the goal, opportunity, and outcome sets will be recorded as shown in Figure 2.

Step 2

Step two must backup five propositions in parallel before the NB and NN sets are considered; however, only four plans are shown in Figure 2. This is because the proposition involving nearness of two locations cannot be planned for and may only be verified or denied when the action sequence is being observed.

Once the four plans of Step 2 are added to the hypothesis (at this point it will contain 5 plans) and the proper goal, outcome, and opportunity sets are defined, the Conflict Critic can be invoked. Recalling the definitions for the NB and NN sets and considering Figures 1 and 2, the following facts can be shown to be true.

$$OUT(MO\text{-}1,\langle Person\ at\ L1\rangle) \wedge GOAL(MP\text{-}1,\langle Person\ at\ L1\rangle) \Rightarrow NN(MO\text{-}1,MP\text{-}1)$$

$$IOT(MP\text{-}1,\langle Person\ at\ L1\rangle,HP\text{-}1) \wedge OPP(MO\text{-}1,\sim\langle Person\ at\ L1\rangle) \Rightarrow NB(MP\text{-}1,MO\text{-}1,HP\text{-}1)$$

The two facts are derived from very different pieces of information. The first fact comes from the realization that certain plans will perform superfluous actions when invoked after other plans. However, the second fact is not quite as obvious. If MP-1 is followed by MO-1 it must be the case that the person has moved from L1 to L2 after MP-1 has been performed. This would not permit the observer to attribute IOT(MP-1,<Person at L1>,HP-1) to the actor, since the actor moves away before HP-1 is done.

Step 3

This last step shows several sets of propositions being backed up in parallel. Notice that the simple proposition of <Person at Li> is assumed to be satisfiable and is never backed up. After this step, the Conflict Critic will provide further restrictions to the current partial ordering. The following list is not complete and samples the type of information that is obtained.

IOT(MP-2,<Person at L2>,MO-1) ∧ OUT(MP-1,~<Person at L2>)
=> NB(MP-2,MP-1,MO-1)

IOT(MP-2,<Person at L2>,MO-1)∧ OUT(MP-3,~<Person at L2>)
=> NB(MP-2,MP-3,MO-1)

OUT(MO-2,<Person at L3>) ∧ GOAL(MP-3,<Person at L3>)
=> NN(MO-2,MP-3)

Interaction between Hypothesizer and Recognizer

As mentioned in the introduction the hypothesizer and recognizer are heavily interrelated. One of the most obvious interrelations involves the ability to hypothesize plans and then to incrementally recognize them. As this recognition process is carried out it becomes possible to mark sections of the hypothesized plan as "phantom", based on the information from the partial order specified by the hypothesizer and on the actions observed by the recognizer. These actions provide effects which are quite similar in nature to those of Sacerdoti's "ELIMINATE REDUNDANT PRECONDITIONS" critic. Thus, as actions are observed the recognizer further restricts the hypothesized plan and thereby reduces the width of the search space with which it is working.

The recognizer gains further efficiencies by updating the definition of the "next expected" action based on knowledge of the partial ordering known prior to the observation of an action. This knowledge greatly reduces the complexity of the matching performed by the recognizer by restricting the matches only to plans which have no predecessors in the hypothesis.

Conclusion

This paper has presented a design for plan hypothesizer which can augment the predictive manner in which an observer understands the actions he perceives. This method of hypothesizing a plan has similarities with Sacerdoti's and Tate's work; however, the underlying assumptions of our system are quite different than either of the other two mentioned.

This method produces a parallel plan hypothesis partially ordered by its goal structure which can be accepted or modified by the recognizer as it interprets the observed actions. The structure of the hypothesis has been designed primarily to permit efficient recognition to proceed, rather than to expedite execution. Due to this design, the ability to optimize a hypothesis has been transferred from the hypothesizing phase to the recognition phase. The ability to recognize a plan rather than to execute one requires that hypothesizer and recognizer be intimately interdependent. The Hypothesize and Revise paradigm provides a foundation for examining the interdependence of these two tasks and relating this interdependence with the ability of people to understand actions.

A plan recognition process using the Hypothesize and Revise paradigm has been implemented on the Rutgers PDP-10 in a combination of AIMDS, FUZZY, and LISP. The plan hypothesizer is expected to be an integral part of this system.

References

1. E. D. Sacerdoti, The Nonlinear Nature of Plans. Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tblisi, USSR, 1975, p206-214.

2. C. F. Schmidt, N. S. Sridharan and J. L. Goodson, Recognizing Plans and Summarizing Actions. Proceedings of the Summer Conference on Artificial Intelligence and Simulation of Behavior, Edinburgh, July 1976, p291-306.

3. C. F. Schmidt, Understanding Human Actions. In B. L. Nash-Webber and R. C. Schank (editors), Theoretical Issues in Natural Language Processing: An Interdisciplinary Workshop in Computational Linguistics, Psychology, Linguistics, and Artificial Intelligence, M.I.T., Cambridge, 1975, p196-200.

4. C. F. Schmidt and N. S. Sridharan, Plan Recognition Using a Hypothesize and Revise Paradigm: An Example. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, M.I.T., Cambridge, August 1977, p480-488.

5. C. F. Schmidt, N. S. Sridharan and J. L. Goodson, The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence. Artificial Intelligence, Special Issue on Applications to the Sciences and Medicine, 1978 (in press).

6. A. Tate, Project Planning Using a Hierarchic Non-linear Planner, Department of Artificial Intelligence, Report No. 25, University of Edinburgh, 1976.

Plan Name : String Picture at L*

OPP set : <Picture at L*>
<Tools at L*>
<Person at L*>
<Parts at L*>

OUT set : <Picture Strung>
~<Parts at L*>

Plan Name : Nail in Wall at L*

OPP set : <Nail at L1>
<Tools at L1>
<Person at L1>
<L* near L1>

OUT set : <Nail in Wall at L*>
~<Nail at L1>

Plan Name : Hang Picture at L*

OPP set : <Picture at L1>
<Picture Strung>
<Person at L1>
<L* near L1>
<Nail in Wall at L*>

OUT set : <Picture Hung at L*>
~<Picture at L1>

Plan Name : Move Object to L*

OPP set : <Object at L1>
<Person at L1>

OUT set : <Object at L*>
~<Object at L1>
<Person at L*>
~<Person at L1>

Plan Name : Move Person to L*

OPP set : <Person at L1>

OUT set : <Person at L*>
~<Person at L1>

Figure 1

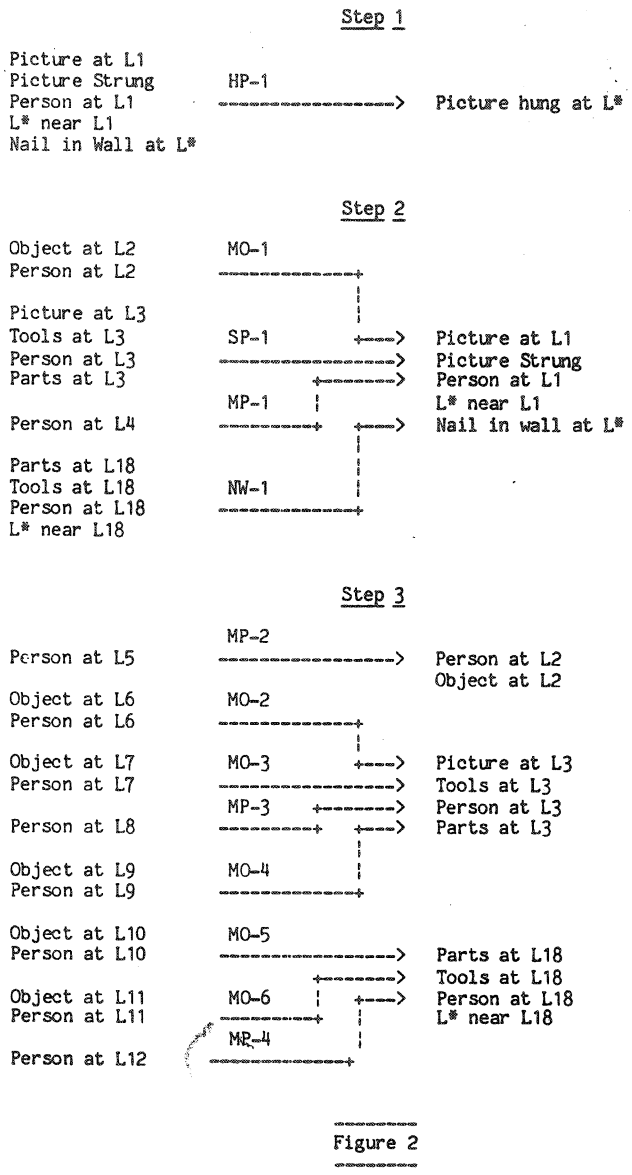


Figure 2

THE FRAME SHIFT MECHANISM IN NATURAL LANGUAGE

Luc Steels
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge Massachusetts 02139 USA

The frame shift mechanism is a method to resolve the conflict between the goals of a certain language communication and the tools available for turning these types of goals into natural language. It consists in assembling a new specification relating cognitive goals with natural language means.

This paper reports on one aspect of a global system for natural language communication currently under construction. The system has been set up as a testbed for a number of ideas about natural language understanding, production and learning. Major portions of the theoretical basis have been developed and several aspects have already been implemented in an experimental ACTOR-based problem solving language in which cognitive processing is performed by communication between experts. In this paper one idea will be discussed very briefly.

Warning ! Due to space limitations it was impossible to provide even a sketchy overview of the main principles underlying the whole system or to give sufficient detail to make the examples fully understandable for someone unfamiliar with the theory. The reader should therefore concentrate on the main idea rather than worry about the details.

1. Assume that in the language system there is a knowledge structure known as the *dictionary*. It relates "semantic" specifications or *factors* concerning meaning, semantic function, semantic type, viewpoint, etc. to an underlying word form, i.e. a root or stem related to an actual word by morphological rules. The factors are properties of the structures which serve as an interface between cognitive systems and the language producer or parser.

In the system under discussion a cluster of a root with its semantic specifications is represented as a frame, where the various semantic specifications and the underlying word form are the slots. Partly to guide the search, the frames are organized into a hierarchical frame system on the basis of shared subspecifications . The gradual refinement of specifications gives the dictionary the shape of a discrimination net. Inheritance and default mechanisms compensate for this type of organization during "bottom up" consultation.

2. Each factor that occurs in the dictionary has a number of *structural implications.* For example the factor semantic function implies a certain relational environment, a certain order environment, certain morphological features of the root and a certain syntactic features environment. Or each semantic type implies a specific set of possible cases, and a specific set of case indicators such as morphological endings, orderings or prepositions.

Similar things are available for other factors.

In the system under discussion a specification defining the structural implications of a factor is again represented in terms of a frame with the structural features as slots. And again these frames are organized into a frame system on the basis of shared subspecifications. For example the premodifiers are a subclass of the attributive adjuncts which are a subclass of the adjuncts, etc.

The set of frame systems relating factors to their structural properties is called the *grammar*. Depending on the factor one can make a distinction between relational grammar, case grammar, and so on.

3. Now I will introduce the topic of this paper by telling a hypothetical story about the development of language. Assume that initially there was a dictionary frame for all factor valuations the cognitive system was ever interested in.

For example, there is one frame for RED, with meaning "red", semantic function attributive adjunct (as in 'the red table'), semantic type STATE, etc. The root 'red' can then only be used with meaning "red", semantic function attributive adjunct, type STATE and so on.

But now suppose that the cognitive system wants to find a root with the same values for all factors except one. For example the meaning, type, perspective would all remain constant but instead of attributive adjunct, a (grammatical) object is required (as in 'Red is a nice color'). Now there is a problem: no dictionary frame exists for this specification.

There are essentially three ways to solve this problem.

First the grammar could tell the cognitive system responsible for the initial structure that this structure is not realizable and that therefore some restructuring must take place. This kind of backtracking is presumably very expensive, partially because inferencing and conceptualizing are very complicated processes in themselves.

A second solution is to expand the dictionary by introducing a new frame with all specifications identical except the conflicting factor. From the point of view of the grammar this is obviously the most efficient solution (although other language users have to agree of course). However the dictionary is already extremely large and extension should be avoided at all cost.

A third solution is the one natural languages seem to have adopted: *the frame shift mechanism.*

The frame shift mechanism is a way to resolve the conflict between a specification offered by the cognitive system and the available specification in a dictionary frame by activating a certain global method to construct a new specification. Typical methods are addition of certain structural signals to indicate the shift or inheritance of (some of) the structural properties of the factor value that caused the conflict.

The big thing is that the method does not only work in one situation but for all conflicts of the same type. The specification of a method itself is in the present system again represented in frames where the slots now deal with the aspects that are to be added or taken over. I will call these frames *transfer frames.* Transfer frames are also organized in a frame system on the basis of shared subspecifications. The transfer frames are activated when the "complaint departments" of the dictionary or grammar consulting agents receive messages signalling unsolvable conflicts.

Natural languages continue to change. A natural development concerning frame shift is that very frequent shifts are frozen into the dictionary as refinements of

the already available frames. Doing this will result in increased efficiency for the consultation process, the transfer frames are being compiled so to speak. But because a certain transfer frame is no longer productive some of the flexibility of the language system is lost. After some generations the information to do the shift will no longer be available in the transfer frame knowledge structure.

Frame shifts appear to have certain interesting properties, e.g. frame shift due to conflict in one factor may cause frame shifts for other factors. Due to space limitations these cannot be discussed here, instead I will introduce (very briefly) some examples.

4.1. *Functional shift.* Assume that each (underlying) language word implies one semantic function (or grammatical relation depending on the way you look at it). For example 'poor' is always used attributively, i.e. to amplify the introduction of an entity by a (grammatical) object, 'girl' is always used as a grammatical object to introduce a certain entity, 'walk' is always used as a verbal adjunct, etc.

With the frame shift mechanism we are able to extend the use of each underlying word to other semantic functions. For example: 'poor' can be used as a grammatical object to introduce entities as in 'he gives to the poor'. Note the following transfer characteristics in the frame shift at hand

(i) Certain structural aspects of grammatical objects have been "inherited". For example 'poor' now takes the position of a normal object and it creates a certain relational environment common for grammatical objects in the sense that a determiner (here 'the') and a case indicator (here 'to') can take part in it.

(ii) Certain structural properties of attributive adjuncts remain however. For example a superlative is still possible ('The poorest of the poor live here'), certain aspects of the relational environment are preserved , e.g. an adverbial adjunct, not allowed with objects, can still be connected.

(iii) Certain structural properties of grammatical objects are NOT inherited. For example only generic use is allowed ('A poor came here yesterday' is ungrammatical).

(iv) There seem to be no special signals indicating that the shift has taken place.

Frame shifts concerning the semantic function are so common that you can almost shift from each semantic function to each other function.

4.2. *Meaning shift.* Assume that each (underlying) language word implies one meaning (in the sense of a predicate or property the cognitive system knows about.) For example 'red' is always used to denote "having a particular color", 'girl' is always used to denote "a (young) person of the female sex", 'walk' is always used to denote "a particular movement", etc.

With the frame shift mechanism we are able to extend the use of each underlying word to other meanings. For example 'red' can also be used with the meaning of "revolutionary or anarchistic", as in 'the red brigade'. The brigade does not have the color red, rather it is a group of people with revolutionary or anarchistic ideals. Also note that this frame shift does not have any "structural implications", even the possibility of performing a functional shift to grammatical objects is preserved.

Another very common meaning shift is from so called "literal" to "nonliteral" usage. A certain property which has been introduced for physical entities is by analogy applied to abstract ones, as e.g. from 'clear glass' to 'a clear idea', or from 'a sharp knife' to 'a sharp intelligence'.

The phenomenon of meaning shift is so common that almost any natural language word has some "shifted" meanings. In most cases (as with 'red' as

'revolutionary' or so called idioms) these shifts are probably frozen in the dictionary, but other shifts are still productive (e.g. use of quotation marks). Most meaning shifts do not have structural properties signalling them (although consider the shift from 'red' to 'reddish'). This makes it particularly hard for language understanding systems to recover them.

4.3. *Type shift*. In the system under discussion (underlying) words are classified into several types, such as STATE, where an entity is said to have a certain property, EVENT, where something is said to happen to a certain entity, or ACT where a certain entity does a certain thing so that something happens to another entity. This classification governs two things: the kind of cases and circumstantials that may occur with the underlying word form and the surface case features that will be used to introduce the cases associated with the predicate expressed by this root.

Semantically speaking the type refers to the way a certain "scene" is perceived. When the type being perceived is different from the bias type of the root chosen to characterize the scene, a type conflict occurs and the frame shift mechanism comes into action. Without going in great detail here are two examples.

- 'Open' is typically a STATE word, as in 'the door is open'. Suppose now that we see a scene where something happens to the door such that it is open now. We want to describe this scene as an event : something (become open) happens to the door. But there is a conflict with open as STATE and the perception of the scene as an EVENT. The so called "inchoative" frame shift allows us then to say 'The door opened'. Note that not only a type shift but also a functional shift has occured.

- Suppose we have another scene where something happens to the door such that it is open now and we want to incorporate in the natural language description another entity that initiated this particular event. So now we perceive the scene as an ACT . The so called "causative" frame shift allows us to obtain 'He opened the door'. Note that there are no structural indicators distinguishing between EVENT and ACT. Note also that in the case that 'open' is an event it is not possible to express the initiator ('The door opened by him' is ungrammatical).

The phenomenon of type shift is again so common that it is possible to shift from almost any type to any other type. Most of these shifts are still very productive and rarely are there structural signals to indicate it.

4.4. *Viewpoint shift*. In the system under discussion another semantic parameter of great importance for the construction of the specification of the surface case features is that of viewpoint. A viewpoint is one case slot by which a predicate-case structure is connected to the rest of the communication. It is the entity with which the language user approaches a certain scene. For example in 'John paints the door' the viewpoint of 'paints' is the initiator of the action, John. Each dictionary frame contains a "bias" viewpoint. E.g. for 'paint' the bias viewpoint is initiator. Sometimes (as with buy and sell) there is a different root depending solely on a difference in viewpoint.

What happens if the language user wants to approach the scene from a different viewpoint than the one present in the specification which is adequate as regards the other factors ? A frame shift occurs with a new case now filling the viewpoint.

For example we go from 'John paints the door' to 'The door was painted by John', or from 'boys stealing a car' to 'a car stolen by some boys', etc. The structural features indicating the shift include different word ordering, use of morphological features and a certain effect on the surface case features.

There are many more examples of frame shifts (for example at the categorial level, such as from 'beauty' to 'beautiful' or at a pragmatic level, such as from an interrogative to a declarative in so called rhetorical questions) but I hope that the examples given will suffice to give an idea of what is going on here.

5. *Comments.* As far as I can tell no language processing system currently in existence deals in a decent manner with the phenomena identified here. Usually the result of frame shift is incorporated explicitly in the language description itself. This is not the best approach for two reasons. First of all the grammar is more complex than necessary, leading to more storage and reduced efficiency. Second some frame shifts cannot be foreseen because they are still very productive (consider for example use of metaphor).

Linguists began to recognize various shifts (especially on the categorial, relational and lexical level) from the late twenties onwards (cf. Bally's transposition, Benveniste's transfer, Tesniere's translation, etc.). Following Harris the transformational school studied the phenomena concerning frame shift by postulating a "kernel" representation and transforming it based on the various possible shifts. It is now well known that this approach leads to unsurmountable problems. Within transformational grammar these problems have lead to the development of a "lexicalist" approach and the introduction of various "hacks" such as the so called X-bar theory.

Apart from the fact that in a generative grammar the function of frame shift phenomena cannot be explained, the major difference with the transformational approach and the one presented here is that the mechanism of frame shift is a feature of the process of *consulting* linguistic knowledge. It is as if the grammar tries to resolve conflicts internally before the control structures of the parsing and production processes incorporate the results in their respective intermediate linguistic structures.

It is fairly obvious that this new way of thinking forces a complete revision of current grammatical description and although many things have to be investigated, results so far obtained are more than promising.

Acknowledgment

I am indebted to Roger Duffey, Kenneth Kahn, David McDonald and Marvin Minsky for comments on the form or content of this paper. This research was sponsored by IIE through an ITT fellowship and conducted at the MIT AI lab.

PLANNING IN UNCERTAIN ENVIRONMENTS THROUGH SITUATION CALCULUS

Olga Štěpánková
Institute for Computation Techniques, ČVUT
Horská 3, 128 00 Praha 2

ABSTRACT

We try to broaden the scope of the situation calculus developed in the tradition of Green /69/, Mc Carthy, Hayes /69/ to cope with uncertainity. The situation dependent object functions are used for that purpose. Our method is explained on examples of "cat and mouse in a maze" and a BRIDGE termination. We shall impose some restrictions on the world, the robot and its goals under which a theorem prover can carry the planning of robot's activity in a changing or not completely known environment. Finally we show how the activity of the executing robot of Siklóssy can be planned in our formalism.

1. INTRODUCTION

Most of the current simulated robots are able to cope only with a totally passive environment - the robot's actions are the only cause of the changes in the world. Moreover it is supposed that the robot knows everything (that can be expressed in its language) about its environment. The exceptions, among those stable and known worlds, are the systems proposed by Mc Carthy and Hayes /69/, Hendrix /73/, Young /76/ and Siklóssy /73/. The first one sketches a rich formalism for representation of knowledge and abilities, including some changing properties, e.g. name-on-street-sign(s) or description of a position of a failling body. Hendrix simulates the world which has "... the ability to model growing flowers, running streams and a sun, which wanders gradually across the heavens". The Young's system rationale is to understand the development of a child's model of the world and its ability to acquire new production nets for description of building and falling in the block world.

Siklóssy /73/ argues that a planning robot in an uncertain environment is inadequate and contrasts it with a robot of a new design - executing robot.

Let the world of a robot be described by means of situation calculus (e.g. Štěpánková, Havel /76a/). This is a subclass of sorted theories of first order predicate calculus with two main sorts - the sort "object" and "situation". Obviously the first sort refers to all objects of the robot's world and it can be divided into several different subsorts, if needed. The object constants describe the objects as robot, mouse, natural number, etc.

The only situation constant is s_0, interpreted as the initial state. The situation terms (similarly as in Green /69/) are results of the application of the functions, describing

operators of the robot, with some object parameters, on situation terms. Let A be a formula and t a situation term. We say that A is uniform with respect to t (uniform w.r.t.t) if A does not include any quantification over situation variables and if any term maximaly occuring in A is either object term or t itself. If A is uniform with respect to s or $\varphi(\bar{x},s)$ then $A[\![t]\!]$, $A[\![\varphi(\bar{x},t)]\!]$ resp. denote the formula A after substitution of t for any s.

The intended interpretation of the situation term $\varphi^\circ(\bar{a},t)$, where φ is a robot's action, $\bar{a}$ is a sequence of object parameters and t a situation term, is the situation corresponding to the state of the world after application of the action $\varphi(\bar{a})$ to the world described by the situation term t. Robot's operator φ is specified by formulas C_φ, R_φ giving the prerequisite and the result of φ, by parameters $\bar{x}_\varphi$ and by a set $Inc_{\varphi(\bar{x}_\varphi)}$ of pairs of formulas uniform with resp. to s, $\varphi^\circ(\bar{x}_\varphi,s)$ resp. enumerating the incidental phenomena of $\varphi(\bar{x}_\varphi)$ (Štěpánková, Havel /76b/). The incidental phenomena describe the frame of an action:
Let $\bar{a}$ be a sequence of variable-free object terms of the same length as $\bar{x}_\varphi$, let t be a variable-free situation term. If $\langle A,B\rangle \in Inc_{\varphi(\bar{x}_\varphi)}$ and $C_\varphi(\bar{a})$, A hold in situation t, then B must hold in $\varphi^\circ(\bar{a},t)$. The corresponding axiom is $C_\varphi(\bar{a})[\![t]\!] \& A[\![t]\!] \rightarrow B[\![\varphi^\circ(\bar{a},t)]\!]$.

Some features of the world are described by the object functions which depend on a situation parameter, e.g. "the room, where the mouse is in situation s" - pmouse(s), "the blood preasure of Mr A after five minutes of running" - blood preasure (A, running°(5,s)). This is an important difference from standard situation calculus. Our formalism gains a new strength from the use of situation-dependent object functions. The terms involving these functions have full freedom to be used as parameters of robot's actions. How it works and what sort of problems it causes will be illustrated by the following examples.

2. TOM AND JERRY EXAMPLE

A cat lives in a maze of rooms 1 - 9 (see Figure 1), some of which are connected by doors. Let us adopt the axiom

$$\forall s \; \forall x \; (\neg \, DOOR(x,x,s))$$

The cat can

a) Walk through the maze using the operator goto°(x,y,s), the prerequisite $C_{goto}(x,y,s)$ of which is "the cat is in room x and there is an open door between x and y" and the result is "the cat is in the room y". The axiom describing this action is

$$(IN(cat,x,s) \;\&\; DOOR(x,y,s)) \rightarrow IN(cat,y,goto^\circ(x,y,s))$$

The incidental phenomena $Inc_{goto(x,y)}$ of goto°(x,y) include for any z_1, z_2

$$\langle DOOR(z_1,z_2,s) \;,\; DOOR(z_1,z_2,goto^\circ(x,y,s)) \rangle$$

$$\langle \neg DOOR(z_1,z_2,s) \;,\; \neg DOOR(z_1,z_2,goto^\circ(x,y,s)) \rangle$$

b) Open (close) the doors between any two adjacent rooms using some mechanism situated in the room No.1. The conditions of both these actions $\underline{open}^{\circ}(x,y,s)$, $\underline{close}^{\circ}(x,y,s)$, are the same "the cat is in the room No.1 and the rooms x,y are adjacent" the results are "the door between x,y is open", "the door between x,y is closed", resp. The corresponding axioms are

$$(IN(\underline{cat},1,s) \;\&\; ADJ(x,y,s)) \rightarrow DOOR(x,y,\underline{open}^{\circ}(x,y,s))$$

$$(IN(\underline{cat},1,s) \;\&\; ADJ(x,y,s)) \rightarrow \neg DOOR(x,y,\underline{close}^{\circ}(x,y,s))$$

The incidental phenomena of the operator $\underline{open}(x,y)$ include

$$\langle IN(\underline{cat},1,s),\ IN(\underline{cat},1,\underline{open}^{\circ}(x,y,s))\rangle$$

If $x \neq z_1$ & $y \neq z_2$ or $x \neq z_2$ & $y \neq z_1$ then

$\langle DOOR(z_1,z_2,s), DOOR(z_1,z_2,\underline{open}^{\circ}(x,y,s))\rangle$,

$\langle \neg DOOR(z_1,z_2,s), \neg DOOR(z_1,z_2,\underline{open}^{\circ}(x,y,s))\rangle$ are from $Inc_{\underline{open}}(x,y)$

Similarly for $\underline{close}^{\circ}(x,y,s)$.

There lives also a mouse in this maze. The mouse is all the time on the run. It reacts on any cat's action by a move to one of the rooms which is adjacent to its present position and the door of which is open. Let us suppose, that the mouse's move is simultaneous to any cat's action - they end at the same time. Let the position of the mouse be denoted $\underline{pmouse}(s)$. Then it holds

$$\forall s \quad IN(\underline{mouse},\underline{pmouse}(s),s)$$

Moreover let the cat be able to detect somehow the actual position of the mouse in the maze. The acquisition of this knowledge is not accompanied by any change in the world - the last axiom does not change the situation. Three possible different states can result from the original one after cat's move to the room 1 in Figure 2.

We are no more in the world with a sole agent. The result of the cat's action cannot be reliably predicted - it depends on the mood of the mouse in many cases. The actions of the mouse change the world but these changes should not be included in the description of the cat's operators-these are much more general and could be used even in different problem domains. Such changes are again incidental phenomena of cat's actions:

Let the mouse be in room z_1 and $\varphi(x,y,s)$ let be any cat's action different then $\underline{goto}^{\circ}(x,z,s)$, $\underline{open}^{\circ}(z,y,s)$, $\underline{open}^{\circ}(z,y,s)$, $\underline{close}^{\circ}(z,y,s)$, $\underline{close}^{\circ}(x,z,s)$, than

$\langle IN(\underline{mouse},z_1,s) \;\&\; (\exists z_2)(DOOR(z_2,z_1,s));$

$\underline{pmouse}(\varphi(x,y,s)) \neq z_1 \;\&\; DOOR(\underline{pmouse}(\varphi(x,y,s)),z_1,\varphi(x,y,s))\rangle$

$\langle IN(\underline{mouse},z_1,s) \;\&\; (\forall z_2)(\neg DOOR(z_2,z_1,s));$

$\underline{pmouse}(\varphi(x,y,s)) = z_1\rangle$

are from $Inc_{\varphi(x,y)}$.

Moreover suppose that it holds for the remaining actions: the mouse does not try to slip through the cat's paws, i.e. Inc $_{\underline{goto}(x,z_1)}$ include the pairs :

$\langle IN(\underline{mouse},z_1,s)\,\&\,(\exists z_2)(z_2 \neq x \;\&\; DOOR(z_2,z_1,s))\;;$
$\quad \underline{pmouse}(\underline{goto}^{\circ}(x,z_1,s)) \neq z_1 \;\&\; \underline{pmouse}(\underline{goto}^{\circ}(x,z,s)) \neq x \rangle$

$\langle IN(\underline{mouse},z_1,s)\;\&\;(\forall z_2)(z_2 \neq x \rightarrow \neg DOOR(z_2,z_1,s))\;;$
$\quad \underline{pmouse}(\underline{goto}^{\circ}(x,z_1,s)) = z_1 \rangle$

The mouse is afraid of a moving door and refrains from its use, Inc $_{\underline{open}(z_1,y)}$ includes

$\langle IN(\underline{mouse},z_1,s) \;\&\; \exists z_2(z_2 \neq y \;\&\; DOOR(z_2,z_1,s))\;;$
$\quad \underline{pmouse}((\underline{open}^{\circ}(z_1,y,s)) \neq z_1 \;\&\; \underline{pmouse}(\underline{open}^{\circ}(z,y,s)) \neq y \rangle$

$IN(\underline{mouse},z_1,s) \;\&\; (\forall z_2)(DOOR(z_2,z_1,s) \rightarrow z_2 = y)\;,$
$\quad \underline{pmouse}(\underline{open}^{\circ}(z_1,y,s)) = z_1 \rangle$

The cat's problem is given by the initial and the goal situation on Figure 3. The cat knows that the following formula (denoted $X^{\circ}[\![s_0]\!]$) holds

$$X^{\circ}[\![s]\!] = IN(\underline{cat},2,s_0) \;\&\; IN(\underline{mouse},5,s_0) \;\&\; DOOR(1,2,s_0) \;\&\; DOOR(5,8,s_0) \;\&\; DOOR(5,6,s_0) \;\&\; DOOR(7,8,s_0) \ldots$$

and seeks the state in which the formula $Y^{\circ}[\![s]\!]$ holds

$$Y^{\circ}[\![s]\!] = (IN(\underline{mouse},8,s) \;\&\; (\forall y)\; \neg DOOR(y,8,s)) \vee (IN(\underline{mouse},6,s) \;\&\; (\forall y)\; \neg DOOR(y,6,s))$$

It is not difficult to prove from the above axioms and some additional ones describing the geometry of the maze that $X^{\circ}[\![s]\!] \rightarrow (\exists s)Y^{\circ}[\![s]\!]$. After skolemisation these axioms form an open theory. The Hilbert-Ackerman theorem can be used to specify a finite set $\mathcal{T}$ of variable free situation terms for which it is provable, that

$$X^{\circ}[\![s_0]\!] \rightarrow \bigvee_{t \in \mathcal{T}} Sb^{s}_{t}(Y^{\circ}[\![s]\!])$$

($Sb^{s}_{t}A$ denotes substitution of t for s in A).

There are several such sets in this case, e.g.

$\mathcal{T}_0 = \{\underline{close}^{\circ}(5,\underline{pmouse}(\underline{goto}^{\circ}(1,2,s)),\ \underline{goto}^{\circ}(1,2,s))\}$

$\mathcal{T}_1 = \{\underline{close}^{\circ}(5,6,\underline{goto}^{\circ}(1,2,s));\ \underline{close}^{\circ}(5,8,\underline{goto}^{\circ}(1,2,s))\}$

$\mathcal{T}_2 = \{\underline{close}^{\circ}(5,6,\underline{open}^{\circ}(1,2,\underline{close}^{\circ}(5,8,\underline{goto}^{\circ}(1,2,s))))\}$.

The first set $\mathcal{T}_0$ suggests a most intuitive solution of the above problem: "a cat goes to the room 1 and closes the door through which the mouse left the room 5 (reacting to the cat's move)". The solution $\mathcal{T}_1$ represents a branching plan :

1. goto°(2,1,s)
2. if the mouse is in room 8 then close the door between 5 and 8, otherwise close door between rooms 5,6.

This solution needs again the cat's ability to detect the immediate position of a mouse. In this case there exists also a solution Υ_2 which is rather long but independent of the cat's ability to identify the mouse's position. The correctness of this last solution is proved by the series of drawings - Figure 4.

Comparison of the solution sets Υ_0, Υ_1, Υ_2 shows that the solutions Υ_0, Υ_1 using the ability of the cat "to find out the immediate position of the mouse" are more understandable, easier to prove correct and also shorter. The number of cat's actions in the solutions Υ_0 and Υ_1 is the same but it is easier to establish it from the solution Υ_0, because the need for detection of mouse's position in Υ_1 is hidden in the necessity to decide which path in the branching plan to follow.

3. DESCRIPTION OF A TWO PERSON GAME

The use of state dependent object functions allows to describe also different plays in predicate calculus. How could the axioms look like for some kind of two person whist ? Let P(i,s), i = 1,2 mean that the person i is to play one of his cards.

The action of discarding x of the first person play1(x,1) is specified by its obvious condition

$$C_{play\,1}(x)[\![s]\!] \;=\; P(1,s)\ \&\ IN\ HAND(1,x,s)$$

The result of play 1°(x,s) describes the world in which the second player already answered by discarding one of his cards:

$$C_{play\,1}(x)[\![s]\!] \rightarrow \{ONTABLE(1,x,play\ 1^{\circ}(x,s))\ \&$$

$$\&\ (\forall i)(\neg P(i,play\ 1^{\circ}(x,s)))\ \&$$

$$\&\ (\exists y)[ONTABLE(2,y,play\ 1^{\circ}(x,s))\ \&\ INHAND(z,y,s)\ \&$$

$$(suit(x) = suit(y) \vee (\forall z(INHAND(z,z,s) \rightarrow suit(x) \neq suit(z))\ \&$$

$$suit(y) \neq suit(x)))]\}$$

During skolemization of this axiom the existential quantifier $\exists y$ is replaced by the function h(x,play1°(x,s)) - the value of which can be obviously determined only in the state corresponding to the situation play 1°(x,s). The incidental axioms make sure that the cards different than x and h(x,play 1°(x,s)) stay in hands of the players as before, e.g. $z \neq x$, $z \neq h(x,play\ 1^{\circ}(x,s))$

$$\langle INHAND(i,z,s)\ ,\ INHAND(i,z,play\ 1^{\circ}(x,s))\rangle \in Inc_{play\,1}(x)$$

Dually, with the condition P(2,s), could be axiomatized the action of discarding of the second player - play 2°(z,s).

Let the function count(i,s) denote the amount of pairs of cards the player has gained up to the situation s . The cards are ordered, their ordering is denoted $\subset$. The next possible move is to take :

$$(\forall i)(\neg P(i,s)) \;\&\; ONTABLE(1,x,s) \;\&\; ONTABLE(2,y,s)) \rightarrow ((suit(x) \neq suit(y) \;\&\; x \supset y) \rightarrow count(1, take^{0}(x,y,s)) = count(1,s)+1) \;\&\; \ldots$$

The goal in the play is to reach a state in which

$$(\forall y)(\neg INHAND(INHAND(1,y,s) \;\&\; INHAND(2,y,s)) \;\&\; count(1,s) \quad count(2,s)$$

Suppose we are interested only in an ending of a play described by the following facts:

1) First player is to play, i.e. $P(1,s)$ holds
2) $count(1,s) = count(2,s)+1$
3) both players have in hand just 2 cards left
   $(\forall i=1,2)(\forall z_1,z_2,z_3)(\bigwedge_{j\neq 3} INHAND(i,z_j,s) \rightarrow z_1=z_2 \vee z_1=z_3)$
4) both cards of the second player have the same suit
5) the cards of the first player have two different suits.

The function f(y,s) determines a card which has the player 1 in s in hand and which is of a different suit than y

$$(\forall y)(suit(f(y,s)) \neq suit(y) \;\&\; INHAND(1,f(y,s),s))$$

It can be proved from the above axioms and the facts about the initial situation s_0 that the goal state of the first player can be reached, i.e. $(\exists s)Goal(s)$ is provable. The situation term describing this situation is as follows:

denote $s_1 = play\ 1^{0}(f(a),s)$, $a = h(x, play\ 1^{0}(x,s))$ then the desired situation is

$$play\ 1^{0}(f(f(a)), take^{0}(f(a), h(f(a),s), play\ 1^{0}(f(a),s))).$$

This solution advises to the first player to lead the card which has different suit than the card which the second player would use as an "answer" to a move of the first one. This would allow the first player to find the suit of cards which has the second player according to the description of play 1. The rest of the proof that this advice is reasonable is trivial. We can see that the solution of the problem in cards is clever but useless unless the first player finds the value of the function $h(b, play\ 1^{0}(b,s))$ for some constant b, revealing the suit of the second player by some illegal way.

The last example calls attention to the form of axioms which describe the world and its changes. The cause of the deduction of useless solution was the axiom describing $play\ 1^{0}(x,s)$, which allows to deduce something about the former situation from a new one ($(\exists y)(INHAND(2,y,s) \;\&\; \ldots)$). We must look for restrictions which would make the problem solving in predicate calculus safe against such unapplicable solutions.

4. USEFUL THEORIES OF SITUATION CALCULUS

Let us define by induction an ordering of situation terms of the situation calculus with object functions dependent at most on one situation argument and every situation function just on one situation argument and possibly several object arguments. Intuitively the ordering means that some situation described by the situation term t_1 "occurred earlier" then that described by t_2 ($t_1 \prec t_2$).

Definition: The only $*$-subterm of a variable s is s and of the constant s_0 is s_0. Let t be a situation term of the form $\varphi(\bar{a},t_1)$, where φ is a situation function of the theory under consideration, $\bar{a}$ is a sequence of object terms of the appropriate length, t_1 a situation term. A situation term t' is $*$-subterm of t iff it is either $\varphi(\bar{a},t_1)$, or is $*$-subterm of t'.

Definition: Let t be a situation term. Any object constant or variable is regular term with respect to t (RT(t)). Let f be an object function with object arguments only, then the term $f(a_1, \ldots, a_m)$ is RT(t) iff each a_i, $i \leq n$ is RT(t). Let h be an object function with one situation argument or a situation function, then $h(b_1, \ldots, b_m, t_1)$ is RT(t) iff each object term b_i, $i \leq m$ is RT(t) and t_1 is $*$-subterm of t and t_1 is RT(t_1). A situation term is regular iff it is regular with respect to itself.

Theorem

Let K be an open theory of situation calculus with axioms of two types only:

a) core axioms $(\forall s)\, A[\![s]\!]$, where A is formula uniform w.r.s, in which all maximally occuring terms are regular RT(s)

b) transition axioms $(\forall s)(A_1[\![s]\!] \rightarrow A_2[\![\varphi(\bar{x},s)]\!]$, where φ is a situation function, A_1, A_2 are open formulas uniform w.r.s, $\varphi(\bar{x},s)$ in which all maximally occuring terms are RT(s), RT($\varphi(x,s)$), resp., and $\bar{x}$ is a sequence of all free object variables of the formula A_1 & A_2

Let the problem in K be specified by the non-open formula

$$X[\![s_0]\!] \rightarrow (\exists s)\, Y[\![s]\!] ,$$

which can be transformed into a prenex normal form so, that its variant does not include any $\forall$ quantifier and each object term maximally occurring in $X[\![s_0]\!]$, $Y[\![s]\!]$ is regular term with respect to s_0, s, resp. This formula is provable in K iff there is a finite set $\mathfrak{T}$ of regular variable-free situation terms of K such that

$$K \vdash X[\![s_0]\!] \rightarrow \bigvee_{t \in \mathfrak{T}} Y[\![t]\!] .$$

This theorem deserves a very careful interpretation. In our former paper /6/ we have shown how a finite set of non-open regular terms can be interpreted as a program for

robot's behaviour, provided the robot has an explication ability for a given theory (he is able to detect the actual value of all functions at a given instant).

An obvious but suppressed condition for the feasibility of this program was the robot's full information about the actual state of the world. The specified set of situation terms can be interpreted as a program for robot's behaviour even when the robot knows much less about his environment. He must be able to decide which conjunction A_i of conditions of transition axioms is true in the current world if disjunction

$$\bigvee_{i \leq m} A_i[[t]]$$

can be proved about the corresponding state t (represented by a regular non-open term). Let use call this a decision ability.

5. CONCLUSIONS

Obviously a robot, which is supposed to have full information about the facts in the current world, can easily verify whether a formula is true in his current environment using this description as a model. He has then the decision ability. Provided that he has also the explication property, he can efficiently use the situation calculus for his planning, even if he is not the sole agent in the world, (see the cat and mouse example).

On the other hand let us consider the robot which is the sole agent in the world, but has not a complete knowledge of the facts holding in the environment. Let us suppose, that the robot has an ability to find out the facts about the environment which appear in conditions of transitions, when he needs them. Let this be done in such a way that the inquuiry does not influence the facts the robot had proved or achieved up to this instant, e.g. a careful exploration of the robot's world, as it is performed by the executing robot of Siklossy. Suppose the robot ends the exploration in his primary position, then this activity is described in the situation calculus by the axiom $(\forall s)(\forall x)(\exists y)\ IN(x,y,s)$ after skolemization $IN(x,loc(x,s),s)$.

The robot's explication ability for the theory states: the location of any object can be found. In this case it implies also decision ability due to the fact that the robot causes no change except his acquisition of some new facts during the exploration. Moreover the world does not respond by some predicted or unpredicted change. Thus the behaviour of the executing robot of Siklossy can be planned in an adequate calculus with situation dependent object functions, because this robot has both explication and decision abilities.

It would be very interesting to specify lucid necessary conditions, under which the planning of a robot in uncertain environment can use the situation calculus (e.g. what problems could solve the cat if he is not able to decide whether some door is open or closed at the given moment - and can only deduce some facts from the former positions of the mouse).

References

J.Mc Carthy and P.J.Hayes /69/, Some philosophical problems from the standpoint of artificial intelligence, Machine Intelligence 4 (1969), 463-505

G.G.Hendrix /73/, Modelling simultaneous actions and continuous processes, Artificial Intelligence 4 (1973), 145-180

C.C.Green /69/, Theorem proving by resolution as a basis for question answering systems, Machine Intelligence 4 (1969), 183-205

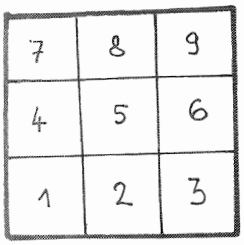
L.Siklóssy and I.Dreussi /73/, Simulation of executing robots in uncertain environments, Technical Report 16, May 1973, Dept. of Computer Science, Univ. of Texas.

O.Štěpánková /78/, Robot planning and the situation calculus (in czech), Technical Report of Institute for Computation Techniques (1978)

O.Štěpánková, I.M.Havel /76a/, A logical theory of robot problem solving, Artificial Intelligence 7 (1976),129-161

O.Štěpánková, I.M.Havel /76b/, Incidental and state dependent phenomena in robot problem solving, Proceedings of AISB Summer Conference, Univ.of Edinburgh (1976), 266-278

R.M.Young /76/, Design choices for a world modelling system, Proceedings of AISB Summer Conference, Univ. of Edinburgh, 1976, 376-386



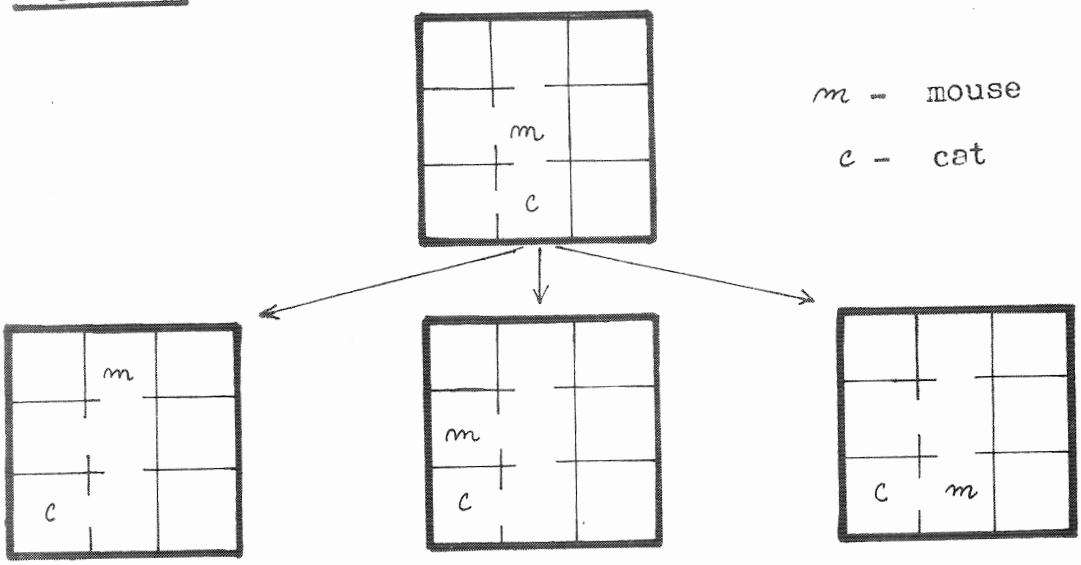
Legend :

the door is closed

the door is open

the position unknown

Figure 1



m - mouse

c - cat

Figure 2

initial state :

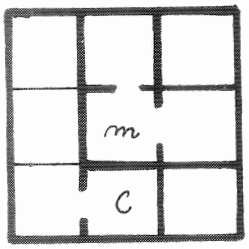
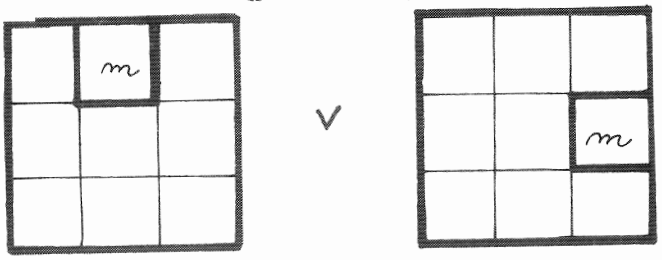


Figure 3

goal state :



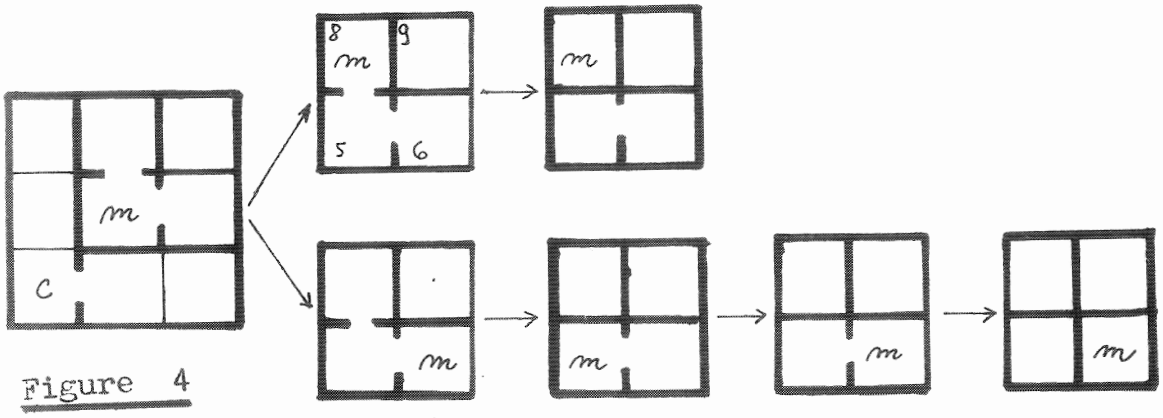


Figure 4

Completeness and Soundness of the Connection Graph Proof Procedure

W. Stephan, J. Siekmann

Institut für Informatik I, Universität Karlsruhe,

D-7500 Karlsruhe 1, Postfach 6380, Germany

Introduction

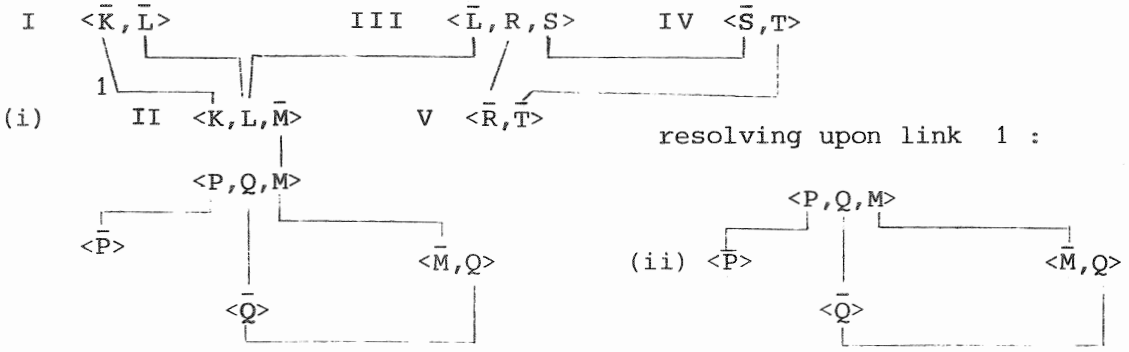
For over ten years of research in automatic theorem proving the axioms and the negated theorem (the input clauses) given to a theorem prover (ATP) have been considered as a *set* of clauses, that is, they have been treated with as little internal structure as possible. The traditional *refinement* is a predicate ranging over clauses or literals and is used in order to cut down the size of the search space and almost all effort in traditional research on automatic theorem proving was spent on finding such predicates.
This view is convenient but strange, since there are many possible ways to structure the input clauses and this structure can have a dramatic effect on the behaviour of the ATP:even the simplest such structure, the order in which the clauses are written down, has such an effect.

Only recently several workers independently considered the initial input clauses not as a set, but as some sort of graph structure - the graph itself as well as the procedure operating on it being defined in various ways: [3], [9], [10] and [11].
From all the graphtheoretic approaches, Kowalski's connection graph proof procedure [3] appears to have the strongest impact, theoretically as well as in practical terms, on automatic theorem provers.
In *practical terms* there are remarkable efficiency improvements over existing proof procedures:

- The resolutionstep for a particular pair of complementary literals is done once at most. (Since the link is deleted after the resolutionstep.)
- The unsuccessful search for potentially unifiable literals is eliminated, and
- a "cancellation" process similar to Waltz's [4] scene analysis program (known to workers in the field of Artificial Intelligence as "jigsaw phenomena") leads to a strange and often enormous reduction in the search space.

Further advantages are "the maximal confusion it makes between search space and search strategy" ([3] p. 589) and the ease by which particular known search strategies can be simulated.
The cancellation process is also the main reason for the strong *logical impact* of this proof procedure and it partially explains, why some of the theoretical problems (completeness, soundness and others) have been open now several

years [i]. To demonstrate that completeness and soundness of the connection graph proof procedure are not at all obvious, consider for instance:



Resolving upon link 1 leads to a tautology, which is deleted, hence I and II are deleted since $K,\bar{K}$ is now pure. Similarly clauses III, IV and V are deleted; i.e. after one step the whole connectiongraph (i) 'shrinks' to (ii) and it is in general not obvious whether the reduced graph still consists of an unsatisfiable set of clauses iff the initial one was (soundness). It is also conceivable, that some of the deleted links may have been necessary for a derivation (completeness).

Soundness at the Ground Level

In the following let S be an unsatisfiable set of ground clauses and let <S> be the connection graph obtained from S.

The proof proceeds in two steps: first we show the soundness for a special derivation (the α-derivation) and the general result then follows from a Church-Rosser-Property.

Def. 1.1: An *α-derivationstep*

$$\langle S_1 \rangle \vdash_{\alpha} \langle S_2 \rangle$$

is defined as:

(i) select arbitrarily some literal L_o in $\langle S_1 \rangle$

(ii) for all links in $M(L_o)$ form connection graph derivations, where $M(L_o)$ is the set of all links connecting positive and negative occurrences of L_o.

An α-derivation is a formalization of Colmerauer's cancellation system (7) in [3], and is a generalization of the one-literal-rule of Davis and Putnam [7].
Note that every occurence of L_o and every occurence of $\bar{L}_o$ $\langle S_1 \rangle$ is pure after $\vdash_{\alpha}$; i.e. $\langle S_2 \rangle$ does not contain any literal L_o nor its negation.

theorem 1.1: An α-derivation is sound; i.e. let:

$$\langle\langle S_1 \rangle\rangle \vdash^{*}_{\alpha} \langle S_n \rangle$$

then S_1 is unsatisfiable iff S_n is.

The proof of the soundness and completeness for α-derivations is not too difficult using standard (model theoretic) proof techniques of automatic theorem proving. The difficulty in establishing the general result is precisely that the usual standard techniques appear to fail. Intuitively the difference between α-derivation and more general derivations is that in an α-derivation no unifiable literals exist which are not connected by a link.
In order to prove the soundness for an arbitrary selection strategy we therefor prove a "Church-Rosser-Property" for connection graphs of which the soundness is then an immediate consequence.

theorem 1.2.: (Church-Rosser-Property.) Let:

(i) $\langle S_o \rangle \vdash^* \langle S_n \rangle$ be some derivation

(ii) $\langle S_o \rangle \vdash^*_\alpha \langle S_m \rangle$ be some α-derivation

Then exist derivations with

(iii) $\langle S_n \rangle \vdash^* \langle S_k \rangle$ and

(iv) $\langle S_m \rangle \vdash^* \langle S_k \rangle$

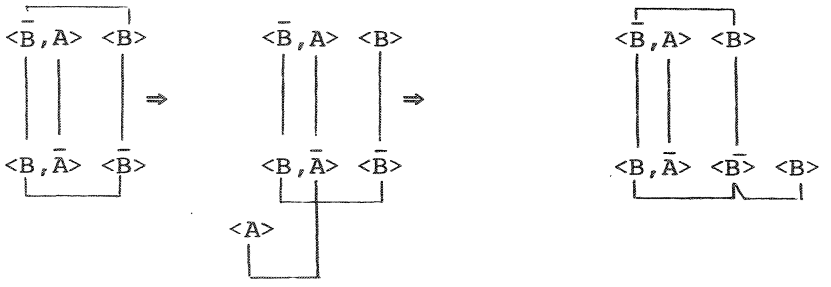
theorem 1.3.: (Soundness.) Let S_o be an unsatisfiable set of ground clauses; then for any derivation

$$\langle\langle S_o \rangle\rangle \vdash^* \langle S_n \rangle \quad ,$$

S_n is an unsatisfiable set of ground clauses.

Completeness at the ground level

The following example shows that an arbitrary selection strategy will in general not be complete, since some essential links may never be selected:



(i) After we circulated a first draft of this paper, F. Brown, University of Edinburgh, sent us his personal notes[1], in which he also proves the completeness of various C-graph strategies. His notes were completed prior to ours. We also acknowledge a helpful discussion on the notion of 'covering' ('exhaustiveness' in his notes) with F. Brown.

The most immediate criterion for a selection strategy to be complete would be to ask that every link is "covered" by that strategy, i.e. that every link is eventually resolved upon. Unfortunately no selection strategy exists that could fulfill this requirement. Moreover one would like a condition as weak as possible, shared by as many strategies as possible. For that reason we define covering as follows:

Def. 1.9: A selection strategy β is *covering* if for every link l in $\langle S_i \rangle$, $\langle\langle S_1 \rangle\rangle \vdash_\beta^* \langle S_i \rangle$, $i \in \mathbb{N}$; there exists a number m_l, such that after m_l derivation steps

$$\langle\langle S_1 \rangle\rangle \vdash_\beta^* \langle S_i \rangle \vdash_\beta \langle S_{i+1} \rangle \vdash_\beta \ldots \vdash_\beta \langle S_{i+m_l} \rangle$$

link l does not exist in $\langle S_{i+m_l} \rangle$.

In [8] we prove the completeness for derivation strategies, which are covering in the above sense. We also exhibit an example to demonstrate that the connection graph proof procedure is not a decision procedure at the ground level even for covering strategies.

The General Level

The completeness at the general level is established by "lifting" the completeness result of the ground level. Once the basic 'trick' is discovered, the lifting itself is more or less technical and does not provide much insight into the mechanism of the connection graph proof procedure.
The 'trick' in the lifting results rests essentially upon an obvious factorization of the Herbrand universe. In particular if S is just one clause C, we will write P(C) for the set of ground clauses obtained from C; i.e. a set of ground clauses can naturally be regarded as a set of equivalence classes; C^g is a ground clause:

Def. 2.2.:

For C_1^g, $C_2^g \in P(S)$ and $C \in S$:

$$C_1^g \sim C_2^g$$

iff $C_1^g \in P(C)$ and $C_2 \in P(C)$.

That is, essentially we are considering the factored set $P(S)_\sim$ induced by $\sim$. With these definitions we define a mapping from the factored ground graph $\langle Sg \rangle$ onto the graph at the general level $\langle S \rangle$: if "$\leqq$" (the predicate induced by that mapping) holds, then there is a corresponding clause at the general level for each ground clause and every link at the ground level corresponds to some link at the general level.

Literature:

[1] F. Brown: 'Notes on Chains and Connection Graphs', 1976, personal notes, Department of Computation and Logic, Edinburgh University

[3] R. Kowalski: 'A proof procedure using Connection Graphs', JACM, vol 22, no. 4, 1975

[4] D. Waltz: 'Generating Semantic Descriptions from Drawings of Scenes with Shadows, Ph. D., MIT, 1972

[7] Davis, Putnam: 'A comp. proc. for quant. theory', MACM, vol 2, no. 2.

[8] W. Stephan, J. Siekmann: 'Completeness and Soundness of the Connection Graph Proof Procedure, Universität Karlsruhe, Interner Bericht 7/76

[9] R.E. Shostak: 'Refutation Graphs', J. of Art. Intelligence, vol 7, No. 1, 1976

[10] S. Sickel: 'Clause Interconnectivity Graphs, IEEE Trans. on Comp.', vol c-25, No. 8, 1976

[11] Yates, Raphael, Hart: 'Resolution Graphs', J. of Artificial Intelligence, vol 1, 1970

COMMON-SENSE REPRESENTATION

P. L. Suzman
Department of Artificial Intelligence
University of Edinburgh

Abstract

Some problems with existing formal representation languages are discussed, and an alternative semantics is proposed that attempts to build in a particular view of the world. This is essentially that the world consists of objects whose properties are determined by the properties of their parts. The semantics also makes provision for predicates to have intensions as well as extensions, so they can be reasoned about. The implications of the semantics are discussed with particular reference to the problems of representing change, where it is contrasted to the approach taken in STRIPS and the Situational Calculus.

Descriptive Terms

Representational Language, Semantics, Representation of Change.

Introduction

In this paper we shall be concerned with the modelling and representation of non-mathematical domains such as are typically found in planning, problem-solving and game-playing.

One basic approach to this task - the descriptive or axiomatic approach - is to model the domain by making true statements about it in some formal language (typically First Order Predicate Calculus - FOPC). We can think of the domain as being reflected in or modelled by a model (in the Tarskian sense) of the sentences.

While the descriptive approach has many desirable features (e.g. see Hayes, 1977), it has proved embarassingly difficult to effectively use it in even toy problems - one gets bedevilled by monkeys standing on their own shoulders trying to reach bananas that one cannot show are still in existence.

In this paper we suggest that it is not the descriptive approach per se that is responsible for this failure, but rather deficiencies in existing descriptive languages and the way they are used.

World-Views

A central thesis of this paper is that any descriptive language has some "world-view" - i.e. it has built into its semantics certain viewpoints about the domains it models; and that deficiencies in this world-view have serious consequences, not only in axiomatising domains, but also in proving theorems in those domains.

What then is FOPC's world-view? An FOPC model consists of a set of individuals together with some relations and functions on these individuals to interpret the relation and function symbols of the language. Thus FOPC embodies no commitment to any particular species of individual - it essentially cares only about the relations on them. As far as these relations go, their own properties (e.g. transitivity) and the interrelationships between different relations can be captured only by means of their extensions . Thus the relations themselves are not directly available for discussion.

Some consequences of this semantics can best be discussed in the context of a simple example. The most natural way of using FOPC to

model some simple world is to think of the individuals in our model as corresponding to things in the world, and the properties and relations in our model as corresponding to properties and relations in our world. So to capture that some object is red, one typically asserts Red(b), where the denotation of our constant "b" will represent our object. This style is, however,too restrictive of the sort of statements that one can make - e.g. it is hard to assert that an object is single-coloured, or to ask what colour it is (as opposed to whether it is red).

One reaction to this problem is to move to a style in which properties in the world become individuals in our model as well - e.g. Colour(b,red), or the even more extreme simulation of second order calculus, Holds(b,red). But this enlarged ontology brings with it its own problems - not least the reduction in the power of standard proof techniques that accompanies it (e.g. in resolution, steps that previously could be dealt with syntactically by the unification algorithm get replaced by problems of deciding whether two terms are equal).

The Search for a More Appropriate Semantics

Is there then some other semantics that, while perhaps not as general as the standard semantics, will serve us better in our modelling tasks? It is to be emphasised that we are not looking for some syntactic variant of FOPC (like semantic nets - see Woods, 1975), but an alternative semantics.

Let us examine more closely the worlds which we seek to model, and the sort of statements we wish to make about them. We shall attempt to refine the view that the world consists of objects having properties. Notice that these objects are not necessarily "atomic" (in fact are usually not) - they have parts which in turn can be considered as objects having their own properties.

By way of example, assume we have a cup and saucer whose combined weight we wish to assert as 100g. A statement such as "(cup weighs 50g & saucer weighs 50g) or (cup weighs 49g & saucer weighs 51g) or ... or (cup weighs 100g & saucer weighs 0g)" while logically correct, (assuming that things can have only integral weights and we have some assertions forcing the disjunction to be exclusive rather than inclusive) is clearly unsatisfactory. It has been made at the wrong "level" - we should have been talking about the whole rather than the parts.

This notion of the level at which we talk is not restricted to the objects alone - it also applies to the properties which we are specifying. Many natural language adjectives, for example, can be seen to be fundamentally disjunctive in nature - e.g. when we assert that our table is red we are really saying that it is one of a whole range of shades (scarlet, puce ..) that we have clustered together and called "red".

"Red" is a simple example - in general properties are structured in much more complex ways. Consider say the ways in which we can specify a person's position - "in New York", "in Scotland" and "in Edinburgh" are clearly part of a complex structure of sometimes mutually exclusive positions.

It is thus desirable that our ontology of both objects and predicates be rich enough to allow us to describe things at an appropriate level, and where necessary change levels.

Bearing this in mind, we shall now make a fundamental assumption about the nature of individuals in the world, namely that each

individual is completely characterised by its parts - two individuals are the same just when they have the same parts. This far reaching assumption is borrowed from the Nominalist school of philosophy (Goodman, 1966), and indeed the semantics we shall propose has its roots in a simple semantics for a nominalist language (the Calculus of Bundles) proposed by Eberle (1970).

For the purposes of this paper we shall make the additional (somewhat dubious) assumption that every individual has ultimate parts which we shall call atoms. (It would seem possible to achieve the same effects, at the expense of somewhat greater complexity, if we drop this assumption). While these atoms have no parts, they do enjoy certain elementary properties, which we shall refer to as particles.The atoms are completely characterised by these particles - indeed are to be thought of as made up of them just as compound individuals in their turn are made up of atoms. Particles will intuitively correspond to such things as particular colours, positions, times and so on.

We can express these suggestions in set theoretic terms, but it should be emphasised that in reality we need only a very limited form of set theory. (Our semantics is thus much less complex than the Second Order Predicate Calculus with its attendant difficulties.) We start with a (typically finite) set of particles, P; atoms are simply sets of particles and our compound objects simply sets of atoms. Thus if we have a finite set of particles, we will have a finite (but much larger) set of possible compounds.

How are our predicates to be reflected in our semantics? The basic idea here is that the predicates will also denote - i.e. they will have intensions. The simplest predicates will be those whose intensional denotation is some particular particle - some particular shade of red say, e.g. scarlet. To assert of some atomic object "b" that it is scarlet, one asserts "scarlet is an element of b ".

Now if we arrange for "red" to denote the atom whose elements are all the particles corresponding to different shades of red (scarlet, blood-red, etc.), then we can assert that b is red by asserting "red overlaps b", where overlaps means having non-empty intersection. Thus the fact that being scarlet is a particular case of being red is reflected directly in the fact that "scarlet is an element of red".

Thus for atomic objects, predication is often reflected as overlapping - this in essence is all Eberle's semantics permitted, with no more complex objects, properties and predication allowed.

When we are dealing with compound objects a more complex notion than overlapping is needed. When in English we say an object is red our meaning is somewhat vague - possible interpretations stretch from simply requiring that some part of it is red to requiring that all parts of it be red. Both are among the notions of predication we shall need, the first being an existential notion (some of ... is ...), the second a universal one (all of ... is...).

There is no need to restrict our predicates to being atomic objects; we can also have them represented by compounds or even sets of compounds. The latter with predication being taken as elementhood are of course simply standard extensions. Interesting notions of predication involving these higher level properties are duals of those mentioned before i.e. (... is some of...) and (...is all of...), and also, crucial for dealing with mutually exclusive properties, (...is one of...).

It should be pointed out that we have used the word "intension" rather loosely in view of the different notions of predication we have

used. Properly speaking we should think of the intension as both the denotation of the property and the particular type of predication involved.

In the rest of the paper we shall explore the consequences of this choice of semantics on the problem of the represention of change.

The Representation of Change

It is perhaps in the representation of change that a representation language faces its greatest challenge. Here issues that look to be of only philosophical interest in the static case turn out to have important practical consequences.

Two of the major approaches in A.I. to the representation of change are that of the situational calculus (McCarthy & Hayes, 1969) and that embodied in STRIPS (Fikes & Nilsson, 1971). Both have proved to have serious deficiencies which bear closer analysis. The fundamental problem faced (and as yet unsurmounted) by the situational calculus is that of effectively relating two different situations. The basic problem - essentially the well known frame problem - is in describing small changes in the world without the aid of what Waldinger (1978) has called the "STRIPS assumption" - that things not explicitly referred to remain unchanged. Hayes (1971) attempted to embody such an assumption by using a causality notion - the idea of "connectedness" - the idea that all the facts that change can be thought of as being connected to the action that caused the changes; and so we can prove them to be unchanged whenever we can prove them to be unconnected to the action.

Hayes's proposal, while an advance, is not without its problems - not least its alarming complexity . Other problems include not being able to specify that just one property of some object has changed, and some tacit assumptions it makes about the sort of properties objects can enjoy.

The other basic approach, that of STRIPS, is essentially syntactic; this has not, however, prevented it from proving an effective practical tool. For a good discussion of some of its problems, see Waldinger (1978). The basic weakness that we shall focus on here is its inability to effectively cope with more than one level of description - a problem that only really emerges when it is used to model complex domains.

Consequently STRIPS models have the tendency to capture essentially static relations between properties by how they interact dynamically. A good example of this is the blocks world concept "clear" or "cleartop". A natural definition of this concept is that a block is clear just when there is no other block on it. This relationship is typically captured "dynamically" in STRIPS block worlds - i.e. the action specifications take care to ensure that a new state never arises with a block that is both clear and has another block on it; and this, together with initial situations that never present such a state of affairs, to some extent capture the concept.

One effect of this style is to make the task of problem solvers using the model much harder. Consider the plight of a problem solver presented with the subgoal ON(a,b) & CLEAR(b) - how is this to be rejected as impossible without trying all ways of reaching it from the initial state?

There is thus an interesting duality - in the situational calculus it is easy to capture relationships between properties, but hard to update them when some change; while STRIPS allows us to update them easily, at the expense of hiding relationships between

them.

What of a language using our proposed semantics? The basic idea here is to regard a situation as simply a compound object (like any other) consisting of all the objects present at some time. This intuitive view of a situation contrasts with that in the standard semantics, where it is by no means obvious to just what thing in the world (if any) the denotation of a situation constant corresponds. The close relationship that our approach has with the standard "possible worlds" semantics of modal logic (Scott, 1970) is clear; the situations correspond to possible worlds.

This view of a situation allows one to decompose the situation into parts, and to relate two similiar situations by asserting that parts of them are equal. If two situations S0 and S1 differ only in that some individual b is different,we can simply assert that S0\b=S1\b. (Where \ is set subtraction). Notice that in the standard situational calculus the same problem would require assertions for each predicate.

We can now distinguish between two sorts of side-effects of an action - the "genuine" and the "logical" side-effects. A genuine side-effect occurs when, as a result of being somehow physically connected to the individuals directly affected by the action, another individual also becomes affected (e.g. a block is moved causing another block resting on it also to move). A logical side-effect occurs in that some compound object changes by virtue of some part of it changing (e.g. a block is moved causing the distance between it and some other block to change). Actions thus only have to specify their genuine side-effects, for the logical-effects will be independently computable.

The more careful we have been in our choice of basic properties (our particles), the fewer will be the genuine side effects of our actions. If we take "clear" as a particle, for instance, then we shall have to note that moving blocks can have the side-effect of changing the "clearness" of other blocks, in addition of course to bumping into them. It may indeed be possible to restrict genuine side effects to instances of real world physical causality.

Note that in many problems the size and complexity of a situation is small. For instance in chess a situation is basically the pieces with their positions - and so it is easy to specify moves as all the complexity arises in the logical side-effects of the moves. In other words we describe what "things" change, not what facts change - we have, like KRL (Bobrow & Winograd, 1977), an object rather than fact centred world-view.

A full treatment of change must cope with many issues such as causality, time and trans-world identity which cannot here be adequately discussed. It should be noted, however, that in regard to the problem of trans-world identity, we have much more flexibility than other systems, and can model worlds where individuals can disappear, merge or otherwise misbehave. We also have the ability to discuss the "trajectories" of elements through time, for these too are compound objects in our ontology, orthogonal so to speak to the situations.

Conclusions and Work in Progress

We have suggested that some problems encountered with representation languages can best be overcome by embodying characteristics of the worlds we are modelling directly in the semantics of the language. One possibility for such a semantics was

suggested, its fundamental feature being its strongly reductionist world-view - i.e. its commitment to the view that complex properties of a complex object can be reduced to statements about simpler properties of simpler parts of the object. Some advantages that flow from the use of a language having such a semantics, particularly in the modelling of change, were suggested.

The acid test of such a language is its practical utility in modelling and problem solving activity. Preliminary results indicate that the language has powerful semantically based simplification rules making use of the idea that each assertion in the language is "about" some individual. The rules try to replace assertions that are about the same or overlapping individuals by equivalent assertions about separate individuals. We are currently modelling the game of noughts-and-crosses, and here it seems that the model is both intuitive and simple, and that because of our rich ontology, common sense theorems about the game do indeed have short and intuitive (hand generated) proofs.

Acknowledgements

I would like to particularly thank my supervisor, Dr. Gordon Plotkin, for his help, as well as Dr. Pat Hayes and Prof. John McCarthy for some useful discussions.

Note

Some of the material in this paper was originally presented at the Workshop on Automatic Deduction at M.I.T. in 1977.

Bibliography

Bobrow,D. & Winograd,T. (1977) An overview of KRL, a knowledge representation language. Cog. Sc. 1 no. 1.

Eberle,R. (1970) Nominalistic Systems. Dordrecht-Holland.

Fikes,R. & Nilsson,N. (1971) "STRIPS: a new approach to the application of Theorem Proving to Problem Solving" Artificial Intelligence 2.

Goodman,N. (1966) The Structure of Appearance (2nd ed.) Indianapolis.

Hayes,P. (1971) A logic of actions. Machine Intelligence 6 (eds Meltzer,B. & Michie,D.) Edinburgh University Press.

Hayes,P. (1977) In Defence of Logic. Proc 5th IJCAI, Cambridge, Mass.

McCarthy,J & Hayes,P. (1969) Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4 (eds Meltzer,B. & Michie,D.) Edinburgh University Press.

Scott,D. (1970) Advice on Modal Logic. In Philosophical Problems in Logic (ed Lambert,K.) D.Reidel.

Waldinger,R. (1978) Achieving Several Goals Simultaneously Machine Intelligence 8 (ed Elcock,E & Michie,D) Ellis Horwood.

Woods,W. (1975) Whats in a Link. In Representation and Understanding (ed Bobrow & Collins) Academic Press.

TOWARD GREATER GENERALITY IN ARTIFICIAL INTELLIGENCE

Leonard Uhr
University of Wisconsin
U.S.A.

Manfred Kochen
University of Michigan
U.S.A.

ABSTRACT

We have been asked to discuss how research in Artificial Intelligence and Cognitive Modelling might best progress, and to comment on this meeting's papers. The comments must wait for the oral presentation. Here we examine two major issues: A) We describe an approach toward more general AI systems, and pose some rather simple-seeming yet unsolved touchstone problems to be attacked. B) We suggest that AI should attack our very difficult problems with as diverse as possible a set of good approaches.

INTRODUCTION

Cognitive scientists, whether in Psychology or Artificial Intelligence, have for a long time divided cognition into the major sub-functions of perception, problem-solving, decision-making, remembering, language handling, and learning. Each such sub-function is itself so complex that further simplifications are usually found to be necessary, e.g. to perception of gradients, line drawings, chromosomes, or furniture; or problem-solving in puzzles, checkers, chess or logic.

Computers and programming tools have become so powerful that an important problem of practical or scientific value can sometimes be attacked directly, by coding a program that embodies the specific knowledge needed. Thus we are beginning to achieve systems each of which successfully handle its particular problem domain - e.g. recognition of spoken sentences, diagnosis of allergies, a strong game of chess, description of real-world scenes, "intelligent" robots.

But with this major recent thrust AI research has tended to ignore some of our most basic problems: How can the separate cognitive functions be put back together? How can one system (e.g. to play checkers, recognize spoken speech, or describe a room with furniture) be extended and generalized to handle other problems (e.g. play chess, answer questions, describe outdoor scenes with houses, vehicles and hills)? How can a system with realistically large amounts of knowledge about a wide variety of domains of discourse bring the appropriate knowledge to bear in each particular situation? These kinds of problems force us to search for general, rather than powerful, results.

TOUCHSTONE PROBLEMS THAT FORCE SYSTEMS TO BE GENERAL

We now pose several seemingly simple problems that would force programs to begin to combine the usually separated cognitive processes, in relatively general and well-integrated ways. They may appear simple and far more mundane than some of the things that our "expert" (domain-specific) AI systems can do. But we know of no AI systems that can handle them. Yet they are prototypical of the problems that constantly confront any intelligent human being. Essentially, they combine linguistic and perceptual inputs in a common sensory input channel; need memory, problem-solving, perception and semantic analysis; and force the system to make non-deterministic decisions as to appropriate types of actions, where a wide variety of different unanticipated kinds of knowledge and actions might be appropriate.

Consider the following extremely common example: A person crossing a street in traffic hearing a yell that sounds like "Lookout!" will deal quite readily with both the perceptual and the linguistic inputs however thoroughly they are jumbled together. Red lights, crosswalk signs, billboards, etc. will all be noticed and understood in terms of the total street scene, as part of a single well-integrated perceptual achievement. There can be no separate linguistic channel, like the keyboard inputs to robot systems. Nor can linguistic and perceptual information be handled in completely different ways, using entirely separate programs.

To make a beginning, we will simplify drastically:
An "Environment" will be a set of "Happenings". Each "Happening" is a sequence of "Scenes", where each "Scene" is a set of related "Compounds". A "Compound" (which might be an "Object" or a "Word") is a set of related "Primitives" or Compounds.

The relations over Compounds that combine them into a Scene, and the relations over Primitives and Compounds that combine them into Compounds, might either structure them into a graph, order them in two or one spatial dimension (like a photograph or a sentence), or simply collect them, unordered. Thus:

$\text{Environment}_E = \text{Happening}_1, \text{Happening}_2, \ldots. \text{Happening}_H$
$\text{Happening}_H = \text{Scene}_1, \text{Scene}_2 \ldots. \text{Scene}_S$
$\text{Scene}_S = R_1\ (\text{Compound}_{11}, \ldots), R_2\ (\text{Compound}_{21}, \ldots), \ldots R_C\ (\text{Compound}_{C1}, \ldots).$
$\text{Compound}_C = R_1\ (\text{Thing}_{11}, \ldots), R_2\ (\text{Thing}_{21}, \ldots), \ldots R_T\ (\text{Thing}_{T1}, \ldots).$

In addition, the following kinds of compounds will be distinguished:
"Irrelevancies" (e.g. background, noise, or Compounds unrelated to the cognitive system's tasks).
"Borders" (Primitives or Compounds that have been specially chosen to bound or separate Compounds).
"Classes" (different "Variant" instances of the same Object or Word).

We can now start with a very simple set of specifications from the above: Each Happening contains only 1 Scene. Each Scene contains only from 1 to 3 Words and from 1 to 3 Objects, linearly ordered. Each Compound (Word or Object) contains from 1 to 5 Primitives, some ordered, some unordered, and bordered by parentheses. In addition, Irrelevant Primitives (background, noise, etc.) should be scattered about, some at random, others in systematic (but to the program unknown) ways. Roughly 10 to 100 Objects, Words and Primitives should be used. Each Word is a class with roughly 5 possible Variants. Each Object is a class with roughly 100 possible Variants. Each Primitive is a class with roughly 3 possible Variants. (The Irrelevancies and Variants help make the space of possible Scene-strings too large to handle by simply coding a program that stores them all.)

A Few Examples of Touchstone Problems

In what follows "*" denotes strings of Irrelevant background symbols. Suggestive words or pictures replace primitives.

1. .* NAME * ANIMAL * (dog)*(doghouse)*., or equivalently,
.* NAME * ANIMAL *((wagging tail), (barking), (loyalty))*
((house),(small),(one opening))*.

Words are in capital letters. The first word requests the system to act or respond by producing a name. The second word refers to one of the two objects. An appropriate response might be "Spottie", or "Dog").

2. .* WRITE YOUR * NAME * (_____) *.

3. .* POINT * (triangle) * INSIDE * SQUARE * ( * )*.

In this example some objects are symbolized graphically.

4. .* FIND * () * WITHIN * () *.

5. .* IN * () * FIND * () *.

6. .* TRANSFORM * () * INTO * () * INTO * ( ) * RELATE *.

There are many appropriate responses, of differing qualities.

7. .* NAME * ANIMAL *(sheep)*(chair)*.

8. .NAME * PET *(sheep)*(dog)*.

9. .* PET * PET *(sheep)*(dog)*.

Several different responses, and "mistakes", might be appropriate.

10. .* POINT * CIRCLE * SQUARE *( )*( )*.

Such systems should be tested in competitions of the following sort: Anyone who wishes can pose a set of specific problems, whose complexity can be assessed in terms of the number, size and structure of Scenes, Objects, Words, etc. as described above. With this set he can challenge programs for which this kind of generality is claimed.

The program without any modification should then be given the problem set. But that is possible only if the program is capable of learning the needed knowledge from example problems. So we might give the programmer a fixed amount of time and space to encode the needed knowledge.

Once some successes are achieved many less simple and more realistic problems can be attacked. For example, boundaries can be eliminated and compounding continued for several levels. Photos of real scenes can take the place of the simplified symbolic representations. More complex problem-solving and decision-making tasks can be posed.

This, then, poses a set of seemingly simple problems that, we feel, cut to the heart of our basic problem, that of getting a grasp on "intelligent" "cognition", and force us to do this in a general, generalizable way.

THE PARAMOUNT IMPORTANCE OF THE GREATEST POSSIBLE DIVERSITY IN APPROACHES

But now to make a far more important point: First to put it negatively - we could well be wrong. Of course we feel that such problems are worth pursuing, and we hope that others will agree. But more important, we think it is worth reminding people working on the very difficult problems of artificial intelligence that science progresses most rapidly when enthusiastically involved people do what they, each individually, decide is most valid. This means that a wide variety of problems should be pursued, from widely diverse approaches. Such diversity should be encouraged - of course always with the demand for worthwhile results.

Science is an advance into the unknown (by definition; otherwise it would be technology or history). To the extent that most of us explore

the same sub-continents much remains mysterious. But to the extent that a field is new, complex, strange and difficult the best strategy is to explore widely.

This argument is based upon the assumption that we may not yet know what are our basic problems, tools, evaluation techniques, or criteria for success. That is, AI is still a very young science. But there is nothing wrong with that. Nor need we justify our existence. We are simply trying to attack the central age-old problems of philosophy and psychology, and to extend our understanding of the computer and its potentialities. These are large problems. And if our progress is slow we certainly find ourselves in good company. We should not try to talk people into pursuing any particular approach, or into expecting some great practical results (and therefore become forever skeptical if these results never quite materialize).

One of the great joys of a vigorous science lies in its daringly free and open pursuit of knowledge. Artificial Intelligence is certainly a field where we should be valuing and encouraging diversity, and good, fresh, new ideas and approaches.

AI PROGRAMMING METHODOLOGY

Bob J. Wielinga

Laboratory of Psychology, University of Amsterdam
Weesperplein 8, Amsterdam, The Netherlands.

1. The basic issues.

This paper is intended to be a review of some ideas on AI programming methodology that have been around in recent years. Thus the basic question we will address is: what methods should we (c.q. can we) use for developing AI programs. However, first we have to establish what the goals of such a methodology would be. In trying to formulate these goals I will start with a reflection on the goals of AI programming and on the role that programs and programming play in the overall AI paradigm.

There seem to be at least two views in AI on the role of programming: 1) the "technology point of view", which puts an emphasis on the effectiveness of a program to perform tasks which require intelligence, and 2) the "cognitive science point of view", which sees programs as operationalisations of an information processing theory of human intelligence. The approaches overlap and thus, share a number of methodological issues; however their basic goals are different and hence the criteria on which programs are judged. A program that performs a task efficiently -maybe even elegantly- is not necessarily a good operational model of intelligent behaviour: it may be too task-specific, not addressing any fundamental problems.

If we adopt the cognitive science point of view and define the basic goal of AI to be the construction of a computational theory of intelligence, we may ask ourselves: "why program at all?". The answer to this question lies not so much in the basic goals of the discipline, but in the methodology of AI research. Lenat (1977) formulates a basic AI paradigm as follows:

(1) Given a cognitive activity (like playing chess, problem solving or understanding natural language),

(2) develop hypotheses and eventually a theory about what kinds of information processing could be taking place to produce such an ability.

(3) Incorporate that theory into a computer program.

(4) Experiment with the program, observe and evaluate its behaviour in order to determine the principles that produce the intelligent behaviour and the range of their applicability.

Although this paradigm is fairly typical for much AI research, there are considerable differences in the amount of effort put into each of the steps in the process of developing AI programs. The first step -selecting an area of cognitive activity- seems simple, but often a clear formulation of the problem that is being attacked is lacking. Burstall and Goguen (1977) have critisized a tendency in current AI methodology to write programs without clearly specifying what problem the program is expected

to solve. They emphasize the need for creating structured descriptions of theories which can serve as a basis of formal problem specifications. In a similar spirit several authors (Horn, 1977; Draper, 1978; Brady, 1978) have recently argued for a stronger emphasis on the development of AI theories. The general opinion presented in these papers is that one should study the problem one is trying to solve more thoroughly, try to specify its formal structure and the theory it is part of, rather than go ahead and "hack" a program together that solves the problem, but heavily relies on heuristics, clever programming and general problem solving techniques, and hence is difficult to understand, to evaluate and to improve. One of the most striking examples of how a theory of a problem domain can help to understand a program, is the explanation of the successes and failures of Guzman's program (Guzman, 1968) by the Huffman-Clowes-Waltz line-labelling (see Draper, (1978) for a more extensive discussion of this point).

One of the main problems, which this approach to AI -I will call it the "theoretical" approach- is facing, is that of representation. The formal specification of theories and problems requires a formalism to specify them in. Similarly, a theory about some aspect of intelligence has to be able to represent the knowledge and behaviours relevant to the domain in some formal way. This is a notoriously hard problem, which is to a certain extent domain dependent (eg. in vision) and as such of no direct concern of AI programming methodology, but which bears on the more general issue of the relation between knowledge representation and AI programming languages. One could argue that knowledge representation is an area of AI which should be distinguished from AI programming as such. Hayes (1974) makes a distinction between formalisms and languages on the basis of the argument that a formalism should have a proper semantics, i.e. it should be specified what an expression denotes in some world, while a programming language -which may have a proper semantics in the computational sense- allows the implementation of semantically new representational constructs. The rather pragmatic approach to knowledge representation in much AI research tends to ignore such a distinction for a good reason: no computationally adequate representational scheme has been available; one relies on representation languages which are pragmatically expedient rather than based on a proper semantics. The methodological question that arises in this context is to what extent one should let the pragmatics prevail. There seems to be a trade-off between sloppiness and expediency on the one hand and formal adequacy and computational inadequacy on the other hand. In section 4 we will further discuss this issue.

The problem with the theoretical approach, of course is that a proper theory may not be available. Marr (1976) distinguishes two types of theories in AI (and in science in general): 1) theories which are basically abstract accounts of how to solve a particular problem (Marr uses the term method) without reference to actual implementations of that method (algorithms), (Type 1 theories), and 2) algorithmic descriptions of a large number of processes whose interaction is its own simplest description (Type 2 theories). Marr argues that there may be no theory of Type 1 for intelligence. Certainly the current state of AI and of Cognitive Psychology does not allow for even the beginning of the formulation of a general Type 1 theory of intelligence. A consequence of this problem is that writing an AI program becomes a theory construction task: the program is the only way to express the theory. I believe that it is precisely this aspect of AI that gives much AI programming its essentially experimental nature: programs change and evolve continually in the process of developing a theory.

Although I acknowledge the need for the theoretical approach to AI

research in some problem areas, I believe that much AI work will in the near future continue to be of an experimental nature, will explore new problem domains and will point out the crucial problems a theory has to address. A consequence of putting more emphasis on the construction and study of programs than on theories, is the need for a rich variety of programming tools and a flexible programming environment. The problem with the experimental approach is however, how to assess an AI program as a theory of some aspect of intelligence. One obvious way to judge an AI program is to observe its behaviour and to try to figure out how it works if it works and why it fails if it fails, what may not be a simple task. Another (more common) way is to rely on the description of the program as given by its creator. This too may involve much critical reading and elimination of "wishful thinking" that may have crept into the description (or into the program itself). McDermott (1976) argues strongly against the "wishful mnemonics" trend: "we should avoid, for example, labelling any part of our programs as an "understander". It is the job of the text accompanying the program to examine carefully how much understanding is present, how it got there and what its limits are". The need for careful judgements of AI programs brings about an important goal for AI programming: to formulate in a precise way the principles on which the (presumed intelligent) behaviour of a program is based, what its limits are and how it could be extended to more general domains (if at all). This may be difficult. AI programs which lack a firm theoretical framework, often rely heavily on control: multi-processing, heterarchy, heuristic search. These aspects of AI programming languages are not very well understood; for example, no formal models for non-determinism are currently available. Consequently, an important goal of AI programming methodology is to develop tools which allow us to clearly express what we want to express in such a way that the principles which underly a program are still tractable.

Whether one takes the experimental or the theoretical approach, one problem remains: the complexity of AI programs. Winograd (1973) has emphasized the need for methods which enable us to cope with the complexity of large AI programs. This problem is not unique to AI, it occurs in other areas of Computer Science as well. Structured Programming and the development of new and better programming languages is one way to attack this problem. The recent work on programmer's assistants, program-understanding systems and automatic programming are alternative ways to master complexity.

Summarizing, I think that AI programming methodology should address the following basic questions:

(1) How can we implement our ideas, in such a way that our programs show sensible and flexible behaviours, while at the same time the concepts that the program embodies remain clear and understandable?

(2) How can we develop representational formalisms and languages, which are both formally adequate as well as rich enough so that we can use them to represent the things we need to represent?

(3) How do we manage the growing complexity of AI programs?

In section 2 we will discuss whether the Structured Programming methodology provides us with any answers to these questions. Section 3 will address programming systems and languages, while section 4 discusses the role of representation in AI programming.

2. AI and Structured Programming.

We must confront the question "how does Structured Programming fit into AI?". Clearly, the literature on Structured Programming has something to say about programing methodology, but it doesn't seem to have made so much impact on AI programming as it has in other areas of Computer Science. Many programming languages for AI (notably LISP) do not excell from a Structured Programming point of view; program correctness and program verification are often considered irrelevant issues for AI (many important AI programs hardly work at all) and it is often difficult to extract a clear conceptual structure underlying the code of large AI programs.

The reason for this difference in programming practice between AI -at least the experimental approach- and other disciplines of Computer Science, lies -as I see it- in different views on the programming activity itself, which stem from a difference in the nature of the problems that are to be solved. The Structured Programming methodology tries to make the conceptual structures of a program explicit by particular methods of problem analysis and program composition in order to achieve lucid programs which are easily understandable, correct and verifyable. Nobody in AI would question this as a legitimate goal, but the Structured Programming methods do not apply to AI programming in their full generality for two reasons.

Many AI problems are -contrary to most problems in other branches of Computer Science- not very well defined, the desired behaviour of the system cannot be described in simple terms like the description of an initial and a final state. The strict formulation of the problem is part of the AI effort. Notions like understanding a piece of text or perceiving an object, are only intuitively defined, and moreover their interpretation may depend on context, like the current environment and goals of the organism. As a consequence the idea of a correct program is not a very useful concept in AI. Obviously, simple bugs are a nuisance, but on a more abstract level I conjecture that deviations from expected behaviour will always occur in certain conditions, and moreover, that these deviations may be essential in the process of constructing and evaluating an AI theory. Since we do not have many other tools to describe interesting intelligent behaviours besides programming, we have to rely on empirical verification of our implementations.

In spite of these differences it is useful to examine the methods and techniques developed in Structured Programming, and try to adapt them in such a way that they become applicable in AI programming. One of the important methods of Structure Programming is structural design and implementation. This includes notions like "top-down programming", "stepwise refinement" (Wirth, 1972), "levels of abstraction" and problem decomposition (Dahl & Hoare, 1972). A key concept on which these methods are based is a hierarchy of conceptualizations. Dijkstra (1972) formulates it as follows: "The notion of a hierarchy implies that what at one level is regarded as an unanalysable unit, is regarded as a composite object at the next lower level of detail, for which the appropriate grain is an order of magnitude smaller than the corresponding grain appropriate at the next higher level". Such a hierarchy should be reflected in the structure of the program. For reasons similar to those discussed above in relation to the nature of AI problems and programs, I doubt whether this hierarchical program construction method is directly applicable to AI programming. On a high level, AI lacks clear conceptualizations of behaviour and representation of information. On the somewhat lower level

of implementing AI ideas, however, the notion of a hierarchy of abstract concepts is relevant. For example, notions like process, procedural representation of knowledge, pattern directed invocation, frame etc. have become common AI terminology, but the actual implementation of these concepts (eg. in LISP) is often a painful process. There seems to be a gap between the level of abstraction that is used in the design and description of AI programs and the actual concepts provided by the programming languages we use, which typically provide few control constructs beyond recursion and backtracking.

The concepts that Structured Programming has come up with are too low level to bridge this gap; however we should try to develop a hierarchy of concepts which are suitable for AI purposes, and which are implementable at the various levels of abstract machines that correspond to the levels in the hierarchy. Two approaches can be (and have been) used to construct and implement such a hierarchy of concepts: the bottom-up approach, so basically start with building a general framework which serves as a basis for further work, and the top-down approach, which selects a specific problem and develops its own tools for solving the problem. Both approaches have their own specific merits. Without the first approach we would not have the powerful tools available that we have now (INTERLISP, QLISP, CONNIVER, KRL, Production Systems), but without the second approach we probably wouldn't have had many interesting AI programs. There is, however, a constant danger in AI research that is related to these two different methodologies: sliding from one methodology into the other without being fully aware of the consequences of this methodological shift. Many AI projects start out with a specific problem in mind, but after working on this problem for a while, one gets dissatisfied with the available tools and starts implementing new features in an existing system, or even building an entirely new system (1). Building an AI programming system is however a major effort which requires a well thought out design, a good and efficient implementation and proper maintenance and thus, a lot of manpower and a great amount of determination. The chances that a badly planned effort will result in a programming language and environment which satisfies the requirements we have set earlier, are -I believe- minimal. Even systems like CONNIVER and QLISP do not seem to live up to the expectations that the AI community had of them a few years ago. This may be so because of the particular implementations (efficiency and interfacing problems), but it is my opinion that a more important reason is the lack of clear conceptualisations of the ideas that were implemented in these languages. The ideas are powerful, but actually implementing them in such a way that users benefit from them and are not restricted by them, requires a lot of thought. I do not want to suggest that building systems like CONNIVER and QLISP is not a useful thing to do, but we should acknowledge their experimental nature and not expect them to be general tools from the start. Thus, the top-down approach to AI programming can produce new and interesting ideas and programming language features, but only after careful consideration and after having developed a clear idea of the purposes and the range of application should new concepts and features be incorporated in AI languages for general use (see also Hoare, 1973 for a more elaborate discussion of this issue).

The alternative approach is an intermediate one: provide the user of a

---

(1) This may be particularly the case in situations where only limited tools are available (eg. only simple LISP or POP-2 systems) like on many European sites.

language with a number of standard useful concepts, but also with the mechanisms to build his own hierarchy of concepts in an easy way. Mechanisms of this sort are often called abstraction mechanisms. In section 3.3 we will discuss the notion of abstraction in detail.

A second issue raised in the Structured Programming literature is that of formal specification of the behaviour of programs. As has been said before, no proper tools are available for formalising the complex behaviour of AI programs. The idea of describing certain properties of a program is however a useful one. Winograd (1974) introduces a style of programming, "conceptual programming", in which a program is not only specified by the actual code, but also by a large and rich structure of descriptions. These descriptions specify the global structure of the program, input/output behaviour, intentions, goals, conditions and expected results of components of the program. These descriptions are not only useful for debugging, analysing and understanding the program, but they may also be used by a computer system or the program itself for monitoring and debugging purposes. In this latter sense, the descriptions may become a (partial) formal specification on which a (partial) verification (and possibly a meta-evaluation) can be based. Several research projects are on the way to implement such facilities (see section 3.5).

Summarizing the possible contributions that Structured Programming has to offer to a programming methodology for AI, I would say that Structured Programming has raised two important issues: 1) the importance of the abstraction process in programming and the need for abstraction mechanisms, and 2) the need for different levels of descriptions of programs. Although the Structured Programming methodology puts much emphasis on the process of program construction, I believe that it may be premature to establish a strict methodology for AI program construction, but that we should rather concentrate on developing tools to help the process of AI programming. The following section will discuss such tools in more detail.

3. The role of Programming Languages and Systems.

3.1. Dimensions of programming languages.

In this section we will discuss how programming languages and systems can be used (or should be developed) so as to achieve the goals of an AI programming methodology as sketched in section 1. Bornat and Brady (1975) have suggested that programming languages should be described and evaluated along three (not completely separable) dimensions: an underlying machine dimension, a linguistic dimension, and a user dimension. The underlying machine dimension is concerned with the abstract machine on which programs in the language can be run. This aspect of programming languages is important since the abstract machine determines the behaviours that the language can produce. The linguistic properties of a programming language include not only the syntactic features, but also reflect the general expressiveness of the language. The user dimension encompasses most of what is called the programming environment. For the discussion of programming languages and programming systems for AI, I will adopt the categorization given by Bornat & Brady, with two amendments.

The linguistic features that a language provides can be seen as

abstractions from sequences of primitive operations of the underlying abstract machine. Moreover, abstraction mechanisms are tools to enhance the expressive power of a language. For these reasons I would prefer to use the term abstraction dimension although the two concepts are meant to be the same. In addition to the three dimensions, there is a fourth dimension which seems important in AI programming, but which seems less related to programming languages in particular, the descriptive dimension, representing the mechanisms available for constructing a multi-level description of programs. In connection with the notion "conceptual programming", we have discussed the relevance of such descriptions; in section 3.5 we will try to relate the descriptive dimension to the other dimensions.

3.2. Abstract machines.

Every programming language (from MACRO-10 to KRL) has a corresponding underlying abstract (virtual or real) machine which is the processor by which the programs in the language are executed. The abstract machine is characterized by the primitive operations it provides, while the set of primitive operations determines the processes that can be implemented in the language (1). In the process of analysing and describing a programming language or the design of a programming language, it is of prime importance to isolate the principles on which the abstract machine is based because, only when the abstract machine is clearly understood, can the possibilities and restrictions that the language offers become transparant, and thereby the range of applicability. The FUNARG problem in LISP was a typical example of opaque features of the LISP abstract machine, which are now well understood. Similarly, certain design decisions (like scope rules for passing functional values as results in ALGOL-68) can be exposed and analysed, once the abstract machine is defined.

The question that arises in this context, is how to describe abstract machines, how to compare them and evaluate their properties. One way to achieve a formalized definition of abstract machines is to represent them by definitional interpreters (Reynolds, 1976), written in a language which is well understood and has a clear semantics, like pure LISP or the Lambda calculus. Definitional interpreters have been used to define LISP (McCarthy et al. 1962), GEDANKEN (Reynolds, 1971), ISWIM (Landin, 1966). Although these interpreters give some idea of what the abstract machine that underlies a language looks like, it remains difficult to compare the different abstract machines, and to understand why certain features are present or absent in the language. An approach to this problem can be to specify a more general abstract machine, which accommodates all features of a certain set of languages, and to define each particular language in terms of an abstract machine which is an instantiation of the general model. Landin's (1966) SECD machine can be thought of as such a general machine. Bobrow & Wegbreit (1973) proposed a general process model -commonly known as the "spaghetti stack model"- for describing the interactions between control behaviour and environments in languages like

---

(1) Since a programming language usually represents a universal Turing machine, we can of course implement anything in it by writing specialized interpreters (like CONNIVER was implemented in LISP). Here we will ignore this trivial interpretation of the notion of implementation, and will only be concerned with programs that can be expressed in the primitive operations of the underlying abstract machine and abstractions thereof.

LISP. Such models can, as Bobrow & Wegbreit have shown, 1) provide insight in particular constructs in a language, and 2) give indications how certain behaviours can be recognized by the interpreter and be implemented efficiently. The Bobrow & Wegbreit model was however rather oriented towards LISP and failed to describe constructs of other languages properly, such as SIMULA Classes, partial application as in POP-2 (Burstall et al. 1971), and control-closures (as generated by Landin's J-operator). Steele (1977) proposes several modifications and extensions of the Bobrow & Wegbreit model, which allow for static as well as for dynamic binding, and for control-closures.

Bornat & Wielinga (1976) proposed a model based on the ideas of Bobrow & Wegbreit, but with a somewhat wider scope, and without the implementation details (although a prelimenary version of the model was implemented in EVIL (Bornat & Wielinga, 1978) with spaghetti stacks). In this model computational states of an abstract machine are represented as data-structures: Process-State-Descriptors (PSD). A PSD contains slots for specifying the operations that a process in a given state must (can) perform, a slot for local variables, a slot for an access-environment, a slot which describes what action that is to be taken when the process is finished or suspends itself, and a slot for a description of the set of active processes this process is part of. The PSD model accounts for a variety of objects, such as functional values, access-closures (LISP's FUNARG), control-closures (cf. Landin's J-operator, JUMPOUT in POP-2, escape functions), label values (TAG's in CONNIVER), SIMULA CLASSes, partially applied functions (POP-2) and the various closures that can be created in EL1 (Wegbreit, 1973) (see for more details Bornat & Wielinga, 1978).

Formulating an abstract machine in a model like the PSD model helps in understanding the language and its design criteria and may lead to further development of programming language features. Moreover, a language which is based on a well-designed abstract machine, will not only provide well-structured control primitives, but will also be more efficient since the underlying machine can recognize the corresponding control behaviours and hence implement them efficiently. The methodological consequence of this is that programming language designers should continue to develop new abstract machines and new models for computational behaviours, in order to enrich the vocabulary of programming concepts that the AI programmer can use efficiently. The ACTOR-model of computation (Hewitt, 1973) is an example of a model that showed clearly how restrictive traditional programming languages (based on function calls as the basic control mechanism) were, and which provided new ways to think about and to describe control in programs.

A second consequence is that LISP and the abstract machine that underlies LISP will not remain adequate for AI research. The basic LISP machine is too restricted. Extensions such as spaghetti-stack primitives and various facilities which are based on the error-handling mechanism (eg. in INTERLISP) are -I believe- really patches to the basic machine. An entire new design of the abstract machine for new AI programming languages is needed, one which incorporates primitives to handle such notions as process, functional datastructures, data-driven processing, multi-processing.

An example of a class of language features which requires abstract machines which differ from the ones we have discussed above, is the class of generalized invocation mechanisms (Davis, 1977), such as event-driven invocation, goal-directed invocation, content-directed invocation, etc.

Mechanisms of this type are used in Production Systems (a good overview can be found in Davis & King, 1975), PLANNER (Hewitt, 1972; Sussman & Winograd, 1970), QA4 and QLISP (Rulifson, 1973; Wilber, 1976), THERESIAS (Davis, 1977). The common idea that underlies the invocation mechanisms in these languages is that a program can be structured as a set of cooperating knowledge sources, KS (Hayes-Roth & Lesser, 1977), represented as procedures, processes or production rules. In such a system the central activity is to decide what KS to invoke next (Davis, 1977). Such a decision can only be made when more is known about a KS than just its name. Invocation occurs on the basis of some sort description of what knowledge is incorporated in the KS or what goals the KS will try to achieve. This knowledge has to be explicitly present in the system and the system must be able to use this knowledge. In programs written in the more traditional languages (LISP, ALGOL, POP-2, EVIL) this knowledge is implicit in the structure of the program. In section 4 we will elaborate a bit more on the role of this type of knowledge in AI programming.

3.3. Abstraction.

In relation to the Structured Programming methodology we have discussed the process of abstraction in general terms, and contended that abstraction is among the most powerful tools in programming (Dijkstra, 1972). The abstraction dimension of programming languages is important because a programming language is more than just a vehicle for communicating a problem solution to a machine, it is a problem-solving tool. The linguistic repertoire of a language imposes a particular structure on the problem solving behaviour of the user (Sussman & McDermott, 1973). So, the abstractions that are made (or can be made) in a language will affect its use and applicability in a certain domain.

Although the discussion of abstraction in programming languages is fairly recent, the process of abstraction has been the basis of the construction of most high-level languages like ALGOL-60, POP-2, and perhaps even FORTRAN. The linguistic constructs that these languages offer can be thought of as abstractions from the behaviour of the underlying machine. The FOR statement in ALGOL-60 is an abstraction of some sequence of primitive operations of the ALGOL machine. In this interpretation, LISP is not a high-level language, since it doesn't provide many more linguistic structures than the primitives of the LISP machine. POP-2 which has an underlying machine similar to the LISP machine, has a larger repertoire of abstractions, like datastructures, iteration constructs, macro's. I view the control and data structures built in a language as implicit abstractions, constructed by the programming language designer, which, in many circumstances, the user is forced to use whether he wants or not (eg. array's in ALGOL-60).

Besides this type of abstraction, there is in most programming languages a way of introducing abstractions explicitly. Traditionally these include procedure construction and macro's. Recently, there have been several attempts to introduce more powerful abstraction mechanisms in programming languages. Relatively young languages like PASCAL (Jensen & Wirth, 1974), ALGOL-68 (van Wijngaarden etal. 1969) provide abstract data types. SIMULA-67 (Dahl et al. 1968) has a powerful abstraction mechanism in the form of the class concept, which can be seen as a data abstraction with associated operations. The idea to relate data objects to the set of operations that are applicable to these objects is the basis of more recent constructs like the CLU cluster (Liskov et al. 1977), the ALPHARD form (Wulf, 1974; Shaw, 1977) and the EUCLID module (Popek et al. 1977). These languages also provide various forms of control abstractions.

ALPHARD incorporates a mechanism for specifying iterative constructs operating on abstract entities, without explicit dependence on the representation of these entities. CLU has similar control abstraction mechanisms.

The basic rationale for introducing these new and powerful abstraction mechanisms in programming languages is a methodological one: abstractions separate implementation details from the actual desired behaviour and use of a data object, they allow incremental program development by specifying abstract properties of modules separately from their implementation. Abstraction techniques can be of use in AI programming because they may provide us with a tool to master the complexity of large AI programs, provided the abstraction mechanisms do not impose too many restrictions on the user. The loop specialization facility in ALPHARD for instance, discourages (or may even make impossible) the construction of loops in which only parts of a structure are used or modified in the loop. Such restrictions are probably unacceptable for the AI user.

TELOS (Travis et al. 1977) is a PASCAL-based AI language, which contains a number of abstraction mechanisms tailored to AI requirements as well as other features like event handling mechanisms (demons), co-routines, and an associative database. TELOS provides a powerful set of tools for AI programming which are more up to date with the current state of the art in Computer Science than traditional AI languages. The lack of a flexible, interactive programming environment may, however, hamper the use of this language for major AI projects. CSSA (Computing System for Societies of Actors; Boehm et al. 1977) provides abstraction mechanisms based on actors. Although this language seems to contain interesting ideas, it is not (yet) a full-fledged AI language. In any case, the introduction of explicit abstraction mechanisms tailored for AI needs in new programming languages for AI seems a promising way to proceed and deserves more attention.

3.4. The programming environment.

In an earlier section we have propounded that AI programming needs a flexible environment in order to facilitate the construction and analysis of large and complex programs. Crucial to this aim is an interactive environment where all activities of AI programming can be performed in an integrated way: creating a program, running and debugging it, changing it, monitoring and analysing its behaviour and documenting it. Classical programming systems (oriented towards non-interactive languages like ALGOL, FORTRAN, PASCAL etc.) separate these processes: compiling, editing, and documenting are done by separate programs which do not (or hardly) communicate. Programming languages in which programs can be handled as data structures in the language itself, and which are interactive (notably LISP, ELl and EVIL), can provide a more flexible service to the user. LISP, being one of the first languages where programs were accessable to the programmer in the form of lists, changed from a rather simple, mathematically oriented language to the basis of highly sophisticated interactive systems (INTERLISP, MACLISP; see for an extensive account of interactive programming in LISP Sandewall, 1978). INTERLISP (as running on a PDP-10, under the TENEX operating system; Teitelman, 1975) provides probably the most extensive programming assistance that is currently available. The INTERLISP system is centered round a basic LISP interpreter with spaghetti-stack facilities (Bobrow & Wegbreit, 1973) and augmented with an enormous variety of LISP functions and utility packages. The error handling part of the interpreter can transfer control to several modules of the Programmer's Assistant: CLISP, DWIM, break package, error message

package etc. CLISP handles ALGOL-like expressions like IF-THEN-ELSE and WHILE constructs which are not normally provided in LISP (Teitelman, 1973). DWIM (Teitelman, 1972) is a package that detects and optionally corrects spelling mistakes. At break level a variety of information and actions is available: elaborate error analysis, several forms of backtrace, changing variables etc. Sophisticated editing, pretty-printing and cross-reference facilities are also available. Most of these functions have also been implemented in other systems (eg. UCI-LISP provides similar packages) but the essential feature of INTERLISP is the high degree of integration of these modules. The recent development of a "Display Oriented Programmer's Assistant" (Teitelman, 1977) provides a flexible and highly integrated environment where the user can manage several simultaneous tasks and contexts for creating programs, testing them, editing and interacting with the LISP interpreter.

Although research along this line is important because ease of programming clearly is a major requirement of AI, its scope is limited, since this approach is basically non-knowledge based. The INTERLISP system knows about simple mistakes that people make, it knows about internal and external representations of LISP functions and it does a lot of administrative work. It does not, however, know much about the actual program the user is working on, its conceptual structure, goals and intentions, relation to other programs etc. Significant improvements of the capabilities of a programmer's assistant can only be made if the system has a deeper understanding of the program under construction and the methods and goals of the programmer. In section 3.5 we will discuss how such systems could possibly be developed.

Working in a flexible interactive environment has another methodological consequence, besides enhancing the ease of programming: it affects the style of programming. Programming in LISP is often done incrementally, i.e. functions are written, tested and changed as individual units, one after the other; the program is gradually developed in a bottom-up way. This phenomenon is partially due to the interactive environment and partially to the basic structure of the LISP interpreter: the common database and the dynamic binding scheme. In a static binding environment (such as is provided in languages like ALGOL) much larger pieces of a program have to be developed at one time. The lexical scope rules in ALGOL-like languages enforce much more structure on a program than LISP does.

The style of programming that seems "natural" to LISP has been called "structured growth" (Sandewall, 1978): program development starts out with a simple initial program, which is tested and allowed to grow by increasing the complexity of individual modules. The experimental approach to AI probably requires a programming environment which supports structured growth, as well as more top-down directed methods. However, it seems to me that LISP doesn't provide enough structuring tools to properly support such a programming methodology. Abstraction mechanisms which separate conceptual structures from their implementation are -I feel- a minimal requirement for a programming language that is to support structured growth.

3.5. Program descriptions and program understanding.

Earlier we have introduced the term "conceptual programming" (Winograd, 1974; Sandewall, 1977), representing a programming style which integrates all activities of programming in a more "natural" way, preferably with the help of a "knowledgeable" computer system. Central to the conceptual

programming idea is the view of a program as a large information structure, containing not just the actual code, but also -and even more important- more conceptual representations of the program: specifications of the problem it is trying to solve, descriptions of internal goals and intentions, input/output behaviour, overall structure etc. Traditionally only a few representations of programs have been used: program text, with or without comments, flow-charts or tree-diagrams, cross-reference listings, assertions, but only the program text is interpreted by the computer system (apart from program verification systems which may use assertions, but these systems are not advanced enough to handle complex AI programs). The introduction of types in programming languages was a small step towards a more conceptual representation of the various aspects of a program, but in general too restrictive for AI programming. The type of the LISP function MAPCAR for instance, should be represented as something like "a function which takes as arguments a list and a function, and delivers as result a list of objects of the type that the functional argument returns". Winograd (1974) argues rightly for semantic (rather than syntactic) descriptions of objects in programs. Such a description should represent the conceptual role of an object in a program. An intelligent Programmer's Assistant can use these descriptions in the process of debugging, checking and even constructing parts of the program. Given explicit descriptions of the conditions that part of a program requires, and of the goals that it is trying to achieve, such a system could dynamically check these descriptions and debug the program. HACKER (Sussman, 1973) shows how a programming system can do this and even learn from doing it, in a specific domain. The basic idea behind HACKER is a programming system that constructs programs along with descriptions of their teleological structure. This description is then used to debug the program by running it in "careful" mode in which the described goals are continously checked against the actual performance of the program. This research and related work of Goldstein (1974), Sacerdoti (1975), and Waldinger (1975) on the representation of program structure in the form of plans, can provide a conceptual framework for programming systems which "understand" programs. Projects more or less based on such a framework are on the way to construct an "Programmer's Apprentice", i.e. a system which works together with an expert programmer on the construction of realistic programs (Rich & Shrobe, 1977; Smith & Hewitt, 1974). These systems emphasize -in contrast with automatic programming systems (Waldinger & Manna, 1977; Green, 1977)- the cooperative relationship between the programmer and the programming system. The role of the computer system is primarily to help in managing the low level details of the program and to guarantee that it will be correct. Building such systems seems a more realistic goal than the construction of fully automated programming systems.

Even if we put the long term goal of a Programmer's Assistant aside for the time being, the concept of program descriptions remains a relevant issue for programming language designers. One of the cheered features in LISP is the fact that programs can be handled as data, and as such provide -on a very primitive level- a description mechanism (which is actually used for debugging purposes). A programming language which is to support conceptual programming requires both rich linguistic facilities to allow the user to express the conceptual structure of his program, as well as it needs a powerfull abstract machine which is able to interpret these descriptions. One of the claims for KRL is that it encorporates such facilities, at least for declarative aspects of programs: descriptions can be associated with other descriptions, to describe what they are for and how they can be used. Great care should be taken however, to define clearly how such meta-descriptions can be used by the system. KRS (Smith,

structures) on many levels in domain-specific terms. This richness of representation may remove the need for complex process interactions.

Summarizing these points, I would say that there is no competence/performance dichotomy in knowledge based programs, just a continuum of understanding of the problem domain. The more specific descriptions of knowledge units become available, the less general heuristics and meta-knowledge needs to be incorporated: competence becomes performance. In AI the understanding of the problem domain is reflected in the programming methodologies. The general methodology of rule-based systems, general invocation systems and heuristic rule guided systems is characteristic of domains which are not very well understood, which are ill-structured or which are genuine problem solving situations where search is really necessary. Such domains include; medical diagnosis (MYCIN; Shortliffe, 1974), mathematical discovery (AM; Lenat, 1976), mass spectra analysis (Meta-DENDRAL, Buchanan & Mitchell, 1977), and general problem solving (GPS; Newell & Simon, 1972). Systems that work in well understood domains like the blocks world (Winograd, 1972) often use knowledge representations in which the use of knowledge is predefined and concentrate on procedural descriptions of how to achieve a goal. These systems are characterized by a complex flow of control which is not easily represented in rule-based systems. The programming methods used are those of eg. PLANNER, QLISP or CONNIVER.

What then is the effect of this view on AI programming methodology. One consequence I can see, is the need for tools to represent the range of different levels of understanding of a problem domain. KRL (Bobrow & Winograd, 1977) is clearly in line with this methodology: it provides mechanisms for building rich descriptions of knowledge from different view points. It may turn out that the multiple descriptions and matching mechanisms of KRL are an alternative for powerful control structures, provided we know what knowledge to describe. In a similar spirit Davis (1977) argues for the introduction of generalized invocation mechanisms in programming systems, which allow retrieval of a piece of knowledge by general description, rather than just by its goal like in PLANNER. On the other hand, when a domain is not very well understood, a powerful descriptive mechanism for representing control information is needed. AMORD (de Kleer et al. 1977) provides a mechanism for explicit control in reasoning. The internal control domain is formalized in such a way that the system can manipulate the control assertions. The explicit representation of facts about how knowledge can be used, allows the possible qualification of the use of this knowledge.

I would argue for a methodology geared towards the construction of mechanisms for representing the multiplicity of understanding, i.e. systems which can switch from tightly controlled goal directed behaviour, to a much more explorative use of knowledge. Moore & Newell (1973) define the criterion for understanding of knowledge to lie in its use: understanding a piece of knowledge is using that knowledge whenever appropriate. Such appropriate use may come about through careful consideration of the current control state of the system, i.e. its goal and intentions, or through extensive descriptions of the applicability of the knowledge that is available. Tools for describing the multiplicity of understanding may help in the process of constructing a theory of intelligence that accounts both for the erratic behaviours that humans display in new situations or when confronted with new tasks, and for the performance of skilled experts. As Neisser puts it: "Some thinking is deliberate, efficient, and obviously goal-directed; it is usually experienced as self-controlled as well. Other mental activity is rich,

chaotic, and inefficient; it tends to be experienced as involuntary, it just 'happens'. It often seems to be motivated, but not in the same way as directed thought" (Neisser, 1967, p297).

5. Concluding remarks.

In the beginning of this paper we have posed a number of questions that an AI programming methodology should address. Although no clear-cut answers were given, I hope to have identified some methods to enhance future AI programming. One of these methods is the development of new languages for AI programming. I believe AI is outgrowing (or has already outgrown) LISP (see Anderson (1976) for more arguments), and needs an entirely new framework which can serve as a basis for AI research. Not that I believe that there will be -and ever can be- an ultimate AI language, but it is my feeling that the time is there to construct a new system for AI programming, which can serve as a new platform, on a somewhat higher level than LISP, on which the AI programmer can stand. Such a system should reflect the current state of the art in Computer Science (i.e. have proper tools for handling abstraction, processes etc), as well as incorporate well-established AI ideas. It should provide a powerful programming environment, eg. based on the experience with INTERLISP, and incorporate as much ideas of program description facilities and program understanding as is feasible, given the current state of the art. Building such a system will undoubtedly require a lot of effort. KRL may be a step into this direction, but it may also turn out to be too much orientated towards LISP on the one hand and towards natural language on the other hand, to become a general tool for AI. In this paper I have concentrated on rather general issues, and neglected some interesting developments of specific features of AI programming languages and systems. The developing of these new features and experimenting with them is important, but I believe with Hoare (1973) that programming language design and the designing of new features should be two separate things. The programming language designer should not include untried ideas of his own in a new design, but rather try to capture the essence of many well thought out ideas and implement them as clear conceptualizations. This may seem a rather conservative view on the AI programming language effort, but it is my feeling that it may be a key to a successful programming methodology.

The development of tools and languages for AI is only one way to achieve the goals we have set ourselves. The development of formalisms for specifying AI theories and for representing the precise nature of the problems we are working on, is equally important. Burstall & Goguen (1977) put it the following way: "the main intellectual task of programming is elaborating the theories which describe all concepts used in the actual program". The work on formal semantics of representational schemes is an important area of AI research and should receive much more attention of AI programmers. Only when the formal properties of such schemes are properly understood, will there be a chance that they become more widely used and can serve as a new firm base of AI programming.

Both these methodological approaches -developing tools for writing better programs in a more efficient way, and developing better representational formalisms- will hopefully converge towards the central goal of AI methodology: create a better understanding of the efforts and results of the AI discipline, and to give AI programs a better understanding of what they are doing and what they are supposed to do.

Acknowledgements.

Many of the ideas behind this paper were shaped in discussions with the ESSEX AI group. I am particularly indebted to Bruce Anderson, Pat Hayes, Bernhard Suffrin and especially Richard Bornat and Mike Brady. Steve Hardy made many valuable suggestions for improvements of this paper. Thanks also to Jan Elshout, who made valuable comments and with whom I had many stimulating discussions on human intelligence.

6. References.

Anderson, D.B. A brief critique on LISP. Proc. of the AISB Summer Conf. Edinburgh, 1976.

Bobrow, D. G. & Wegbreit, B. A model and Stack implementation of Multiple Environments. CACM 16,(1973),591.

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

Bohm, H.P. Fischer, H.L. and Raulefs, P. CSSA: Language Concepts and Programming Methodology. Proc. of the Symp. on AI and Progr. Lang. Rochester, 1977.

Bornat, R. & Brady, J.M. Dimensions of Programming Languages. Informal notes, University of Essex, 1975.

Brady, J.M. Dangerous Behaviour. AISB Conf. Hamburg, 1978.

Bornat, R. & Wielinga, B.J. Does AI programming really have to be like knitting with spaghetti?. Proc. of the AISB Summer Conf. Edinburgh, 1976, 65.

Bornat, R. & Wielinga, B.J. A Process State Description model of programming languages. In preperation, 1978.

Bornat, R. & Wielinga, B.J. The EVIL programming system. In preperation, 1978.

Brachman, R.J. What's in a concept: structural foundations for semantic networks. Int. Journ. Man-Machine Studies, 9,(1977),127.

Buchanan, B.J. & Mitchell, T. Model directed learning by production rules. Stanford memo HPP-77-6, 977.

Burstall, R.M., Collins, J.S. and Popplestone, R.J. Programming in POP-2. Edinburgh Univ. Press, 1971.

Burstall, R.M. & Goguen J.A. Putting Theories together to make Specifications. Int. Joint Conf. on Atrificial Intelligence 5.

Cercone, N. & Schubert, L. Toward a state-based conceptual representation. Proc. of the IJCAI-75, Tblisi, USSR, 1975, pp83-90.

Dahl, O.J. & Hoare, C.A.R. Hierarchical Program Structures. In: Structured Programming. Academic Press, 1972.

Dahl O.J. et al. The SIMULA 67 Common Base Language. Norwegian Computer Center, Oslo, 1968.

Davis, R. & King, J. An overview of Production Systems. Stanford AIM-271, 1975.

Davis, R. Generalized Procedure Calling and Content-directed invocation. Proc. of the Symposium on Artificial Intelligence and Programming Languages. Rochester, 1977.

Dijkstra, E. W. Notes on Structured Programming. In: Structured Programming, Academic Press, 1972.

Dijkstra, E.W. The humble programmer. CACM, 15(1972).

Dijkstra, E.W. A Discipline of Programming. Englewood Cliffs: Prentice-Hall, 1976.

Dijkstra, E.W. Programming: from craft to scientific discipline. Proc. of the ICS77/ACM Conf. Belgium, 1977.

Draper, S. On the possible relationship between A.I. and Psychology. AISB Quarterly, 29,(1978).

Green, C. A summary of The PSI Program Synthesis System. Proc. of the Fifth Int. Joint Conf. on Artificial Intelligence, Boston,1977.

Goldstein, I. & Roberts, R. B. NUDGE a knowledge-based scheduling program. Proc. of IJCAI-77. Dept. of Comp. Science, Carnegie Mellon, 1977.

Guzman, A. Computer Recognition of Three-Dimensional Objects. thesis, MIT, 1968.

Hawkinson, L.B. The representation of concepts in OWL. Proceedings of the IJCAI-75, MIT, Cambridge Mass. 1975, pp 107-114.

Hayes, P.J. The Language Golux. University of Essex, 1975.

Hayes, P.J. Computation and Deduction. Proc. Symp. Math. Found. Of Computer Science, Czech Acad. of Sciences, 1973.

Hayes, P.J. Some problems and non-problems in representation theory. Proceedings of the AISB conference, Sussex, 1974.

Hayes-Roth, F. & Lesser, V.R. Focus of Attention in the HEARSAY II Speech Understanding System. CMU report, 1977.

Hoare, C.A.R. Hints on Programming Language Design. Stanford AIM-224, 1973.

Hewitt, C. Description and Theoretical Analysis (Using Schemata) of Planner, A Language for Proving Theorems and Manipulating Models in a Robot. MIT AI memo 251, 1972.

Hewitt, C., Bishop, P. and Steiger, R. A universal modular actor formalism for Artificial Intelligence. Proc. of the IJCAI-73, Stanford, 1973.

Horn, B.K.P. Understanding image intensities. Artificial Intelligence 8,(1977),201-231.

Jensen, K. & Wirth, N. PASCAL User manual and report. Springer-Verlag, 1974.

de Kleer, J., Doyle, J., Steele, G.L. and Sussman, G.J. AMORD: Explicit Control of Reasoning. Proc. of the Symp. on AI and Progr. Lang. Rochester, 1977.

Kowalski, R. Predicate Logic as programming language. Univ. o Edinburgh, Comp. Logic memo 70, 1973.

Landin, P.J. The mechanical evaluation of expressions. Comp. Journ. 6(1964),308.

Landin, P.J. A correspondence between ALGOL 60 and Church's lambda notation. CACM 8(1965),89-101,158-165.

Landin, P.J. The next 700 programming languages. CACM 9(1966),157.

Lenat, D.B. AM: An artificial intelligence approach to discovery in mathematics as heuristic search. Stanford AIM-286, 1976.

Lenat, D.B. The Ubiquity of Discovery. Artificial Intelligence, 9,(1977)257.

Liskov, B., Snyder, A., Atkinson, R. and Schaffert, C. Abstraction Mechanisms in CLU. CACM 20(1977),564.

Marr, D. Arificial Intelligence- a personal view. Artificial Intelligence, 9(1977)37.

McCarthy, J., Abrahams, P.W., Edwards, D.J., Hart, T.P. and Levin, M.I. LISP 1.5 Programmer's manual. MIT Press, 1962.

McDermott, D.V. & Sussman, G.J. The CONNIVER reference manual. MIT AI memo 259, 1972.

McDermott, D. V. Artificial Intelligence meets Natural Stupidity. Sigart Newsletter 57 (1976), 4.

Moore, J. & Newell, A. How can Merlin understand? In L. Gregg (Ed.), Knowledge and Cognition. Hillsdale, N.J.: Lawrence Erlbaum Assoc., 1973.

Moses, J. Symbolic Integration. MIT Ph.D. Thesis, 1967.

Neisser, U. Cognitive Psychology. New York: Appleton-Century-Croft, 1967.

Newell, A. & Simon, H. A. Human Problem Solving. Englewood Cliffs, N. J.: Prentice Hall, 1972.

Popek, G.J., Horning, J.J., Lampson, B.W., Mitchell, J.G., London, R.L. Notes on the design of EUCLID. Proc. of an ACM Conf. on Language

Design for Reliable Software. SIGPLAN Notices 12(1977)11.
Pratt, V.R. The competence/performance Dichotomy in Programming. In; Proc. of the Symp. on Principles of Programming Languages, Los Angeles, 1977.
Reynolds, J. C. GEDANKEN- A Simple Typeless Language. CACM, 13, 1971.
Reynolds, J.C. Definitional Interpreters in higher order programming languages. Proc. ACM 25th Nat. Conf. Boston, 1976.
Rich, C & Shrobe, H.E. Initial report on a LISP Programmer's Apprentice. MIT AI-TR-354, 1976.
Rulifson, J.F., Derksen, J.A., Waldinger, R. QA4: A language for writing problem solving programs. SRI AI Center Tech. Note 81, 1973.
Sacerdoti, E.D. A structure for plans and behaviour. SRI Technical Note 109, 1975
Sandewall, E. Some observations on Conceptual Programming. In E.W. Elcock & D. Michie Machine Intelligence 8, Chichester: Ellis Hornwood Ltd. 1977.
Sandewall, E. Programming in the interactive environment: the LISP experience. Comp. Surveys, 10 (1978) 35.
Shaw, M., Wulf, W.A., London, R.L. Abstraction and Verification in ALPHARD: Defining and Specifying Iteration and Generators. CACM 20,(1977),553.
Shortliffe, E.H. MYCIN- A rule based computer program for advisang physicians regarding antimicrobal therapy selection. Stanford AIM-251, 1974.
Smith, B. Hewitt, C. Towards a programmer's apprentice. Proc. of the AISB summer conf. Sussex, 1974.
Smith, B. Knowledge Representation Semantics. Proc. of the IJCAI-1977, Carnegie-Mellon, Pittsburg, 1977, p989.
Sussman, G.J. & Winograd, T. The Microplanner reference manual. MIT AI memo 203, 1970.
Sussman, G.J. & McDermott, D.V. Why is CONNIVING better that PLANNING? MIT-AI memo 255a, 1973.
Sussman, G.J. A computational model of Skill Acquisition. MIT TR-297, 1973.
Steele, G.L. Macaroni is better that spaghetti. Proc. of the Symposium on Artificial Intelligence and Programming Languages(Rochester, 1977.
Travis, L. & Honda, M. Design Rationale for TELOS, A PASCAL-based AI Language. Proc. of the Symposium on Artificial Intelligence and Programming Languages, Rochester, 1977.
Teitelman, W. Automated Programming - The Programmer's Assistant. Proc. Fall Joint Comp. Conf. 1972.
Teitelman, W. CLISP - Conversational LISP. Int. Joint Conf. on Artificial Intelligence 3, 1973.
Teitelman, W. et al. Interlisp reference manual. Xerox Palo Alto Res. Center, 1975.
Teitelman, W. A Display Oriented Programmer's Assistant. Int. Joint Conf. on Atrificial Intelligence 5, 1977.
Wegbreit, B. Procedure Closure in EL1. The Computer Journal, 17(1973)38.
van Wijngaarden, A., Mailloux, B.J., Peck, J.E.L. and Koster, C.H.A. Report on the algolrithmic language Algol 68. Num. Mathematik, 14(1969)79.
Waldinger, R. Manna, Z. The automatic synthesis of recursive programs. Proc. of the Symp. on AI and Progr. Lang. Rochester, 1977.
Waldinger, R. Achieving several goals simulaneously. SRI Technical Note 107, 1975.
Waltz, D. Understanding Line drawings of Scenes with shadows. In: P. Winston (Ed), The psychology of computer vision. New York: McGraw-Hill, 1975.
Warren, D.H.D. & Pereira, L.M. PROLOG: the language and its implementation

compared with LISP. Proc. of the Symp. on AI and Progr. Lang. Rochester, 1977.

Wilber, B.M. A QLISP manual. SRI Tech. Note, 1976.

Winograd, T. Understanding Natural language. Academic Press, 1972.

Winograd, T. Breaking the complexity barrier (again). Proc. of the ACM SIGIR-SIGPLAN interface meeting. Nov. 1973.

Winograd, T. Five lectures on Artificial Intelligence. Stanford AIM 246, 1974.

Wirth, N. Program development by stepwise refinement. CACM, 14, 1972

Woods, W.A. Whats in a link: foundations for semantic networks. In: Bobrow & Collins (Eds.) Representation and understanding. New York: Acad. Press, 1975.

Wulf, W. A. ALPHARD: Towards a Language to Support Structured Programs. Carnegie-Mellon, 1974.

GRAMMAR DISCOVERY AS DATA COMPRESSION

J. G. Wolff
Department of Psychology, The University, Dundee, Scotland.

A discovery procedure is described for non-recursive context-free phrase structure grammars which is based on three data compression principles. It works without the need for semantic information, negative samples, a teacher or pre-segmented data.

This paper describes a grammar discovery procedure (GDP) for non-recursive context-free phrase structure grammars which has been developed as part of an exploration of the kinds of processes which may enable children to discover linguistic and non-linguistic cognitive structures. It is described more fully elsewhere (Wolff, in press). The distinctive features of this GDP are these:-

1. Unlike some other work in this field (e.g., Anderson, 1977; Power & Longuet-Higgins, 1978) and notwithstanding evidence that meanings facilitate language acquisition by humans (Campbell & Wales, 1970; Moeser & Bregman, 1972, 1973), the process is semantics-free. The reasons for this are a desire to preserve transparency and comprehensibility and the intrinsic interest of seeing what can be achieved without semantic information. It is perhaps more accurate to say that the process is semantics-indifferent: similar principles may be at work in the discovery of both linguistic and non-linguistic cognitive structures and in the interrelation of the two (Wolff, 1976). What are here called "elements" are similar to the notions of schemata, frames or scripts in that they can be used as a top-down guide to perceptual analysis (Wolff, 1975, 1978) and they can provide default values where parts of the data are missing or distorted (Wolff, 1976).
2. Only positive samples are used and there is no teacher. It is assumed that for the most part the same is true of children.
3. There are no markers of word, phrase or sentence segments in the data. The reason for this is the probability that children can discover the structure of language without the need for cues of this sort (Hayes & Clark, 1970).
4. The cognitive structures developed by the GDP are modular. Modularity is a desirable design feature where a cognitive system is to be built or modified (Winograd, 1975).

A working assumption is that discovery procedures in children are the product of selection pressures which are likely to have favoured economy in the storage and manipulation of information (Attneave, 1954; Oldfield, 1954). Consequently, it has been thought useful to view GDPs as a set of techniques for data compression. This supposition needs to be qualified by the probable relevance of a number of trade-offs, at least five. (1. Computational processes of sorting, compression and inference at input time may be reduced at a probable cost of additional processing required on retrieval (Bobrow, 1975). 2. Compression of data may impede subsequent manipulation and retrieval of data (Bobrow, 1975; Bobrow & Winograd, 1977). 3. Data compression which is theoretically possible may be unduly costly in computation. It may on occasion be advantageous to choose less costly methods of compression even if they do not achieve maximal compression. 4. It may be advantageous to preserve error-reducing redundancy. 5. There is a trade-off between complexity and discrepancy discussed by Cook & Rosenfeld (1976). If

one is prepared to allow distortions in the representation of the input data then greater degrees of compression become possible. Economies can be achieved by allowing grammars to generalise beyond the corpus on which they are based (e.g., by allowing the formation of recursive structures)).

Three data compression principles are employed in this GDP (see also Cook & Rosenfeld, 1976):-

1. Where a group of contiguous elements occurs more than once in a cognitive system then it is usually advantageous to record the structure of the group once and access it from varying contexts by means of pointers (Schuegraf & Heaps, 1974). A subroutine call in a computer program is a familiar example of this device.
2. Where two or more elements share one or more contexts then they may be grouped into a disjunctive set or element which may be accessed by pointers from varying contexts (a similar principle is described by Kang et al., 1977).
3. Where conflicting groupings in 1 or 2 are possible then it is advantageous to choose the grouping which will encode the greatest amount of the input corpus. In general, frequently occurring groupings are preferred to less frequent ones.

These principles are embodied in two programs: MK10H builds up frequently occurring strings of contiguous elements (known as syntagmatic elements or SYN-elements); GRAM15 operates on the resulting set of SYN-elements to find frequently recurring disjunctive sets of constituents (known as paradigmatic elements or PAR-elements) and replaces the original constituents each by pointer to the appropriate PAR-element. When applied to the texts produced by the simple phrase-structure grammars shown in Fig.1 these two programs create structures which are equivalent to the original grammars.

Figure 1. Grammars used to produce data for the GDP

Text 1
S->(1)PUT(2)ON|(1)MADE(2);(1)->JOAN|LIZ|YOU|HE;(2)->SOME|THEM|FOUR|IT;
Sample:YOUPUTITONHEPUTTHEMONHEMADEITLIZMADESOME...

Text 2
S->(1)(2)ES(3)|(4)(2)(3);(1)->IT|TOM|JANE|JOAN;(2)->MISS|WATCH;(3)->
BIRDS|LIZ|THEM|IT;(4)YOU|MEN|WE|THEY;Sample:TOMMISSESTHEMYOUWATCHBIRDS
THEYMISSLIZ...

Text 3
S->(1)(4)(5)|(6)(7)(8);(1)->(2)(3)|JOHN;(2)->A|THE;(3)->BOY|GIRL;(4)->
LIKES|ATE;(5)->FISH|MEAT;(6)->WE|THEY;(7)WALK|RUN;(8)->FAST|SLOWLY;
Sample:AGIRLATEFISHTHEYWALKFASTJOHNATEMEAT...

Text 4
S->(1)(2)(3)(4)|(5)(6)(7);(1)->BOB|MARY;(2)->LIKES|ATE;(3)->Ø|SOME;(4)->
FISH|MEAT;(5)->WE|THEY;(6)->WALK|RUN;(7)->FAST|SLOWLY;Sample:WEWALK
SLOWLYMARYATEFISHMARYLIKESSOMEMEAT...

Program MK10H, described elsewhere (Wolff, 1975, 1977, 1978; Collet & Wolff, 1977), begins its analysis with only the basic character set (minimal elements) in its dictionary of elements. It scans the data sample keeping a count of all contiguous pairs of elements and at the end of the scan it selects the most frequent pair, or makes an arbitrary choice amongst ties. (In a typical English text it would choose TH.) Then it adds the new element to its dictionary of elements.

The sample is scanned again, keeping a count of contiguous pairs as

before, but it treats every occurrence of TH as a unit. For example, a character sequence like ...INTHEG... would be segmented as ...I,N,TH,E,G... and counts would be made for I/N, N/TH, TH/E and E/G. After this second scan the pair of elements with the highest frequency (e.g., A/N or TH/E) is again selected and added to the dictionary. By repetition of this process a dictionary of frequently occurring character strings is built up, each having a binary hierarchical structure. At present there is no satisfactory stopping rule and the process is stopped when all the "sentences" in the artificial language sample have been isolated. It is expected that an integration of MK10H and GRAM15, on which work is proceeding, will eliminate this problem.

This program shows itself to be sensitive to the structure of the artificial texts used in that it builds up "words" and then joins them as coherent units to form "sentences" (Wolff, 1975). For example, from Text 1 it builds up PUT as (P(UT)), MADE as ((MA)(DE)), SOME as ((S(OM)E) and so on. At a later stage it forms (((LIZ)(PUT))((FOUR)(ON))), ((JOAN)((MADE)(SOME))) and all the other possible sentences. If it was not stopped it would join sentences together. More interestingly, the bracketing which it assigns to natural language texts largely honours the words in these texts, and the program produces a demonstrable compression of the data. It also seems to mimic certain features of children's vocabulary growth (Wolff, 1977).

GRAM15 operates on the dictionary of elements produced by MK10H in the following way. By a simple pattern matching procedure it first draws up a list of contexts for every element which is a constituent of one or more other elements. For example, the element LIZ, which is isolated from Text 1 by MK10H, is a constituent of LIZPUTFOURON, LIZMADEIT, LIZPUTSOMEON among other elements. Thus the contexts of LIZ include *PUTFOURON, *MADEIT and *PUTSOMEON. GRAM15 then makes a systematic comparison between every pair of context lists to find identical contexts and thus to find that disjunctive pair of elements which occurs most frequently in identical contexts. This disjunctive pair is added to the dictionary as a PAR-element and its reference number then replaces the original constituents in all appropriate SYN-elements. For example, the first such pair for Text 1 was (HE|LIZ) which was assigned reference number 94. Such elements as LIZPUTFOURON, LIZMADEIT, HEPUTFOURON, HEMADEIT, etc. become (94)PUTFOURON, (94)MADEIT, (94)PUTFOURON and (94)MADEIT respectively. The resulting duplicates are then deleted. The process is repeated exactly as described until no more matching contexts can be found. The process may be seen more clearly if we follow what happens after PAR-element 94 is formed. In the next cycle context lists are drawn up for all elements, <u>including</u> element 94, and they are systematically compared as before. It happens in this cycle that the pair of elements occurring most frequently in identical contexts is YOU and 94. These two elements are joined to form element 95 which thus has the implicit structure (YOU(HE,LIZ)). Element 95 then replaces YOU and 94 in all appropriate SYN-elements; duplicates are then deleted. In the following cycle element 96 is formed as (JOAN(YOU(HE,LIZ))). In the next three cycles PAR-elements 97, 98 and 99 are built up as (SOME,FOUR), (IT(SOME,FOUR)) and (THEM(IT(SOME,FOUR))) respectively. Eventually the sentences in the dictionary are reduced to two SYN-elements: (96)PUT(99)ON and (96)MADE(99).

Table 1 shows the state of the dictionary derived by MK10H from Text 1 after processing by GRAM15, with the omission of all SYN-elements which occurred only as constituents of larger elements. These constituent elements are "garbage" with no further function and may be discarded.

The same is true of PAR-elements 94, 95, 97 and 98 which are stepping stones in the formation of the functional PAR-elements, 96 and 99.

Table 1. The dictionary derived by MK10H form Text 1 after processing by GRAM15*

Reference Number	Composition
SYN-elements	
62	(96)PUT(99)ON
89	(96)MADE(99)
PAR-elements	
94	HE|LIZ
95	YOU|HE|LIZ
96	JOAN|YOU|HE|LIZ
97	SOME|FOUR
98	IT|SOME|FOUR
99	THEM|IT|SOME|FOUR

*SYN-elements which occur only as constituents of larger elements are omitted.

The dictionary at this stage has assumed essentially the same form as the grammar used to produce the original text. In a similar way MK10H/GRAM15 has proved successful in retrieving the grammars of Texts 2, 3 and 4. Notice that Text 2 has paradigmatic classes which overlap without being hierarchically related (the word IT is common to classes (1) and (3)). In Text 3 there is a SYN-element, [(2)(3)], which contains two PAR-elements and is itself a constituent of a PAR-element, (1), which is part of a SYN-element, [(1)(4)(5)]. In general there is no restriction on the types of constituent any element may have except that, at present, no element may contain itself as a constituent. Notice also from Text 4 that the GDP has provision for detecting a null element, a detail which has not been described in this short account.

The chief shortcoming of this GDP is its failure to generalise significantly beyond the given corpus. It does generalise trivially in that the grammars which are created may produce sentences in a sequence which need not be the same as in the sample corpus. But the PAR-elements are tied directly to the contexts from which they were derived so no sentences can be produced other than those which were in the corpus. Work is in progress at present on an integration of the syntagmatic and paradigmatic clustering processes with provision for generalisation, over-generalisation, correction of overgeneralisation and identification of recursive structures.

References

Anderson, J.R. (1977). Induction of augmented transition networks. Cognitive Science 1, 125-157.

Attneave, F. (1954). Some informational aspects of visual perception. Psychological Review 61, 183-193.

Bobrow, D.G. (1975). Dimensions of representation. In D.G. Bobrow and A. Collins (eds.) Representation and Understanding. New York: Academic Press.

Bobrow, D.G. & Winograd, T. (1977). An overview of KRL, a knowledge representation language. Cognitive Science 1, 3-46.

Campbell, R. & Wales, R. (1970). The study of language acquisition. In J. Lyons (ed.), New Horizons in Linguistics. Harmondsworth: Penguin.

Collet, R.J. & Wolff, J.G. (1977). From phoneme to morpheme - revisited. Bulletin of the Association for Literary and Linguistic Computing 5, 23-25.

Cook, C.M. & Rosenfeld, A. (1976). Some experiments in grammatical inference. In J.C. Simon (ed.) Computer Oriented Learning Processes. Leyden: Noordhoff.

Hayes, J.R. & Clark, H.H. (1970). Experiments on the segmentation of an artificial speech analogue. In J.R. Hayes (ed.), Cognition and the Development of Language. New York: Wiley.

Kang, A.N.C., Lee, R.C.T., Chang, C-L. & Chank, S-K. (1977). Storage reduction through minimal spanning trees and spanning forests. IEEE Transactions on Computers C-26(5), 425-434.

Moeser, S.D. & Bregman, A.S. (1972). The role of reference in the acquisition of a miniature artificial language. Journal of Verbal Learning and Verbal Behaviour 11, 759-769.

Moeser, J.D. & Bregman, A.S. (1973). Imagery and language acquisition. Journal of Verbal Learning and Verbal Behaviour 12, 91-98.

Oldfield, R.C. (1954). Memory mechanisms and the theory of schemata. British Journal of Psychology 45, 14-23.

Power, R.J.D. & Longuet-Higgins, H.C. (1978). Learning to count: a computational model of language acquisition. Proceedings of the Royal Society (London) 200, 391-417.

Schuegraf, E.J. & Heaps, H.S. (1974). A comparison of algorithms for data base compression by use of fragments as language elements. Information Storage and Retrieval 10, 309-319.

Winograd, T. (1975). Frame representations and the declarative - procedural controversy. In D.G. Bobrow and A. Collins (eds.), Representation and Understanding. New York: Academic Press.

Wolff, J.G. (1975). An algorithm for the segmentation of an artificial language analogue. British Journal of Psychology 66, 79-90.

Wolff, J.G. (1976). Frequency, conceptual structure and pattern recognition. British Journal of Psychology 67, 377-390.

Wolff, J.G. (1977). The discovery of segments in natural language. British Journal of Psychology 68, 97-106.

Wolff, J.G. (1978). Recoding of natural language for economy of transmission or storage. Computer Journal 21, 42-44.

Wolff, J.G. (in press). The discovery of syntagmatic and paradigmatic classes. Forthcoming in the Bulletin of the Association for Literacy and Linguistic Computing.