

*PROCEEDINGS OF  
THE AISB-80 CONFERENCE ON*

# *Artificial Intelligence*

*AMSTERDAM*

*1st to 4th July, 1980*

*a1sb-80*

Conference Proceedings

AMSTERDAM

1st to 4th July, 1980

*Programme Chairman:*  
Steven Hardy

*General Chairman:*  
Bob Wielinga

*Programme Committee:*  
Mike Brady (MIT)  
Steven Hardy (Sussex)  
Joerg Siekmann (Karlsruhe)  
Karen Sparck-Jones (Cambridge)  
Bob Wielinga (Amsterdam)  
Richard Young (Cambridge)

*Sponsored by:*  
The Society for the Study of  
Artificial Intelligence and  
the Simulation of Behaviour

# REFEREES

=====

The following is a list of people who offered to referee papers for AISB-80. The Programme Committee was greatly helped in its difficult task of selecting papers for the conference by the comments the referees made.

A. Adam	J. Howe
L. Aiello	S. Isard
A.P. Ambler	M. King
B. Anderson	Y. Kodratoff
W. Bibel	J. Laubsch
A. Bond	C. Longuet-Higgins
A. Borning	C.S. Mellish
M. Brady	R. Milne
M.J. Brooks	H. Nagel
A. Bundy	T. O'Shea
J.A. Campbell	G. Plotkin
M.R.B. Clarke	T. Poggio
B. Clocksin	R. Popplestone
M. Clowes	G. Prini
M. Colombetti	A. Ramsay
J.L. Darlington	P. Raulefs
D. De-Champeaux	G. Ritchie
J. Doran	M. Rosner
S. Draper	E. Sandewall
M. Eisenstadt	C. Schwind
J. Foith	D. Sleeman
G. Gazdar	A. Sloman
G. Gini	M. Somalvico
M. Gini	S. Sutherland
G. Guida	Y. Wilks
K. Hanna	G. Wrightson

-----

Enquiries about the purchase of this volume, or the proceedings of the three previous AISB conferences, should be addresses to:

Dr Steven Hardy  
Cognitive Studies Programme  
University of Sussex  
Falmer, Brighton  
ENGLAND, BN1 9QN  
(Telephone: Brighton (0273) 606755 ext 1002)

## CONTENTS-1

- A. Adam, & J. Laurent, Caen FRANCE,  
Automatic Diagnostics of Semantic Errors.
- W. Bibel, Munchen, WEST GERMANY,  
A Comparative Study of Several Proof Procedures.
- D.G. Bobrow, & I.P. Goldstein, Palo Alto, UNITED STATES of AMERICA,  
Representing Design Alternatives.
- M.A. Bramer, Milton Keynes, ENGLAND,  
Pattern Based Representations of Knowledge:  
In the Game of Chess.
- L. Byrd, & A. Borning, Edinburgh, SCOTLAND  
Extending MECMO to Solve Statics Problems.
- A. Cappelli, G. Ferrari, et al. Pisa, ITALY  
Automatic Analysis of Italian
- A.W.S. Cater, Cambridge, ENGLAND,  
Analysing English Text: A Non-Deterministic Approach  
with Limited Memory.
- S. Cerri and J. Breuker, Pisa, ITALY,  
A Rather Intelligent Language Teacher.
- W. Clocksin, Edinburgh, SCOTLAND,  
The Effect of Motion Contrast on Surface Slant  
and Edge Detection.
- S. Draper, Sussex, ENGLAND,  
Using Models to Augment Rule-Based Programs.
- H. Eigemeier, Ch. Knabe, et al. Bonn, W. GERMANY  
Automatic Implementation of Algebraic Specifications  
of Abstract Data Types.
- M. Eisenstadt, & J. Laubsch, Milton Keynes, ENGLAND  
Towards an Automated Debugging Assistant for  
Novice Programmers.
- N. Eisinger, J. Siekmann, et al. Karlsruhe, WEST GERMANY  
The Markgraf Karl Refutation Procedure.
- E.J. Fidler, Vancouver, CANADA,  
Decision Rules for Binary Choices:  
A Process-Tracing Study.
- P. Greussay, Paris, FRANCE,  
Program Understanding by Reduction Sets.
- G. Hagert, Stockholm, SWEDEN,  
Stability and Change Strategies: Three Simulations  
of One Subject with One-Year Intervals



CONTENTS-2

- E. Hoenkamp, Nijmegen, NETHERLANDS,  
Spontaneous Speech as a Feedback Process.
- K.M. Kahn, Stockholm, SWEDEN,  
How to Program a Society.
- Y. Kodratoff and E. Papon, Paris, FRANCE,  
A System for Program Synthesis and Program Optimization.
- G.F. Luger, New Mexico, UNITED STATES of AMERICA,  
Means Ends Analysis and  
the Solution of Mechanics Problems.
- J. Mayhew & J. Frisby, Sheffield, ENGLAND.  
Computational and Psychophysical Studies  
Towards a Theory of Human Stereopsis
- C.S. Mellish, Edinburgh, SCOTLAND,  
Some Problems in Early Noun Phrase Interpretation.
- R. Milne, Edinburgh, SCOTLAND,  
Using Determinism to Predict Garden Paths.
- S. Ohlsson, Stockholm, SWEDEN,  
Strategy Grammars: An Approach to Generality  
in Computer Simulation of Human Reasoning.
- D. Owen, Sussex, ENGLAND,  
Intermediate Descriptions in POPEYE.
- A. Ramsay, Edinburgh, SCOTLAND,  
Parsing English Text.
- A. Ramsay, Edinburgh, SCOTLAND,  
Understanding English Description of Programs.
- P. Schefe, Hamburg, WEST GERMANY,  
The Fuzzy Set Fallacy.
- A. Sloman, & D. Owen, Sussex, ENGLAND  
Why Visual Systems Process Sketches.
- N.S. Sridharan, C.F. Schmidt, & J.L. Goodson, New Brunswick, U.S.A.  
The Role of World Knowledge in Planning.
- M. Steedman & A. Ades, Warwick, ENGLAND,  
An Algorithmic Account of  
English Main Clause Constructions.
- L. Steels, Ridgefield, UNITED STATES of AMERICA,  
Description Types in the EXPRT-System.
- G. Sussman, J. Holloway, & T.F. Knight, Jr. MIT. U.S.A.  
Computer Aided Evolutionary Design for  
Digital Integrated Systems

CONTENTS-3

- J.G. Wolff, Dundee, SCOTLAND,  
Data Compression, Generalisation and Overgeneralisation  
in an Evolving Theory of Language Development.
- G. Wrightson, Karlsruhe, W. GERMANY  
Avoiding Equivalence Explosions in Theorem Proving -  
Propositional Logic

AUTOMATIC DIAGNOSTICS OF SEMANTIC ERRORS

By Anne ADAM and Jean-Pierre LAURENT

Members of G.R. 22 of C.N.R.S.

Maitre-Assistants at the University of CAEN,

Laboratoire d'Informatique Heuristique

UNIVERSITE DE CAEN, 14032 CAEN Cedex, FRANCE

ABSTRACT

We present in this paper an original approach of Debugging. The general goal in Automatic Verification of Programs is to prove that a program is correct or incorrect. This does not generally provide enough information to catch the bugs. On the opposite, the purpose of the debugging system LAURA, that we have designed and implemented, is to find the errors.

In order to debug a student program, the LAURA system uses a procedural description of the program task, under the form of a *program model*. Debugging is then viewed as a comparison of two graphs, built from the student program and from the program model. The system can apply powerful semantic transformations on the graphs to increase their resemblances and to identify subgraphs that perform a same task.

The LAURA system has shown to be able to determine the correctness of programs implemented in various ways, very different from the program model. It can also express sophisticated diagnostics and set up proper corrections.

Keys words : Debugging, Program Transformations, Graphs of Program, Diagnostics of errors.

INTRODUCTION :

The numerous studies that have been made about the correctness of programs try only to prove whether a program is correct or not. If it is incorrect, no information is given about the errors that effectively exist. So, these methods cannot help us in Debugging.

Thus, we have designed and implemented a system called LAURA, whose first object is not to prove the correctness of a program, but to detect or localize the errors it may contain.

The programs that LAURA deals with have been written by students that are programming apprentices. These programs are compiled before they are submitted to the system and syntactic errors have been eliminated. The system only looks for the semantic errors, which prevent the program from giving the expected results.

I. AN ORIGINAL APPROACH OF THE DEBUGGING PROBLEM

In order to find all the semantic errors that the given program contains, it is essential to have some knowledge of the problem that this program is supposed to solve.

The description of the program's task consists in sets of "assertions" (an assertion is a property about the values of some variables, which has to be true in a given point of the program). The output results that the program intends to achieve are expressed by the mean of one set of output assertions. One set of input assertions corresponds to the data properties. Then it must be proven that the program holds through from input assertions to output assertions. FLOYD [1967], NAUR and HOARE [1971] were the first to consider and to formalize this method that has since been used by many researchers, e.g. ASHCROFT [1975], KATZ and MANNA [1976], LEVITT and WALDINGER [1974].

We do not describe in this paper the classic program verification methods that use assertions. But we must briefly mention the main objections that may be made to the assertions methods and that have led us to try another approach. Firstly, it is very difficult to give complete sets of assertions that describe the problem entirely. Secondly, the introduction of "invariants" (internal assertions given by the programmer, which are necessary to separate the different paths) quickly becomes a problem more difficult than the writing of correct programs. Thirdly, the automatic detection of invariants and the demonstrations of theorems that must be made, are so complex that they cannot be executed by

an automatic system (at least considering the actual power of automatic theorem provers). Last but not least, the methods based on assertions give a boolean answer : the program is correct or not. If not, they do not make any constructive criticism to show what errors have been made. So they are not really adequate for debugging.

We have selected another way of giving the knowledge of the program task to the system. Instead of describing what the program has to achieve, we give informations about how it must proceed. In other terms we give the system the algorithm that the program must use to attain its goals rather than the goals themselves. For that, we use a "program model" which is supposed to be a correct implementation of this algorithm. Thus the LAURA system receives two programs, the program model and a student program, that should be two implementations of the same algorithm. In order to debug the student program, the system has to compare it with the program model. Automatic debugging is then viewed as a comparison of two programs.

In order to simplify further matching, the first purpose of the LAURA system is to make some normalizations.

The system translates each program into a graph. This choice is very fundamental. The vertices of the graph represent the essential actions (assignments, tests, inputs or outputs) and not instructions in the used programming language. The arcs represent the partial order relation between these actions. This transformation of a program into a graph is a first and very important standardization. As a matter of fact, the graph is a representation of the calculus process that a program implies. This representation gets rid of particularities coming from syntactical choices inside the used language. Furthermore it is independent of the language. So it will be possible to deal with student programs written in other languages than that of the program model. Each graph obtained from a program represents a large class of programs which imply the same set of calculus. Many syntactical choices in each language are not taken into account in the graph. (In particular, as the graph is non-linear the labels have disappeared).

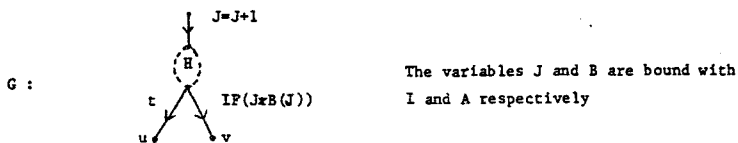
Furthermore, LAURA systematically applies transformations to each graph in order to extend the standardization. For example if the same variable is used for two different purposes, a second variable is generated. Also, if an internal variable is only used to store an intermediary result, useful for several computations, this variable is suppressed. It is interesting to note that standardizing transformations are generally deoptimizing transformations.

When standardizations are finished, the LAURA system has to match the two graphs. It tries to bind variables, vertices and arcs. For this, it can work step by step around a pair of already identified vertices : it tries to bind their immediate successors (respectively their predecessors) then the successors of the successors, etc...

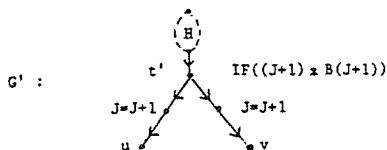
When such an exploration does not give any more result, the system can consider another pair of unidentified vertices and match them. In order to select this pair it uses a function which heuristically evaluates the plausibility for two given vertices to be corresponding ones. The representation of programs by graphs makes possible such a non-linear strategy, that could not be used on ordered sets of instructions in a source language.

Also, the system can use heuristics to apply powerful semantic transformations to the graphs, in order to make two subgraphs which previously had different structures become similar. The system is able to perform splittings, permutations and test-crossings (see Fig 1). It can alter the structure of loops. It can also solve recurrence equations and then, some loops may be replaced by a single formula. (see Fig 2).

In the first graph, there is the vertex  $s$  :  $IF((I+1) \neq A(I+1))$  and in the second graph, there is the subgraph  $G$  :

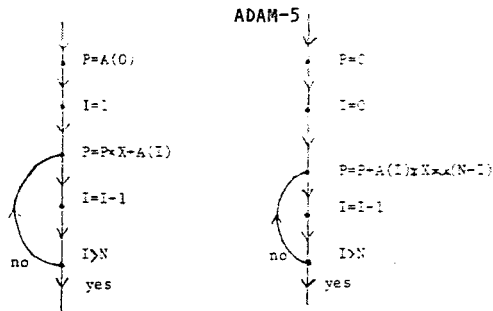


If some conditions about the subgraph  $H$  are verified, the LAURA system can transform  $G$  into :



and  $t'$  may be bound with  $s$

FIG 1 : EXAMPLE OF GRAPH TRANSFORMATION



From each one of these two subgraphs, the LAURA system is able to extract the same formula :

$$P = \sum_{I=0}^N A(I) \times X^{N-I}$$

FIG 2 : EXAMPLE OF RECURRENCE SOLVING

If total identification is possible, the student program is found to be correct (meaning that it computes the same functions as the model and produces the same results besides round-off-errors).

If only partial identification is realized, the system tries to analyse the remaining differences in order to give exact diagnostics and to set up proper corrections. The main diagnostics that LAURA can express are :

- error of variable
- error of constant
- unary connective forgotten
- useless unary connective
- sign error
- branching error
- inversion of two instructions (which are not permutable)
- test error (conditions on arcs do not correspond)

If some subgraphs still remain unidentified, they are printed by the system. In this way the user knows in which part of this implementation a doubt remains. If this part contains a semantic mistake, it has not been pointed out by the system but it has been localized, which is quite a big help in debugging.

The diagram of Fig. 3 shows the different phases of the LAURA system.

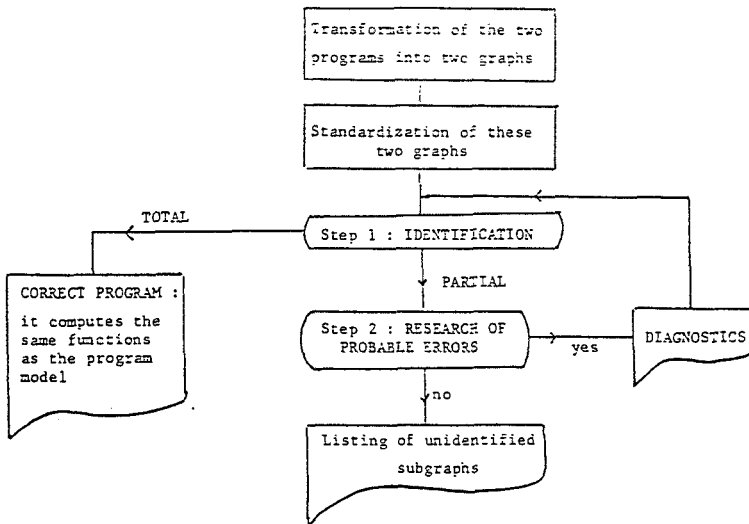


FIG 3 : THE LAURA DIAGRAM

## II. RESULTS

The LAURA system has been tested on about a hundred programs written by students to solve eight different problems in various fields :

- management (tax computation, electric company invoices)
- arithmetic (perfect numbers, Pascal triangle)
- numerical analysis methods (integration, equation roots)
- sorting of an array A of N numbers (using a given algorithm : looking for the maximum and permutation)

Numerous examples are given and commented in [LAURENT, 1978]. We give two of them in this paper.



Example 1 : Text of the exercise :

Find the real roots of the equation  $ax^2 + bx + c = 0$ .

Print their number and their values.

Program model :

```

0 READ(05,11)A,B,C
11 FORMAT(3E12,4)
0 IF(A)51,50,51
51 D=B**2-4*A*C
0 IF(D)1,2,3
1 N=0
70 WRITE(06,12)N
12 FORMAT(I5)
0 GO TO 100
2 X0=-B/(2*A)
20 N=1
0 WRITE(06,13)N,X0
13 FORMAT(I5,E12,4)
0 GO TO 100
3 X1=(-B-SQRT(D))/(2*A)
0 X2=(-B+SQRT(D))/(2*A)
0 N=2
0 WRITE(06,14)N,X1,X2
14 FORMAT(I5,2E12,4)
0 GO TO 100
50 IF(B)61,60,61
61 X0=-C/B
0 GO TO 20
60 IF(C)1,62,1
62 N=-1
0 GO TO 70
100 STOP
0 END

```

Student program :

```

0 READ(05,100)A,B,C
0 IF(A.EQ.0)GOTO 1
0 D=B**2-4*A*C
0 IF(D)2,3,4
4 NR=2
0 D=SQRT(D)
0 R1=(-B+D)/(2*A)
0 R2=(-B-D)/(2*A)
0 WRITE(06,200)NR,R1,R2
0 STOP
3 R1=-B/(2*A)
7 NR=1
0 WRITE(06,300)NR,R1
0 STOP
2 NR=0
12 WRITE(06,400)NR
1 IF(B)5,6,5
6 R1=-B/A
0 GO TO 7
5 IF(C)2,8,2
8 NR=-1
0 GOTO 12
100 FORMAT(3E12,4)
200 FORMAT(I5)
300 FORMAT(I5,E12,4)
400 FORMAT(I5,2E12,4)
0 STOP
0 END

```

The final states of the two graphs correspond to the two following programs that the system generates at the end of the run :

PROGRAMME DE REFERENCE

```

1 READ A
2 READ B
3 READ C
4 IF(A)06,19,06
19 IF(B)20,21,20
21 IF(C)07,22,07
22 N=-1
8 PRINT N
23 STOP
7 N=0
GOTO 8
20 X0=-C/B
11 PRINT 1
12 PRINT X0
GOTO 23
6 IF(B**2.+A*C*(-4.))07,09,14
14 X2=((B**2.+A*C*(-4.))*0.5-B)/A*0.5
13 X1=(-(B**2.+A*C*(-4.))*0.5-B)/A*0.5
16 PRINT 2
18 PRINT X2
17 PRINT X1
GOTO 23
9 X0=B/A*(-0.5)
GOTO 11
END

```

.PROGRAMME ETUDIANT REECRIT PAR LE SYSTEME

```

101 READ A
102 READ B
103 READ C
104 IF(A)106,121,106
121 IF(B)123,122,123
122 R1=-B/A
117 PRINT 1
118 PRINT R1
114 STOP
123 IF(C)119,124,119
124 NR=-1
120 PRINT NR
GOTO 121
119 NR=0
GOTO 120
106 IF(B**2.+A*C*(-4.))119,115,109
109 W1R1=((B**2.+A*C*(-4.))*0.5-B)/A*0.5
110 R2=(-(B**2.+A*C*(-4.))*0.5-B)/A*0.5
111 PRINT 2
112 PRINT W1R1
113 PRINT R2
GOTO 114
115 R1=B/A*(-0.5)
GOTO 117
END

```

# DIAGNOSTICS

```

INSTRUCTION 20 DANS PROGRAMME 1 NON IDENTIFIEE }
INSTRUCTION 122 DANS PROGRAMME 2 NON IDENTIFIEE } (1)

ERREUR DE BRANCHEMENT PROBABLE : ARC(120,121) AU LIEU DE ARC(120,114) (2)
CONDITIONS DIFFERENTES SUR LES ARCS ISSUS DES INSTRUCTIONS 19 ET 121 (3)

```

COMPILE TIME= 3.03 SEC.EXECUTION TIME= 7.55 SEC.

- (1) R1 = -B/A instead of R1 = -C/B
- (2) a stop has been forgotten
- (3) labels 122 and 123 are interchanged

## Example 2 : text of the exercise :

"A perfect number is a positive integer k which is equal to the sum of its divisors, 1 included and not k. Print each perfect number less than or equal to 1000".

## Example 2 :

### Program model :

```

C  0 NOMBRES PARFAITS - CORRIGE
  0 DO 100 I=6,1000
  0 IS=1
  0 K=2
  1 IF(I/K*K.EQ.I)IS=IS+K+I/K
  0 K=K+1
  0 IF(K*K.LT.I)GOTO 1
 100 IF(IS.EQ.I)WRITE(08,10)I
 10 FORMAT(15)
  0 STOP
  0 END

```

### Student program :

```

C  0 NOMBRES PARFAITS - ETUDIANT 1
  0 N=6
  2 I=2
  0 L=1
  1 IF(N-(N/I)*I)10,20,10
 20 L=L+I+N/I
 10 IF((I+1)**2-N)30,40,40
 30 I=I+1
  0 GOTO 1
 40 IF(L-N)50,60,50
 60 WRITE(06,55)N
 55 FORMAT(I4)
 50 IF(N-1000)70,100,100
 70 N=N+1
  0 GOTO 2
100 STOP
  0 END

```

# DIAGNOSTICS

PROGRAMME CORRECT : IL CALCULE LES MEMES FONCTIONS QUE LE PROGRAMME DE REFERENCE

COMPILE TIME= 3.17 SEC.EXECUTION TIME= 5.04 SEC. (IBM 370-168)

In spite of many syntactical and structural differences, the LAURA system has established that the student program is equivalent to the program model.

CONCLUSION

The LAURA system is able to recognize correct programs even if their structures are very different from the structure of the program model. It is also able to express exact diagnostics of errors, or at least to localize the errors. Thus the LAURA system may be a great help in debugging programs.

In particular, it could be an effective tool for student programmers.

We think that three points are essential to explain the good performances that the system has obtained :

- representation of programs by graphs, which gets rid of many syntactical choices.
- use of program transformations, realized on the graphs.
- heuristic strategy to identify step by step the elements of the graphs, to make suitable transformations, to overlook more and more complex differences which probably correspond to semantic local errors.

It should be observed that automatic interpretation of differences is far from obvious. As a matter of fact, a difference may result from an error but it may also result from the use of a variation in the required algorithm. This variation may have no real importance. It may also be an awkwardness or on the contrary a skilful optimization. To discover it, the system should have, besides the knowledge of the task the program has to perform, a great knowledge of the field in which this task has a meaning. So we must choose between two possibilities. The first one is to give a good knowledge of a narrow specialized field and to debug programs in this field only. The second is to debug programs in various fields and to let the user himself make some difficult interpretations. This dilemma is actually a limit for automatic debugging. In our system the second solution has been preferred.

REFERENCES

- A. ADAM : Utilisation des transformations sémantiques pour la correction automatique des programmes. Thèse d'Etat. PARIS VI Nov. 1978.
- J. ARSAC : Syntactic source to source transforms and program manipulation. Comm. of A.C.M. Janv. 1979.
- E. ASHCROFT : Program proving without tears. Colloque IRIA. Arc et Senans July 75.
- T.E. CHEATHAM and others : Symbolic evaluation and the analysis of programs. IEEE Transactions on Software Engineering. July 1979.
- L.P. DEUTSCH : An interactive program verifier. Ph. D. Thesis. Univ. of California at Berkeley. June 1973.
- R.W. FLOYD : Assigning meanings to programs. Proc. Symp. in Applied Math. A.M.S. vol 19 1967.
- C. HEWITT and B. SMITH : Towards a programming apprentice. IEEE Trans. of Software Engineering VI, 10, March 1975.
- C.A.R. HOARE. Proof of a program FIND. Comm. of A.C.M. vol 14, 1 - 1971.
- S.M. KATZ and Z. MANNA : Logical Analysis of programs. Comm. of A.C.M. vol. 19, 4 - 1976.
- J.P. LAURENT : Un système qui met en évidence des erreurs sémantiques dans les programmes. Thèse d'Etat PARIS VI Nov. 1978.
- L.J. OSTERWELL and L.D. FOSDICK : DAVE, a validation, error detection and documentation system for FORTRAN. Software - Practice and Experience 6, Sept. 1976.
- G.R. RUTH : Intelligent program analysis. Artificial Intelligence Vol. 5, 3 1974.
- G.J. SUSSMAN : A computational model of skill acquisition. Report AI-TR 297, MIT 1973.
- R.L. WALDINGER and K.N. LEVITT : Reasoning about programs. Artificial Intelligence Vol 5, 3 1974.
- R.C. WATERS : Automatic Analysis of the logical structure of programs. Report AI-TR 492 M.I.T.

A COMPARATIVE STUDY OF SEVERAL PROOF PROCEDURES

Wolfgang Bibel  
 Institut für Informatik  
 Technische Universität  
 München, Germany

*Abstract.* In this paper three algorithms for testing the complementarity of a matrix (representing a propositional formula) are developed in stages. Any of these algorithms is distinguished from its predecessor by a specific feature (linearity, jump, non-normal form) which endows it with a provable advantage w.r.t. its performance. For well-known proof procedures it is shown that they can be simulated by at least one of these algorithms.

Introduction

Measuring the relative performance of automated theorem proving (ATP) methods is a highly desirable but complicated and thankless task. Given two such methods, the most promising approach to such measurement seems to be an attempt to gain a deeper insight into their respective nature and hopefully to find a common basis on which their shared and differing features become apparent, and thus capable of measurement.

Theorem proving can be viewed as the problem of verifying that each path through a matrix (i.e. a set of clauses) is complementary in the sense that it contains a literal and its complement. This view, was first explicitly adopted by Prawitz in [12], later by the author in [9], and recently by Andrews in [1;2]. It has been formalized for the ground level in [6]. This formalization has already been used in [7] to provide a more profound insight into the nature of resolution. And in fact it is this very insight which will provide the basis for comparison in the present paper. Thus this study should be considered as a direct continuation of that in [7].

In detail we will proceed as follows. In section 1 we begin with a basic path-testing algorithm A1 of the nature mentioned above. Its faults are obvious therefore it will be immediately improved to provide a slightly more sophisticated one A2 which differs from A1 by what is called the *linearity* feature. The improvement is studied in quantitative terms. In section 2 it is shown that the well-known linear methods such as model elimination or SL-resolution [11] can be simulated by A2 which at the same time has a smaller search space than those methods. In section 3 an obvious redundancy is removed from A2 by adding what is called the *jump* feature, thus obtaining an algorithm A3. It is conjectured that A3 can simulate the connection graph procedure [10]. Finally, in section 4 a different redundancy is removed from A2 by implementing what is called the *non-normal form* feature, thus obtaining an algorithm A4 which operates on completely general matrices. It is shown that the refutation graph method [13] can be simulated by A4. In each of these cases "simulation" is to be understood without any increased time or storage requirements.

It should be not too difficult to combine the advantages of A3 and A4 in a future algorithm potentially more powerful than all methods mentioned above and many others with them.

1. Basic algorithms testing complementarity

A propositional formula is valid (or inconsistent) iff its matrix is complementary, i.e. each path through the matrix contains a literal L and its negated form  $\bar{L}$ . Therefore the simplest algorithm, which tests whether a matrix is complementary, is the following.

1.1.A.\*) For each path  $p$  in the given matrix  $M$  do

```

  DONE ← ∅; 1: choose L,K such that L,K ∈ p and (L,K) ∉ DONE;
  if L ≠ K then DONE ← DONE ∪ (L,K); goto 1

```

The algorithms PAIR within the system which has been described in [9], its predecessor in [3], the algorithm (15) from [5], and the process described in section III of [1] are exactly of such a nature. Note that the matrix is not required to be in any normal form. However, from now on until section 4 the discussion will be restricted to matrices in normal form. For those a deterministic version requires only a few lines as follows.

1.2.A. (It is assumed that the given matrix and the active path is represented as a binary array  $M[i,j]$  of literals and a unary array  $P[i]$  of integers, resp.,  $1 \leq j \leq m$ ,  $1 \leq i \leq n$ )

```

for i:=1 until m do P[i]:=1;
1: for i:=1 until m do for j:= i+1 until m do
  if M[i,P[i]] = M[j,P[j]] then goto 2; return ("complementary");
2: k:=1;
3: if P[k] < mk then P[k]:=P[k]+1; goto 1; P[k]:=1;
  if k < m then k:=k+1; goto 3 else return ("non-complementary");

```

The simple matrix  $\begin{smallmatrix} L & K & \bar{L} & \bar{N} \\ N & K & & \end{smallmatrix}$  may exhibit an obvious drawback of this kind of algorithm. (1.2) compares  $L$  with  $K$  and  $L$  with  $\bar{L}$  in the first path  $\{L,K,\bar{L},\bar{N}\}$ ,  $L$  with  $N$  and  $L$  with  $\bar{L}$  in the second path, etc.; altogether it performs 13 comparisons in the four different paths. A different sequence of comparisons would require only 3 comparisons ( $L-\bar{L}, K-N, \bar{N}-\bar{N}$ ) for testing complementarity. There are two reasons for the striking difference. First, these algorithms do not benefit from the fact that a pair of literals may prove *more than one* path complementary. Second, the search for complementary literals is performed in a completely blind way.

It is straightforward to overcome these disadvantages by adjusting the choice of paths and literals to the given matrix in a flexible way. To that effect we may assume that for a given literal  $L$  an occurrence of  $\bar{L}$  in the matrix can be determined in one step. This can be realized by adding the set of connections, or, even simpler, by establishing a list in which for each literal all occurrences in the matrix are noted which increases the amount  $\sum |C|$  of memory required for storing the matrix  $M$  by a factor 2. C&M

1.3.A. (The algorithm is presented in two versions. The *preliminary* version - to be discussed in this and the next section - is obtained from the following text by cancelling everything embraced by brackets [...]. The *full* version - to be discussed in section 3 - is obtained by deletion of the brackets. Both versions test whether a matrix  $M$  in normal form is complementary. In the list variable ACT the list of literals of that part of the active path is stored which have been considered so far; WAIT denotes a stack on which literals to be considered later, together with the values of ACT and  $M$  relevant for them, are stored; push(W,L) means that the value of  $L$  is stored on top of stack  $W$ ; conversely, pop(W) has as its value the top element of stack  $W$ , which is removed from  $W$  except in a call within a boolean expression - like in step 15 below. "dm" abbreviates "ditch-marker".)

---

\*) A., T. abbreviate "algorithm" and "theorem", resp.

```

1: ACT ← ∅; while non-empty(WAIT) do pop(WAIT); [i ← 0;]
2: choose a clause C from the matrix M; M ← M - C;
3: choose a literal L from C;
   if C \ L ≠ ∅ then push(WAIT, (C \ L, ACT, M));
   [i ← i + 1;] ACT ← ACT ∪ {[L, i, 0]};
4: if M = ∅ then return ("non-complementary");
5: if there is no clause C ∈ M such that L ∈ C then
6: "if there is no clause C ∈ M such that
   K ∈ C for some [({K[j, b]}) ∈ ACT then goto 1
7: else "choose C from M such that
   K ∈ C for some [({K[j, b]}) ∈ ACT;
   [push(WAIT, "dm"); dm ← i;]";
8: else choose C from M such that L ∈ C;
9: M ← M - C; C ← C \ L;
10: for all literals K such that
   K ∈ C and [({K[j, b]}) ∈ ACT [for some j and b] do
   "C ← C - K; [if j ≤ dm then replace b by 1 in (K, j, b) ∈ ACT;]"
11: if C ≠ ∅ then goto 3;
12: if empty(WAIT) then return ("complementary");
13: [if pop(WAIT) = "dm" then
   "pop(WAIT);
   while pop(WAIT) = (C', ACT', L', i', 0), M')
   for some C', ACT', L', i', M' and i' ≥ max(0, j)
   where j = max{k | (K, k, 1) ∈ ACT for some N or k = 0}";
   do pop(WAIT); goto 12;]
   (C, ACT, M) ← pop(WAIT); [if ACT = ACT' ∪ (L', i', b') for some ACT', L', i', b',
   then i ← i' else i ← 0;] goto 3;

```

Until section 3 the discussion is restricted to the preliminary version of (1.3). Let us consider our previous example and assume that  $K$  will be chosen in the initializing step while leaving  $N$  at the bottom of  $WAIT$ . In the first comparative step the single  $K$  will be selected as illustrated in figure 1 (full line from  $K$  to  $K$ ). This step proves all paths comple-

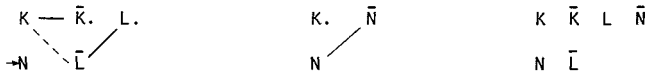


Figure 1. Two snapshots and the whole matrix

mentary which contain this pair  $(K, \bar{K})$ , and none of these paths will be considered again.  $K$  is now stored in  $ACT$ . All remaining paths which pass through  $K$  must pass through  $L$  (dotted line from  $K$  to  $L$ ) which in this case is the only other literal to be chosen during this step. Since  $L$  is the only literal in the clause chosen in the second step (full line from  $L$  to  $L$ ), the process may now turn to the next literal on the stack. Figure 1 illustrates this situation in the first snapshot. All paths containing  $K$  have turned out to be complementary at this moment while all paths containing  $N$  are yet to be considered. But this in fact will be completed in one further step as illustrated in the second snapshot in figure 1.

The correctness and completeness of this algorithm (in its preliminary version) is obvious on the basis of the definition of *complementary matrices*. The only point worth to be mentioned in this respect is the case where for the chosen literal  $L$  no complement can be found in any remaining clause (steps 5-7) like in  $L \text{ --- } \bar{L} \text{ --- } N \text{ --- } \bar{N} \text{ --- } T$ . When the algorithm arrives at the encircled  $N$  this case occurs. But obviously, any literal which has a com-

plement in the active path ACT may play the role of the missing  $\bar{N}$  (actually, this could have been allowed in the algorithm as an alternative choice even in the regular case where  $C$  with  $\bar{L} \in C$  exists). If not even such a literal does exist in any of the remaining clauses then the *whole* matrix is complementary if and only if this holds for the *remaining* matrix, since by construction any path which has ACT as a subpath must contain a complementary pair of literals in the remaining matrix.

We notice the following two virtues of algorithm (1.3). Firstly, within one step, for any pair of literals  $(L, \bar{L})$  currently being considered it proves the complementarity of *all* paths passing through this pair and it never considers these paths again. Secondly, in the regular case (step 8) it never requires more than one step (i.e. one comparison) to prove that a path is complementary, since the choice of complementary pairs of literals is the leading action (not the blind choice of paths as in the previous algorithm). For the steps 7 and 10 the number of comparisons is bounded by the number of elements in ACT which in turn is bounded by  $m := |M|$ . Therefore, (1.3) is faster than any version of (1.1) for a normal form matrix. The speed-up is in fact considerable since, assuming a uniform distribution of literals within the matrix), the number of comparisons *per path* on the average is quadratic in  $m$  for (1.1) while it is less than  $m$  for (1.3). Under the same assumption the number of paths is  $O(n^m)$  where  $n$  is the number of different variables in the matrix which shows that the improvement renders an exponential speed-up.

The price for this improvement is on the memory side, but it is very cheap. If one shares the information within the stack - note that the values for ACT and M build upon those of the previous entry, and that a marker suffices for  $C$  - then we need at most  $3 \cdot m$  auxiliary locations (in addition to the factor 2 mentioned before 1.3). The following theorem summarizes these arguments.

1.4.T. Algorithm (1.3) in its preliminary version is strictly faster than (1.1) for a matrix in normal form, the speed-up being exponential in the number  $m$  of clauses in  $M$  (for a uniform distribution of literals). On the other hand, (1.3) requires at most  $\sum_{C \in M} |C| + 3m$  more memory locations than (1.1).

## 2. Linear resolution methods

Figure 1 has illustrated one key idea which distinguishes algorithm (1.3) - still in its preliminary version - from (1.1) and which will be called the *linearity feature*: the pairs of complementary literals are chained in a linear way (whenever the condition in step 5 is false). This is exactly the feature which characterizes all linear resolution methods. In order to illustrate that let us present the actions of (1.3) for the example of figure 1 in a modified notation which clearly reflects exactly the same steps.

1.  $N$  given
2.  $\bar{L}$  given
3.  $KL$  given
4.  $KN$  given
5.  $N[K]\bar{L}$  extension with 3.
6.  $N$  extension with 2.; bracketed literals deleted (truncation)
7.  $\square$  extension with 1.; bracketed literal deleted

The reader familiar with model elimination (ME) will immediately see that this is a correct ME refutation (see [11] for terminology). ME is just a variant of the class of linear resolution procedures. Hence, there seems to be a close relation between linear resolution and (1.3). This relation will be now explored in precise terms.



(1.3) contains no action which corresponds to merging (or factorization) in ME. As we will see in section 4, factorization is a separate issue which has been solved only in an incomplete way in all linear methods; therefore its implementation is postponed.

In (1.3) the information given by  $M$  prevents the algorithm from extending a path a second time through the same clause. Of course, this provision could be deleted without touching completeness or consistency, since addition of a second copy of a clause to a given matrix apparently does not affect complementarity at all. But it certainly would affect efficiency in a bad way since the number of paths apparently increases with each such copy. Hence this additional feature of (1.3) turns out to be a special virtue which is implemented in ME by the restriction of a deduction to contain *acceptable* chains only. Since without factoring the notion of acceptability would have to be adapted, we rather implement the information given by  $M$  by attaching the corresponding clause numbers to the A-literals and by restricting extension to be allowed only with clauses whose number has not yet been attached to any A-literal in the given chain. With this irrelevant modification ME without factorization will be denoted by PME (pure ME).

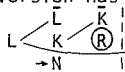
2.1.T. Each PME refutation of a matrix  $M$  can be simulated by (1.3) with exactly the same number of steps. (The representational form of (1.3) endows it with a little advantage over PME w.r.t. space.)

*Proof.* The simulation is obvious with the following correspondence. PME extension can be simulated by the steps 3,8,9 of (1.3). PME reduction can be simulated by line 10. PME deletion of bracketed literals (truncation) for completion of both, extension and reduction, is performed in 13. Note that the A-literals are those stored in ACT, while the B-literals are those stored in the first component of the entries in WAIT.

The implementation of (1.3) suggested in section 1 requires only one pointer instead of a whole cell in a chain. This amounts to  $3 \cdot \sum |C| + 2m$  locations required at most by PME as opposed to  $2 \cdot \sum |C| + 3m$  locations mentioned in section 1 for (1.3).  $\square$

### 3. Connection graph method

Algorithm (1.3) in its preliminary version has a redundancy which becomes apparent by the following example<sup>\*)</sup>:



Upon arrival at  $R$  the algorithm chooses the clause  $\{L\}$  in step 7. Complementarity is now obvious; the algorithm, however, does not notice this and considers the literal  $N$  on the stack. In general, instead of the single literal  $N$  the stack may contain arbitrarily many entries. Also, the example may be easily modified such that the deletion rule (see [7], lemma 3.1) does not apply. All linear methods like ME suffer from this redundancy.

This redundancy disappears in the full version of (1.3) which is discussed now. Namely, after the choice of the clause  $\{L\}$  as before "0" is replaced by "1" in the first element  $(L,1,0)$  in ACT within step 10 which allows to empty the whole stack in step 13. The justification for this is provided by the following theorem.

<sup>\*)</sup> A similar example was mentioned by R. Kowalski in a discussion with the author. Closer examination of it revealed several defaults in an earlier version of this paper.

3.1.T. Algorithm (1.3) - in its full version - is sound and complete.

*Proof.* For the preliminary version soundness and completeness was obvious. The only crucial difference apparently is within step 13 where subgoals on WAIT are simply deleted by the full version. Hence completeness is trivially carried over from the preliminary version. It is a little more difficult to show the same for soundness.

Let  $M$  denote any matrix. It was already mentioned in section 1 that (1.3) - in any version - determines pairs  $(L, \bar{L})$  of complementary literals, and thus proves complementary any path containing this pair; moreover, (1.3) never considers such a path again. Let us assume that during a run of (1.3) we note in a separate copy of  $M$  the pairs considered in this way by setting links between the respective occurrences of these literals.

Now, assume (1.3) runs in the full version on  $M$  but without the while-statement in step 13, and consider any situation when it arrives at step 13 and the if-condition turns out to be true. Let  $A$  denote the set of links which have been set so far in  $M$ , and  $P$  the set of paths which so far turned out to be complementary. This situation is illustrated in figure 2.

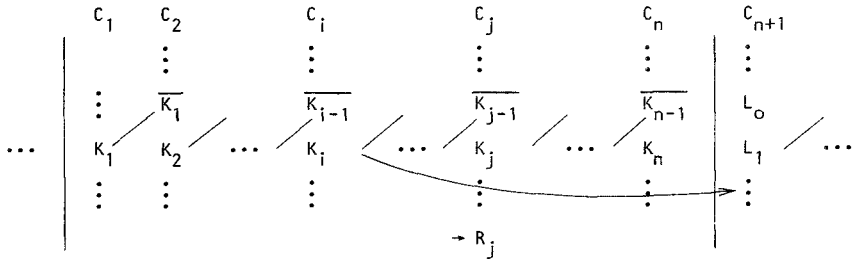


Figure 2. The connection graph  $(M, A)$

$K_1, \dots, K_n$  are literals in ACT (not necessarily all).  $K_j$ ,  $0 \leq i < n$ , denotes the rightmost literal which is connected with a literal beyond the ditch determined by the ditch-marker  $dm$  currently being considered in the if-condition of step 13 and illustrated by the right vertical line. The left vertical line illustrates the next  $dm$  on WAIT or the end of the matrix if there is none.  $j$  is determined by  $i < j \leq n$ .  $R$  is any literal in the clause  $C_j$  on WAIT remaining from  $C_j$ . (Note that the example at the beginning of this section is a special case of this general situation.)

Now, let  $P'$  and  $P_0$  be obtained from any path  $P \in P$  through  $M$  by replacing in  $P$  the literal  $S_{j'}$  from clause  $C_{j'}$  by  $R_{j'}$  for all  $j' \in J$  for any  $J$  such that  $\emptyset \neq J \subseteq \{i+1, \dots, n\}$ , and the literal  $S_j$  from clause  $C_j$  by  $K_j$  for all  $j \in \{i+1, \dots, n\}$ , resp.. By construction,  $P_0 \in P$  and the link which yielded complementarity of  $P_0$  cannot contain any of the literals  $K_{i+1}, \dots, K_n$ . Hence, such a link is also contained in  $P'$ , i.e.

$P' \in P$ . On the other hand, switching on the while-loop in step 13 prevents the examination of exactly those paths through  $M$ , which can be obtained as  $P'$ , and therefore has no effect w.r.t. soundness.  $\square$

After removal of the redundancy mentioned above the distinction, that has been elaborated between the connection graph procedure and SL-resolution

by Kowalski in [10], cannot be made between the connection graph procedure and (1.3). In fact, it seems that the following *conjecture* holds.

3.2. Each connection graph refutation of a matrix  $M$  can be simulated by (1.3) - or perhaps some further improvement thereof - with the same number of steps, considering corresponding pairs of literals but possibly in a different sequence.

In (3.2) would turn out to be true then this would mean a considerable advantage for (1.3) w.r.t. storage (no inherited links to be stored!) and w.r.t. the search space (no increase of the matrix!).

#### 4. Non-normal form methods

Figure 3 illustrates a redundancy inherent in all the methods discussed so far, and first noticed by Shostak in [13]. In the first snapshot, the path-checking process (1.3), which here simulates the SL-refutation given in figure 4a in [13], has just proved those paths complementary which contain  $\bar{N}$  but not  $Q$ . In the process which takes place between the two snapshots all

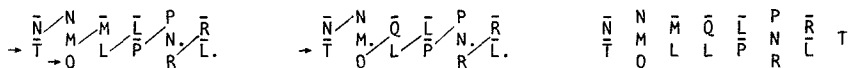
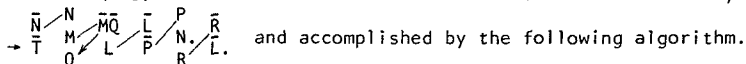


Figure 3. A redundancy in normal form methods  
(2 snapshots and the whole matrix)

paths through  $\bar{N}$  and  $Q$  turn out to be complementary. But except for the first step this whole process has already been performed before in literally the same way (compare the last four clauses in both snapshots).

This redundancy can be easily removed by generalizing the preliminary version of (1.3) to non-normal form matrices. This is illustrated by



4.1.A. (Preliminary version of (1.3) generalized to arbitrary matrices. Notation as in 1.3)

- 1:  $ACT \leftarrow \emptyset$ ; while non-empty(WAIT) do pop(WAIT); push(WAIT, "l<sub>m</sub>");
- 2: choose a clause  $C$  from the matrix  $M$ ;  $M \leftarrow M \setminus C$ ;
- 3: choose a matrix  $M'$  from  $C$ ;  
if  $C \setminus M' \neq \emptyset$  then push(WAIT,  $(C \setminus M', ACT, M)$ );  
if  $M'$  is not a literal then "push(WAIT, "l<sub>m</sub>");  $M \leftarrow M' \cup M$ ; goto 2<sup>1</sup>;  
 $L \leftarrow M'$ ;  $ACT \leftarrow ACT \cup L$ ;
- 4: if  $M = \emptyset$  then return ("non-complementary");
- 5: if there is no clause  $C \in M$  such that  $\bar{L}$  occurs in  $C$  then
- 6: "if there is no clause  $C \in M$  such that  $R$  occurs in  $C$  for some  $K \in ACT$  then  
"while pop(WAIT)  $\neq$  "l<sub>m</sub>" do pop(WAIT);  $ACT \leftarrow \emptyset$ ; goto 2<sup>1</sup>
- 7: else "choose  $K \in ACT$  such that  $\bar{K}$  occurs in  $M$ ;  $L \leftarrow K^{11}$ ;
- 8: choose  $C$  from  $M$  such that  $L$  occurs in  $C$ ;
- 9:  $M \leftarrow M \setminus C$ ; choose  $M' \in C$  such that  $\bar{L}$  occurs in  $M'$ ;  $C \leftarrow C \setminus M'$ ;
- 9<sup>1</sup>: for each  $C' \in M'$  such that  $C' = \{K\}$  and  $K \neq \bar{L}$  for some  $K$  do  
"check each entry on WAIT from top until the first occurrence  
of "l<sub>m</sub>" whether for its first component, say  $C''$ ,  $K \in C''$ ;  
if this is true then cancel this occurrence of  $K$ ; if  
even  $C'' = \{K\}$  then remove the whole entry from WAIT<sup>1</sup>;

```

10: among the matrices of C delete those which are literals K such that
    K ∈ ACT;
11: if M' = L̄ and C ≠ ∅ then goto 3;
    if M' ≠ L̄ and C ≠ ∅ then 'push(WAIT, (C, ACT, M)); push(WAIT, "lm");
    goto 11';
    if M' = L̄ and C = ∅ then goto 12;
11': choose C from M' such that L̄ occurs in C; M ← (M' \ C) ∪ M;
    choose M' from C such that L̄ occurs in M'; C ← C \ M';
    goto 11;
12: while pop(WAIT) = "lm" do pop(WAIT);
    if empty(WAIT) then return ('complementary');
13: (C, ACT, M) ← pop(WAIT); goto 3;

```

4.2.T. a) Algorithm (4.1) is complete and sound.

b) For each matrix M in normal form there is a generalized matrix M' consisting of a factorized version of M such that each refutation of M by Shostak's graph construction procedure [13] can be simulated by (4.1) applied to M' with the same number of steps (while the converse does not hold).

For reasons of space it is only mentioned that step 9' simulates "reduction using a C-literal", the other steps behaving similar as pointed out in the proof for (2.1).

Acknowledgements. I thank R. Kowalski and G. Wrightson for discussions and comments and A. Bussmann for the typescript.

#### References

- [1] P.B. Andrews, Refutations by matings, IEEE Transactions on Computers C-25 (1976) 801-807.
- [2] P.B. Andrews, General matings, Proc. Fourth Workshop on Automated Deduction, Austin, Tex. (W.H. Joyner, jr., ed., Yorktown Heights, 1979) 19-25. Also JACM (to appear).
- [3] W. Bibel, An approach to a systematic theorem proving procedure in first-order logic, Computing 12 (1974) 43-55.
- [4] W. Bibel, Effizienzvergleiche von Beweisprozeduren, GI-4. Jahrestagung, Lecture Notes in Computer Science, 28 (Springer Verlag, Berlin-Heidelberg-New York, 1975) 153-160.
- [5] W. Bibel, A syntactic connection between proof procedures and refutation procedures, in Theoretical Comp. Science, 3rd GI Conf., Lecture Notes in Computer Science 48 (Springer Verlag, Berlin, 1977) 215-224; First presented at the Conference on Automatic Theorem Proving, Oberwolfach, (Jan. 1976).
- [6] W. Bibel, Tautology testing with a generalized matrix reduction method, Theoretical Computer Science 8 (1979), 31-44.
- [7] W. Bibel, On matrices with connections, Bericht 79, Universität Karlsruhe (1979), submitted to JACM.
- [8] W. Bibel, A theoretical basis of the systematic proof method, Proc. MFCS'80, Lecture notes in Computer Science (Springer Verlag, Berlin, to appear).
- [9] W. Bibel and J. Schreiber, Proof search in a Gentzen-like system of first-order logic, Proc. Int. Computing Symposium (North-Holland, Amsterdam, 1975) 205-212.
- [10] R. Kowalski, A proof procedure using Connection Graphs, JACM 22 (1975) 572-595.
- [11] D.W. Loveland, Automated theorem proving: a logical basis (North-Holland, Amsterdam, 1978).
- [12] D. Prawitz, A proof procedure with matrix reduction, Lecture Notes in Mathematics 125 (Springer, Berlin, 1970) 207-213.
- [13] R.E. Shostak, Refutation graphs, Artificial Intelligence 7 (1976) 51-64.

## Representing Design Alternatives<sup>1</sup>

Daniel G. Bobrow and Ira P. Goldstein  
*Xerox Palo Alto Research Center  
 Palo Alto, California 94304, U.S.A*

### Abstract:

Artificial intelligence systems are complex designed artifacts. Techniques used in AI systems to describe structures and to represent alternatives can be used to support the design of the systems themselves. PIE is an experimental personal information environment which provides users with descriptive structures for programs and documents. In PIE, alternative designs for programs and documents are simultaneously viewable in the system through the use of a context structured database. This short paper gives an overview of how the use of these facilities improves the design environment for builders of software systems.

### Introduction

A major activity in artificial intelligence research is the design of complex systems. Yet most software environments do not support this activity well. They do not allow within the system description of different properties of a design nor the flexible examination of alternative designs. All designers create alternative solutions, develop them to various degrees, compare their properties, then choose among them. Yet most software environments do not allow alternative definitions of procedures and data structures to exist simultaneously; nor do they provide a representation for the evolution of a particular set of definitions across time. It is our hypothesis that a context-structured database can substantially improve the programmer's ability to manage the evolution of his software designs.

Present computing environments support the creation of alternative designs only with file services. Typically users record significant alternatives in files of different names; the evolution of a given alternative is recorded in files of the same name with different version numbers. We contend that this use of files provides both an impoverished structure as well as an inflexible one. The poverty is a result of the fact that file names are simply a limited length sequence of characters, hardly an adequate scheme to describe the purpose and contents of a file, and its relation to other files. It can be an adequate reminder to the originator of the name, but is often opaque to a new reader. The rigidity is a reflection of the fact that one typically cannot use parts of files as part of a new composite design, except by tedious text editing. Finally, the most serious limitation is that files are "off-line" in the sense that the alternative designs are not stored within the computing environment in a form that can be easily manipulated by the

<sup>1</sup>To be published in the Proceedings, Artificial Intelligence and Simulation of Behavior Conference, July, 1980, Amsterdam. A more extended discussion of this research can be found in Goldstein & Bobrow [80].

programmer. Although Interlisp [Teitelman, 78] provides some facilities for manipulating pieces of a file (e.g. individual function definitions), it still suffers from the "off-line" limitation.

To ameliorate this software bottleneck, we have constructed a computing environment in which "on-line" descriptions of alternative software designs can be readily created and manipulated. We use a context-structured description-centered database to describe code. Such databases have been explored in artificial intelligence research for over a decade as a mechanism to represent alternative world views. [e.g. Hewitt, 71; Sussman & McDermott, 72].

Our application of this machinery is novel in several respects. (1) Previous applications have focussed on the use of such databases by mechanical problem solvers. We are exploring the use of such databases in a mixed-initiative fashion with the user primarily responsible for their creation and maintenance. (2) Previous applications have always demanded a uniform overhead in space and time for adopting the context machinery. We are exploring configurations for a design environment that allow the programmer to trade flexibility for efficiency, decreasing the system's investment in tracking the evolution of particular parts of a design at the price of not being able to represent alternatives simultaneously in primary memory. Thus, employing the design environment is not an all or nothing choice for the user. (3) Previous applications have been to problems of limited complexity. In our application of context structured databases to software design, we are exploring their utility in a world several orders of magnitude more complex.

To understand the pros and cons of context structured environments for software design, we have implemented a prototype environment and conducted several experiments. The environment is called PIE, an acronym for personal information environment. PIE allows the user to build context sensitive descriptions of code, documents, and, indeed, any object for which a machine representation exists. PIE has been employed (1) to allow a programmer to create alternative software designs, examine their properties, then choose one as the production version, (2) to coordinate the interactive design of two programmers, and (3) to coordinate the documentation and definitions of an evolving package of code.

### **The Smalltalk environment**

To describe PIE further, we must first introduce Smalltalk [Ingalls, 78; Kay, 74], the programming environment in which it has been implemented. Smalltalk is an object-oriented programming language. (See Dahl & Nygaard [66] on Simula and Hewitt et al [73] on "actors" for related work on such programming languages). Behavior arises from the transmission of messages between objects. Each object is, in essence, a

simulation of a computer. It can respond to some number of messages and it maintains its own state between message invocations.

The message set of an object is specified by Smalltalk's class structure. Each object is an instance of a class. When a message is sent to the object, it asks its class for the method associated with that message. The class either contains the definition directly, or if not, passes the request to its superclass. For the object to understand the message, its definition must occur somewhere in this superclass chain. Thus, objects of the same class are analogous to computer products of the same model.

Figure 1 shows a fragment of the definition of a Smalltalk class for **Spaceship**. The fragment shown indicates that instances of **Spaceship** understand messages that simulate motion and collision and that each instance carries its own private state regarding its position and velocity.

---

### Class new title: Spaceship

```
superClass: Object "class Object is the root of the superClass hierarchy."
declare: 'allSpaceships' "a class variable --shared by all instances"
fields: 'position velocity' "instance variables -- each instance has private versions of these"
```

### Moving "methods are divided into 'protocols' -- this one is called Moving"

```
accelerate: dv "dv is the argument of the method with selector accelerate"
    [velocity+velocity+dv]

move [position+position+velocity. "points understand the message +"]
    self crashes => "self refers to this instance. => indicates a conditional expression"
    [↑ self explode] "if condition is true, move returns with value of self explode"
    self display. "done if condition is false -- display is a message this instance understands"]
```

### Collisions "another protocol"

```
crashes | ship "ship is a local variable for the activation"
"This assumes that all ships are of unit size, and collide only when at the same point"
[for: ship from: allSpaceships do: [ ship collideAt: position =>[↑true]].↑false]

collideAt: place
"a method to test if I collide with another object at place."
[position=place =>[↑true] ↑false]
```

Figure 1: Partial Definition of a Smalltalk class

---

We chose Smalltalk over Lisp, the usual vehicle for AI research, because Smalltalk has a superior set of interactive display facilities. DLISP [Teitelman, 77] provides enough capabilities we believe, but was not available on the same fast hardware. These interactive display facilities were of critical importance to allow the functionality of the design environment to be delivered to a user. No matter how powerful the design tools,

no experiments would have been possible with an interface based on an inadequate communication channel. Using Smalltalk, however, has required that we reimplement machinery common to such AI languages as FRL [Goldstein & Roberts, 77] and KRL [Bobrow et al, 77]. This has proved straightforward because the object oriented structure of Smalltalk is congenial to the frame-based viewpoint of a AI representation languages.

### The PIE environment

To describe Smalltalk code, we created a class of Smalltalk objects called *nodes*. Nodes are analogous to KRL units, or FRL frames: they consist of a set of attribute value pairs with support for attached procedures, the use of defaults, meta-descriptions and inheritance.

PIE provides convenient ways of viewing relationships between nodes, and viewing and changing the properties of nodes. One can automatically create nodes which describe existing pieces of the Smalltalk system, and conversely, make the system congruent with a description of it. Node23 in Figure 2 is a description that might have been computed from one method of the Smalltalk code shown in Figure 1.

---

```

Node23
class      Node17      "Node17 is the node describing the class Spaceship"
selector   'crashes'   "This is a unique string -- like a Lisp Atom"
localVariables ('ship)  "This is a set of unique strings"
variablesUsed ('ship 'allSpaceships 'position 'mySize)
methodBody "This is an editable paragraph"
  [for: ship from: allSpaceships
   do: [ ship collideAt: position =>[+true]].+false]
comment
  'This assumes that all ships are of unit size, and collide
   only when at the same point'

```

---

Figure 2. A node describing the method for **crashes**

In PIE, changing the values of any of these attributes does not automatically change the object being described by the node. The node describes an intended object in the system, not necessarily the version that exists in the system. This is worth emphasizing as one of the principles characterizing our point of view towards the design process.

★ **The Description Principle:** In a system there should exist a descriptive level at which objects can be described without actually affecting the objects themselves.



## Representing alternative designs

Using node structure, there are two distinct ways to have alternative descriptions of the same object: coreference and context. We have explored both, with our current preference being for the use of contexts.

Coreference uses separate nodes to describe separate alternatives. In Figure 3, **Node25** is a description of an alternative version of **crashes**. The intended identity of the **Node23** and **Node25** (they are both describing the same object) is made explicit with the **coreferentNodes** attribute.

---

```

Node25
class      Node18           "Node18 is the node describing the class Spaceship which differs
                             from Node17 in having an additional instance variable -- mySize"
selector  'crashes
localVariables ('ship)
variablesUsed ('ship 'allSpaceships 'position 'mySize)
methodBody "a different method body"
  [for: ship from: allSpaceships
   do: [ ship collideAt: position of: mySize =>[true]].ifFalse]
comment 'Uses mySize for each ship to determine overlap'
coreferentNodes (Node23)

```

**Figure 3. An alternative method for *crashes***

---

However, coreference has certain difficulties. The first is that it does not represent the manner in which two descriptions may differ on some attributes but otherwise be identical. The second is that the coordination of the choice of **Node23** vs. **Node25** and other choices in the system for consistency is not expressed. For this reason we have chosen to explore another way of expressing alternatives.

In this second method, all descriptions (values of attributes) of any node are relative to a context. *Context* as we use the term extends the notion of context as used in Conniver [Sussman & McDermott, 72], and has certain similarities to the vistas of partitioned semantic nets [Hendrix, 75].

★ **The Context Principle:** All attribute-values in the system are relative to a context, and alternatives in a system are expressed by alternative contexts.

When one retrieves the values of attributes of a node, one does so in a particular context, and only the values assigned in that context are visible.

## Incremental design

Design involves more than the consideration of alternatives. It also involves the incremental development of a single alternative. Every programmer is aware that software has a life cycle: following its birth, it undergoes progressive refinement in response to changing external requirements. PIE supports the incremental modification of a design by providing a fine structure to contexts that we have not, as yet, discussed.

A context is structured as a sequence of layers. It is these layers that allow the state of a context to evolve. The assignment of a value to a property is done in a particular layer. Thus the assertion that a particular procedure has a certain source code definition is made in a layer. Retrieval from a context is done by looking up the value of an attribute, layer by layer. If a value is asserted for the attribute in the first layer of the context, then this value is returned. If not, the next layer is examined. This process is repeated until the layers are exhausted.

Figure 4 shows a layer C containing some coordinated changes to the spaceship class of Figure 1. This layer contains those changes necessary to allow the class to use size information in determining collisions. In a context which contained this layer dominating those containing the information implicit in Figure 1, the changes would be visible. Those attribute-values such as the superclass of Spaceship that are not contained in layer C would be found in less dominant layers.

---

```

Node17 "the node for the class Spaceship"
  fields: ('position' 'velocity' 'mySize)    "a change in a declaration"
  methods (... Node23 Node27 ...)

Node23 "the node for the method crashes"
  methodBody
    [for: ship from: allSpaceships
     do: [ ship collideAt: position of: mySize = >[true]].false]

Node27 "the node for the method that tests for a collision"
  selector 'collideAt:of:'
  methodBody
    [(position + mySize > place-size) and: (position - mySize < place + size) = >[true]
     false]

```

**Figure 4. Layer C, containing coordinated changes to use mySize**

---

Figure 5 shows several spaceship nodes in which the values of attributes have not been filtered by a context sensitive lookup. Instead, we see the underlying data structure, which is an association list of layers and values. Layer B is the base layer in which all the nodes were presumed to have been originally defined for this example.

---

```

Node17 "the node for the class Spaceship"
  fields:   LayerB ('position' 'velocity')
           LayerC ('position' 'velocity' 'mySize')
Node23 "the node for the method crashes"
  methodBody
    LayerB
      [for: ship from: allSpaceships
       do: [ ship collideAt: position =>[true]].true]
    LayerC
      [for: ship from: allSpaceships
       do: [ ship collideAt: position of: mySize =>[true]].true]

```

Figure 5. An unlayered view of node structure

---

Extending a context by creating a new layer is an operation that is sometimes done by the system, and sometimes by the user. The current PIE system adds a layer to a context each time the context is modified in a new session. Thus, a user can easily back up to the state of a design during a previous working session. The user can create layers at will. This may be done when he or she feels that a given groups of changes should be coordinated. Typically, the user will group dependent changes in the same layer.

Given the existence of layers, a complex design developed over many stages can be summarized into a single new layer. The old layers, reflecting past choices, can then be deleted. Thus, the designer, if he wishes, can compress the past, achieving a more compact representation at the price of no longer representing the dynamics of the design.

### Coordinating designs

So far we have emphasized that aspect of design which consists of a single individual manipulating alternatives. A complementary facet of the design process involves merging two partial designs. This task inevitably arises when the design process is undertaken by a team rather than an individual. To coordinate partial designs, one needs an environment with these properties: (1) *non-interference*. Two designs may overlap. It must be possible to examine the overlap without the designs overwriting one another. (2) *incompleteness*. It must not be necessary for a design to be complete before it is examined. (3) *merging*. It must be convenient to create a common design from the individual contributions. It was encouraging for us to learn that the context/layer machinery created to manage alternatives lent itself well to meeting these requirements for coordinating partial designs.

Non-interference between the overlap of two partial designs was accomplished by adopting the convention that different designers place their contributions in separate layers. Thus, where an overlap occurred, the divergent values for some common attributes were separated by distinct layers. Handling incomplete designs of software was facilitated by the distinction between intensional node descriptions and the actual code definitions. Since the node descriptions were not installed code, they could be partial and hence non-executable with no difficulty.

Merging two designs can be viewed as a process that creates a new layer into which are placed the desired values for attributes as selected from two or more competing contexts. It is hence very much like the summarization process described earlier, but it is relative to more than one context and requires user interaction. For complex designs, the merge process is, of course, non-trivial. We do not, and indeed cannot, claim that PIE eliminates this complexity. What it does provide is a more finely grained descriptive structure than files in which to manipulate the pieces of the design.

Understanding how to merge two designs is facilitated by examining commentary supplied by the designers regarding the rationale of their choices. But this raises the classic software problem of coordinating documentation with design. Fortunately no additional machinery is required in PIE to address this problem. Commentary such as the rationale of a procedure, or its dependencies on other procedures, can be stored as attribute value pairs within the node describing the procedure in question. A request to be informed of the rationale of some change is answered by fetching this information from the same layer as the one which records the change, thus keeping them coordinated. Figure 4 shows how the rationales of various method definitions are recorded in the layer along with the altered definitions.

### **Complexity.**

We claimed in the introduction that PIE copes with problems several orders of magnitude more complex than those previously represented in AI systems such as Conniver. By complexity we mean both the size of the data base in the system, and the variety of operations done on contexts. The Conniver database was never efficient enough to implement any useable subsystems. McDermott's [McDermott, 74] examination of the Monkey and Bananas problem within Conniver exercised it to its limit.

PIE is able to build a context sensitive description of any class within Smalltalk. Thus, it can be applied to any programming problem that a Smalltalk programmer undertakes. This is analogous to using Conniver to build a programmer's interface to Lisp. Attacking problems of this size is, in part, possible because we have more computational resources than were available in the early 70's. PIE runs as a stand alone job on a processor with at least the power of a KA10. However, it is also possible because we have implemented

machinery to allow the programmer to move between context sensitive and context free descriptions at will. Thus, there is a more congenial marriage between PIE and Smalltalk than there was between Lisp and Conniver. This is discussed in the next section.

An interesting side effect of PIE's ability to describe any code within Smalltalk is that it can and has been used to describe itself. Thus, PIE's present capabilities have passed the test of being sufficiently powerful to support its own development, for example, by allowing us to examine alternative implementations of the PIE user interface within PIE.

### **Efficiency versus Flexibility**

PIE allows the user to trade flexibility for efficiency. At one extreme, the user can employ standard Smalltalk mechanisms for defining new code. If this route is chosen, then no evolutionary history is maintained, and no context overhead is paid. However, if the user wishes to pay the price of some decrease in efficiency of storage and retrieval time, then he can first build a set of nodes describing Smalltalk code, then continue his development in a context structured fashion. From this point forward, the evolutionary history is maintained. If the user reaches the point where he once again prefers efficiency to flexibility, the context definitions can be converted to pure Smalltalk and the layers deleted. If desired, the user can first store the layers remotely, preserving the ability to recreate the context description later. All these facilities are currently implemented.

This discussion suggests how a central design facility can serve as the nucleus of a network of remote servers that provide current packages to users. Periodically, the design server can release new layers to these servers with updates to particular designs. The servers can then generate new Smalltalk versions and release these designs to clients. Clients who wish to know what has changed, can get a description from the new layer.

### **Interaction**

PIE's ability to represent non-trivial alternative designs raises deep problems related to the user interface. How can we make available this power in a useable form? What are the cognitive requirements of the programmer? Presently we are employing an interface modelled on the standard Smalltalk interface for examining and altering code. This interface, called the browser, displays a hierarchy of descriptions of Smalltalk code to the user. The user can examine any method by a process of selection that specifies first a category of classes, then a particular class, then a protocol of methods within the class, and finally a particular method. This scheme of organizing code into a four-level taxonomy has been adopted in PIE to minimize the overhead for a Smalltalk user learning to employ the PIE environment. However, PIE makes this classification context

dependent. As with the standard Smalltalk browser, the user can alter the definitions of any object viewed. But these alterations are made in the dominant layer of the associated context, and do not affect the Smalltalk kernel itself, whereas making changes with the standard Smalltalk browser forces immediate incorporation of any changes.

Research is needed to explore whether this interface is adequate given the increased complexity of a context structured environment. In Smalltalk, the hierarchy of code definitions is the primary structural organization. In PIE, this hierarchy is now context dependent. Has this additional complexity made the Smalltalk organization inadequate? Will we need a classification scheme with more levels of division, or will some other kind of organization be appropriate? Just one of the problems that we will have to consider is that in a design environment, there is no need for a particular method description to be associated with only a single class, even though the actual Smalltalk system requires that the method be separately compiled for each class to which it belongs. Hence, a strict hierarchy is obviously inadequate.

## Conclusions

This paper presents only a sketch of the PIE system; our research is reported in greater detail in Goldstein & Bobrow [80]. We have not discussed here issues in the design of the user interface, although a successful interface is critical to delivery of these capabilities to the user. We only suggest here that layered networks are applicable to more than software: an extended example in cooperative writing of a document is given in the larger work. Finally, the system has as yet had only limited use. We do not know which features will be used most, which need to be automated to be helpful, and which may prove to be too complex to be useful. Recording and analyzing this experience is an important part of our research program.

A major theme of Artificial Intelligence research has been the development of languages to describe complex evolving structures. In general, these structures have been the belief structures of an artificial being about some subject matter (e.g., the SRI consultant's [Hart, 75] beliefs about the state of a water pump being constructed, or SAM's [Schank et al, 75] beliefs about what went on in a story it just read). We have been exploring the premise that these techniques can be used to describe the complex evolving structure of a software system, and as such can provide aids to the designer of such a system. One use of artificial intelligence is to amplify human intelligence. We suggest that the (recursive) application of AI techniques to AI can have a powerful effect on the development of the field.

## References

- Bobrow, Daniel G., Winograd, Terry, and the KRL Research Group, Experience with KRL-0: One cycle of a knowledge representation language, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA: 1977, 213-222.
- Dahl, O.J., and Nygaard, K., SIMULA—an ALGOL-Based Simulation Language, *CACM* 9, September 1966, 671-678.
- Goldstein, I.P. and Bobrow, D.G., *A layered approach to software design*, Palo Alto, CA: Xerox Palo Alto Research Center, Computer Science Laboratory, 1980, in preparation.
- Goldstein, I.P. and Roberts, R.B., NUDGE, A knowledge-based scheduling program, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA: 1977, 257-263.
- Hart, P., Progress on a computer based consultant, *Advance papers of the fourth international joint conference on artificial intelligence*, Tbilisi: 1975, 831-841.
- Hendrix, Gary G., Expanding the utility of semantic networks through partitioning, *Advance papers of the fourth international joint conference on artificial intelligence*, Tbilisi: 1975, 115-121.
- Hewitt, C., Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot, Ph.D. Thesis, June 1971 (Reprinted in AI-TR-258 MIT-AI Laboratory, April 1972.)
- Hewitt, C., Bishop, P., and Steiger, R., A universal modular ACTOR formalism for artificial intelligence, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, 1973, 235-245.
- Ingalls, Daniel H., The Smalltalk-76 Programming System: Design and Implementation, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, Tucson, AZ: January 1978, 9-16.
- Kay, A., SMALLTALK, A communication medium for children of all ages, Palo Alto, CA: Xerox Palo Alto Research Center, Systems Science Laboratory, 1974.
- McDermott, D.V., *Assimilation of new information by a natural language-understanding system*, AI-TR-291 MIT-AI Laboratory, February 1974.
- Schank, Roger, and the Yale AI Project, SAM—A story understander, Yale University, Computer Science Research Report #43, August 1975.
- Sussman, G., and McDermott, D., From PLANNER to CONNIVER—A genetic approach, *Fall Joint Computer Conference*, Montvale, NJ: AFIPS Press, 1972.
- Teitelman, W., *Interlisp reference manual*, Palo Alto, CA: Xerox Palo Alto Research Center, Computer Science Laboratory, 1978.
- Teitelman, W., A display oriented programmer's assistant, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA: 1977.

PATTERN-BASED REPRESENTATIONS OF KNOWLEDGE  
IN THE GAME OF CHESS

M.A.Bramer  
Mathematics Faculty  
Open University  
Milton Keynes, ENGLAND MK7 6AA.

Abstract

The focus of recent Artificial Intelligence research into Computer Chess has been on endgames. These afford the possibility of controlled experimentation, whilst retaining much of the complexity of the full game of chess. This paper discusses some of the specific reasons for complexity in the endgame and considers its effects on human chess-playing strategy, textbook descriptions and the development of programs. In programming the endgame the researcher is faced with a range of decisions concerning the quality of play to be aimed at, the balance between knowledge and search to be adopted and the degree to which the playing strategy should be understandable to human chessplayers. A model for representing pattern-knowledge is described which has enabled the development of algorithms to play a number of endgames. Three algorithms representing different levels of performance for the endgame King and Pawn against King are compared, in order to discuss the tradeoff between complexity and completeness, on the one hand, and compactness and comprehensibility, on the other. Finally, the role of search in reducing the amount of knowledge to be memorised is considered and an extension to the basic model to incorporate deeper search is discussed.

Introduction

The game of Chess combines complexity with a well-defined structure, together with an extensive background culture against which a given standard of program performance can be evaluated.

The focus of much research into computer chess has moved towards the study of endgames, which retain much of the complexity of the full game whilst affording the possibility of controlled experiments and precise quantitative analysis. It is also notable that conventional chess-playing programs using deep search with simple evaluation functions generally perform very badly in endgames, where knowledge, rather than calculation, is probably the major factor in human play.

Studies of even the most elementary endgames such as King and Pawn against King (KPK) and King and Rook against King (KRR) have revealed surprising complexity. Numerous difficult cases have been found which are not given in textbooks and conventional programming techniques have proved most unsuccessful. Knowledge-based algorithms for a variety of endgames have been given by Bramer (1977b), Bramer and Clarke (1979), Bratko and Michie (1980) and elsewhere, in each case developed after a lengthy series of trials and careful refinement.



There are a number of reasons for this unexpected complexity.

(1) Limitations of current theoretical knowledge of chess. Experiments have revealed important errors and omissions in the known theory of various endgames, with erroneous evaluations previously made by experts and counter-intuitive moves and results found in several important positions.

KPK and KRK are believed to be fully understood theoretically by reasonably strong players. Nevertheless there are difficult cases which are not given in the majority of widely available textbooks, or (in some cases) in any of them. The authors have either been unaware of the difficulties or have excluded them as unimportant.

(2) Boundary effects caused by the board edge. Inability to manoeuvre beyond the Rook files or the first or eighth ranks leads to difficulties and 'special cases' affecting general strategies (particularly for KPK).

(3) 'Discontinuities' in the rules of chess. Stalemate, the option of an initial double Pawn move and pawn promotion can be viewed as 'discontinuities' in the normal rules (being unable to avoid King capture loses, pieces move in the same way anywhere on the board) and both are significant in endgame play.

(4) Chessboard geometry.

The geometry of the chessboard is non-Euclidean (and varies from piece to piece). Measuring in terms of King moves (one square vertically, horizontally or diagonally at a time) the distance from square A1 to square A7 is 6 units either directly or via two sides of a triangle (A1-B2-C3-D4-C5-B6-A7). The situation is compounded by the existence of squares on to which a King may not legally move, which alters its 'effective distance' from a given square. Some analysis of effective distance in the KPK case is given in Bramer (1977a).

With this geometry it is difficult to define even apparently simple geometrical relationships, such as 'Black King can take Pawn' for KPK.

From the above it is clear that endgames, especially those with only a small number of pieces, differ from middlegames by being much more ill-behaved, with numerous special and unexpected cases arising. Moreover the traditional computer chess technique of using a sophisticated search algorithm with a fairly simple evaluation function is not applicable (at least without major modification) to endgames, where it is easy to find examples of positions which would require a search of 30 ply or more deep to find the one (counter-intuitive) winning move.

#### Textbook descriptions of endgame strategies

From the chess literature it is evident that endgame play depends much more on the use of plans based on a knowledge of significant configurations (or patterns) of pieces than on deep analysis of possible variations.

For elementary endgames such as KPK and KRK, the plans are very simple (e.g. 'move as close to White's Pawn as possible') and the search very shallow, in fact almost non-existent.

Conventional chess-playing programs start with search and use knowledge to reduce the amount of search required. In endgame play (especially with few pieces) it is probably more appropriate to think of using search as a means of reducing the amount of knowledge that must be stored.

A typical textbook description comprises a small number of general 'rules of play' together with some example variations from diagrammed positions. The rules are normally only imprecisely worded and omit important details which have to be inferred from the variations given.

Although standard textbooks such as Fine (1941) are often thought of as definitive and exhaustive, this is far from true. Aside from gaps or errors in chess theory as mentioned previously, there is no attempt made to deal with all possible situations which can arise even in the simplest endgames. Fine remarks "I have given a large number of rules which are at times incorrect from a strictly mathematical point of view, but are nevertheless true by and large and are of the greatest practical value". Thus he concentrates on the typical cases to the exclusion (in general) of rarely arising exceptions even when these are known, and no effort is made to demonstrate the most efficient strategies (in the sense of the shortest possible win in every position).

#### Programming the endgame

In attempting to model the strong player's knowledge of the endgame, the researcher is faced by a number of decisions. One is whether to adopt a structural or a procedural representation, another is the level of performance for which he should aim. A helpful distinction can be made between winning algorithms which are optimal (i.e. the stronger side wins wherever possible in the smallest possible number of moves) and those which are correct (the stronger side wins wherever possible but not necessarily as quickly as possible). The evidence and examples given in Bramer (1980) strongly suggest that, even for KPK, strong players perform sub-optimally, although almost certainly correctly. For complex endgames strong players do not always perform even correctly.

There is a principle of sufficiency involved here. The game-theoretic maximum number of moves needed for the stronger side to win any winnable KPK position is only 19. The rules allow for 50 moves (without any piece taken or any Pawn moved) before a draw can be claimed. It is simply not worthwhile to overload the memory with numerous special cases (or spend time performing a deep analysis) to achieve optimal play, even assuming this is feasible, if there is a simple algorithm which suffices for correctness, still well within the constraints of the 50-move rule.

On the other hand, the endgame King, Bishop and Knight versus King is thought to require up to 34 moves to win and an error in certain critical positions can easily lead to an exceeding of the 50 move limit. In these cases it is worthwhile memorizing much more detail of difficult cases, although not necessarily all of them.

Clarke (1977) draws attention to the tradeoff between knowledge and search. At the extremes are a program which has full knowledge and uses no search (i.e. it simply looks up the best move in a table) and one which uses extensive search and has no non-trivial knowledge (i.e. it uses only the definitions of won, drawn and lost terminal positions). In general, programs will lie somewhere along this spectrum, with recent

Artificial Intelligence research concentrating on programs towards the 'knowledge' end.

Michie (1980) has described another tradeoff: this time between the performance of a program and its comprehensibility to subject experts. High-performance programs which are also comprehensible are referred to as lying inside a 'human window'. In applied domains such as medicine, the importance of this concept is that with an appropriate representation it may be possible to satisfy subject experts of the accuracy of machine judgements, for example by describing the factors which were taken into account, the weighting given to each, the diagnostic inference rules applied and the reliability attached to the result.

Such considerations argue strongly in favour of the choice of a structural representation, particularly one based on rules or patterns. A pattern-based approach also allows subject experts, as well as judging the rules given, to add their own experience in codified form.

An interesting case where trust in an unfathomable program was required has already arisen in chess. Michie (1977) reports that the grandmaster Bronstein made use of a database of the best move in every position for part of the King, Queen and Pawn against King and Queen endgame for analysis of an adjourned position. If the move retrieved from the database had conflicted with Bronstein's own judgement it would have been virtually impossible to check whether it arose from an error in creating the database or was in fact accurate.

Rule-based representations of a body of knowledge can be viewed as having two possible functions: one as a replacement for the textbook, to be committed to memory by the chessplayer, the general medical practitioner etc. and used as required, the other is as an expert computerized assistant typically used in an interactive mode. In both cases, there is an important need for comprehensibility. However, a set of rules for the former will generally need to be much briefer than for the latter, to match the limitations of human short-term memory, and again there is a tradeoff, this time between accuracy (or completeness) and compactness within a given framework.

A desirable feature of a rule-based expert system is that learning within it will generally proceed monotonically, i.e. that adding a new rule should not invalidate old ones but should lead to an improvement in performance. This is only likely to be true if the underlying representation is well chosen. The proliferation of rules each covering a small number of cases which the subject expert would not regard as reflecting aspects of the complexity of the domain in question is a good indication that the representation is probably not appropriate.

The weakness of general-purpose representations of knowledge is that they fail to take into account the specific features of the domain under consideration. Thus it may be that in specifying the 'King can catch Pawn' predicate, some descriptors should always be used in preference to others where possible or that some descriptors should only be used in conjunction with others, or if others do not appear, etc.

#### A model for representing pattern-knowledge for chess endgames

A representation designed to enable the chessplayer's knowledge of an endgame to be represented in a structurally simple and compact form, capable

of incremental iterative refinement to improve its performance whilst preserving its initial properties, is given in Bramer (1977b) and Bramer and Clarke (1979) and summarized below. It is assumed that the problem is to construct an algorithm to find a move for a chosen side (say White) in any position  $p$ , for a given endgame. The basic move finding algorithm is then as follows:

- (a) generate the set  $Q$  of immediate successors (Black to move) of  $p$
- (b) find the highest ranked member of  $Q$ , say  $q$
- (c) play the move corresponding to  $q$ .

To achieve step (b) an implicit ranking is defined on the set  $Q^*$  of all legal BTM (Black to move) positions for the endgame in question. Each such position is assigned to exactly one of a number of disjoint and exhaustive classes which partition the set  $Q^*$ .

The ranking of each BTM position is then determined by its class value (which is constant for all the positions in any class) and the values of a number of associated functions. These vary from one class to another, in general. For positions in the same class, the functions used are always the same although their values will vary from one position to another. To compare the values of two positions, their class values are compared, with the larger value indicating the higher-ranked position. If there is a tie, the first associated function is used for comparison. If there remains a tie, the second associated function is used, and so on. (Any ties remaining after all the associated functions have been used are resolved arbitrarily.) When comparing the values of associated functions, in some cases the larger value is preferred, in some cases the smaller is, depending on the particular function. The intention is that each class should correspond to some significant static feature of the endgame as perceived by chessplayers, e.g. 'Black is in check'. The associated functions correspond to relevant numerical values, such as the distance between the two Kings.

Assigning a position  $q$  to a class is achieved by working through a series of predicates (called rules) in turn until one is satisfied. (Subsequent rules are not evaluated.) A position  $q$  is defined to belong to a particular class  $N$  if and only if rule  $N$  is satisfied by  $q$  and none of the preceding rules are satisfied.

This procedure ensures that each position belongs to only one class and helps to simplify the definition of the rules. To ensure that each position belongs to some class, the final rule is defined to be always true for any position  $q$ .

This model was used initially to develop an algorithm for the stronger side (White) of KPK which was thought to be a fully correct strategy as a result of extensive testing, reported in Bramer (1977b). Subsequent analysis revealed that the algorithm was not entirely correct. An optimal strategy was developed by a process of iterative refinement using a database of the shortest-path winning move (or moves) in every position. A correct algorithm has now been refined from the original 'near correct version' by the semi-automatic refinement process based on inspection of 'win-trees' given in Bramer (1979) and is summarized in the Appendix. (The development of a correct algorithm for KKK is described in Bramer (1979).)

Three algorithms for King and Pawn against King - a comparison

The following figures give basic information about each of the algorithms.

Figure 1 - Three algorithms for KPK

Algorithm	Description	Classes	Associated Functions	Max. depth of win (ply)
A	'near correct' strategy	19	9	--
B	correct strategy	20	10	44
C	optimal strategy	38	13	38

Figure 2 - 'Optimality Levels' for algorithms A (near-correct) and B (correct)

	Algorithm A*	Algorithm B
Move played is optimal	59,888 (95.93%)	60,462 (96.77%)
Move played increases depth by 1	1,526 (2.44%)	1,075 (1.72%)
" " " " 3	673	660
" " " " 5	251	222
" " " " 7	68	47
" " " " 9	19	11
" " " " 11	5	1
" " " " 13	-	-
" " " " 15	2	2
<u>Total</u>	<u>62,432</u>	<u>62,480</u>

(Breakdown for all legal WTM positions, which are theoretical wins).

\* Excluding non-win preserving moves

Figure 3 - Class membership for algorithm B (correct strategy)

Class	Number of positions (BTM)	Class	Number of positions (BTM)
1	10,093 (10.3%)	14	1,620 (1.7%)
2	9	15	8,507 (8.7%)
3	12,985 (13.3%)	16	2,632 (2.7%)
4	35,026 (35.7%)	17	1
5	14,422 (14.7%)	18	4,971 (5.1%)
6	694 (0.7%)	19	5
7	50	20	4
8	6,045 (6.2%)		
9	42	Total	97,992
10	66		
11	584 (0.6%)		
12	60		
13	176		

(Pawn on file A-D, ranks 2-8.)

Classes 2, 7, 9, 10, 12, 13, 17, 19 and 20 total together 413 members (0.42%).

Figure 2 shows the 'Optimality Levels' for algorithms A and B; i.e. the amount by which the move selected in each theoretically won WTM position changed the maximum depth. For an optimal move the depth is decreased by one ply. The figure shows that in both cases the great majority of moves are either optimal or increase the depth by only one ply. The differences between the two algorithms are small and, in fact, algorithm B was formed from A by the addition of one new class (with only one member) and one new associated function plus slight changes affecting a few other classes.

However, the difference in performance is substantial. Algorithm A fails to win from as many as 4,602 theoretically won positions (WTM) and 4,351 (BTM). Only in 48 positions (WTM) is a move played which does not preserve White's winning advantage, the other positions simply transform into one another in cycles. This result reinforces the evidence given in Bramer (1979) for the KRK endgame that a change to the move played in a small number of positions can drastically alter the overall performance of an algorithm. Testing even by expert human players might never reveal that A was not a correct algorithm. Its errors will in general result in a cycle but this may be after many moves of otherwise expert play, possibly in response to poor play by Black.

To improve the performance of KPK from correct (B) to optimal (C) requires an increase from 20 to 38 classes and from 10 to 13 associated functions. This near-doubling of the size of the algorithm results in a relatively minor improvement in performance. The number of individual positions played optimally rises from 96.77% to 100% and the maximum depth is reduced from 22 moves (44 ply) to 19 (38 ply). The definitions of 38 classes would probably be too many to commit to memory, if the algorithm were to be used as a replacement for the textbook, whereas 20 classes would probably be acceptable. Either number would be satisfactory for an expert computerised assistant. The pattern of distribution of depths for algorithm B is very similar to that for C (theoretical maximum depths), and this is also true for algorithm A, which would tend to support the appropriateness of the representation adopted. In making the transition from algorithm B to C, it is clear that a 'diminishing returns' effect is involved. The maximum depth of 22 moves for algorithm B is still well within the 50-move drawing limit. For a practical player to take on the additional memory burden required to play optimally would simply not be worthwhile, an unnecessary violation of the principle of sufficiency.

The class membership table for algorithm B (Figure 3) shows that a fairly small number of classes account for the great majority of positions. The nine smallest classes contain less than a half of one percent of the positions, and four classes contain less than ten positions each. It is useful to consider whether classes with a low level of membership reflect 'special cases' of the domain in question or merely result from the particular representation used. In the case of algorithm B, the classes concerned do seem to correspond to clear special cases arising from the boundary effects, rule discontinuities etc. referred to previously. For example, Class 17 is used to deal with recognized difficulties with a Knight Pawn and Class 2 contains all the positions where Black is stalemated. Some of these special cases although important are not given in the major textbooks (or not in all of them) and this is more markedly so for algorithm C (optimal strategy). Bramer (1980) gives several examples of positions which are clearly special cases but are of no practical significance and would certainly never be quoted in textbooks. It is

evident that textbooks omit many special cases, even those which are necessary for correct play, and that a major reason for this is to reduce the amount to be memorized.

It is not suggested that algorithm B is a minimal correct strategy for KPK, i.e. one with the fewest number of rules possible. By removing certain of the special cases or by other changes it might be possible to construct an algorithm which was still correct but had a greater maximum depth, although still within the 50-move limit. Such a strategy would doubtless still have to include classes to deal with a (possibly reduced) number of special cases. If the aim were not to achieve correctness, but to perform expertly in practical play or to serve as a teaching document superior to standard textbooks, it could be argued that an abbreviated algorithm which omitted special cases but was still reliable in the great majority of cases was preferable.

### Extending the model

The most important way in which people reduce the amount of knowledge memorized without necessary loss of correctness is by making use of analysis or search.

For experienced players, search plays little part in simple endgames (although exact counting does) but increasingly more as the problem becomes more complex. Unfortunately, tournament chess-playing programs have found it necessary to use search of a volume and kind which is most unlike that of expert human players. An extension of the model previously described enables the use of pattern-knowledge to be combined with search deeper than one ply, but which is capable of careful control.

The final class (defined to be always true) is called the residual class; those with class values greater than that of this class are called positive, those with lower values are called negative. These latter two categories broadly reflect features of the endgame which are particularly desirable or especially undesirable. The most likely source of difficulty in programming complex endgames lies in specifying a sufficiently large number of positive or negative classes, with many important positions thus falling into the residual class.

There are four possibilities for a given set of successor positions:

- (a) at least one belongs to a positive class;
- (b) all belong to negative classes;
- (c) all belong to negative classes, except one which belongs to the residual class;
- (d) two or more positions belong to the residual class and the remainder (if any) belong to negative classes.

In cases (a), (b) and (c) the most favourable position can be found statically using position values in the usual way. In case (d), either the positions in the residual class can be taken as terminal and the associated functions used to calculate the value statically, in the usual way, or an analysis tree can be generated from each of the residual class positions, with the negative class positions rejected altogether.

Constructing an analysis tree in this way has the effect of reducing the amount of search by pruning all branches to positions in negative classes and defining terminal states of a given set of positions as a whole

(cases (a), (b) and (c) above). Since a residual class position can, at any stage, be regarded as terminal, with the static value of the position backed up the tree, the amount of analysis performed is subject to close control. In general, this form of the model is distinguished from conventional tree-searching in that search is intended to be used in a controlled way only as necessary to supplement the pattern-knowledge which it is believed is the fundamental component of the chessplayer's endgame knowledge. It corresponds roughly to the high-level rule 'if in an unfamiliar situation, search for possible forcing variations into known positions'. An alternative approach which uses recognition of patterns to invoke a goal-directed search (where each pattern is associated with its own list of goals) is described by Bratko and Michie (1980).

#### Appendix

#### A correct algorithm for the endgame King and Pawn against King

Figure 4 - Classes for King and Pawn against King (summary)

Class	Property of position q (Black to move)	Class value	Associated Functions
1	Pawn en prise	1	2, 3
2	Black is stalemated	2	-
3	Pawn is on eighth rank	20	-
4	Pawn can "run"	19	1
5	Black King is effectively closer to the Pawn than White (adjusted for second rank case)	3	2, 3
16	(Some Rook Pawn cases)	4	-
6	Black can move to "blockade" square	5	1, 2, 3
17	(Knight Pawn position - special case)	11	-
7	White is on the "blockade" square and Black can take opposition	6	-
8	Black King at least two files from White King on same side of Pawn, White King not below Pawn's rank	18	1, 4, 7
9	Kings on critical squares on same rank	17	1
10	Kings on critical squares, Black one rank above White	16	1
11	White King on Pawn's rank, above Pawn	15	1, 7
12	Kings in vertical opposition, with the White King on a critical square	14	-
13	Kings in opposition, White King above Pawn or both on sixth rank	12	1
14	White King on a critical square	10	1, 6, 5, 10
18	White King on a file above Pawn	9	4, 1, 8
19	(Pawn on sixth rank-special case)	8	-
20	(Pawn on fifth rank-special case)	13	-
15	(always true)	7	2, 3, 7, 9



Figure 5 - Associated Functions for King and Pawn against King

Function	Value of function
1	The Pawn's rank*
2	The file or rank distance between the Kings, whichever is the larger**
3	The file or rank distance between the Kings, whichever is the smaller**
4	The file distance between the White King and the Pawn**
5	The file distance between the White King and the Pawn*
6	The number of ranks the White King is above the Pawn*
7	The White King's rank*
8	The rank distance between the White King and the Pawn**
9	The file distance between the Kings*
10	The White King's file**
* The largest value should be taken	
** The smallest value should be taken	

#### References

- Bramer, M.A. (1977 a). King and Pawn against King : Using Effective Distance. Open University, Faculty of Mathematics, Technical Report.
- Bramer, M.A. (1977 b). Representation of Knowledge for Chess Endgames : Towards a Self-improving System. Ph.D. thesis, Open University.
- Bramer, M.A. (1979). Testing Correctness of Strategies in Game-playing Programs. Proceedings of the Sixth International Joint Conference on Artificial Intelligence.
- Bramer, M.A. (1980). Representing Pattern-Knowledge for Chess Endgames : an Optimal Algorithm for King and Pawn against King. In Clarke, M.R.B. (ed.), Advances in Computer Chess 2. Edinburgh University Press.
- Bramer, M.A. and Clarke, M.R.B. (1979). A Model for the Representation of Pattern-Knowledge for the Endgame in Chess. Int. J. Man-Machine Studies, 11, pp. 635-649.
- Bratko, I. and Michie, D. (1980). A Representation for Pattern Knowledge in Chess End-games. In Clarke, M.R.B. (ed.), Advances in Computer Chess 2. Edinburgh University Press.
- Clarke, M.R.B. (1977). A Quantitative Study of King and Pawn against King. In Clarke, M.R.B. (ed.), Advances in Computer Chess 1. Edinburgh University Press, pp. 108-118.
- Fine, R. (1941). Basic Chess Endings. David McKay, New York.
- Michie, D. (1977). End-game Secrets. Computer Weekly, 3rd November, (Chesslab).
- Michie, D. (1980). Problems of the Conceptual Interface between Machine and Human Problem-solvers. Edinburgh : Machine Intelligence Research Unit, Experimental Programming Report No. 36.

## EXTENDING MECO TO SOLVE STATICS PROBLEMS

by

Lawrence Byrd and Alan Borning  
 Department of Artificial Intelligence  
 University of Edinburgh

## ABSTRACT

Meco is a computer program for solving mechanics problems. To test the generality and extensibility of its representation and search control mechanisms, we extended Meco to handle problems from a new domain, namely that of statics problems. This paper describes the representation and solution of some of these problems.

Keywords: statics, mechanics, problem solving, inference

Introduction

Meco is a computer program for solving mechanics problems [2, 4]. Problems previously solved by the program have been from the areas of pulley problems, motion on complex paths, and motion under constant acceleration.

One of our goals in designing and building Meco has been to investigate general, extensible representation and search control mechanisms. We decided to test how well that goal had been satisfied by extending Meco to handle problems from a new domain, namely that of statics problems. To do this, we had to add several kinds of knowledge to the program: new types of objects, such as rods and springs; new quantities, such as elasticities; new formulae, such as Hooke's law; and new strategies, such as taking moments about a point. We found that in fact the program was easily extended. For the most part, we simply had to add new physics knowledge; only minor modification and debugging of the original set of rules was required.

The full Meco program is divided into three modules. The first module, the natural language component, accepts a problem statement in English, and from this produces a set of predicate calculus assertions describing a situation. The second module, the problem solving component, accepts such assertions as input, builds a strategy for solving the problem, and generates as its output a set of algebraic equations. These equations are then passed to the third module, an algebraic equation solver [3], which produces the final answer. In the work described here, we bypassed the natural language analysis phase, and entered the problems directly as formal assertions. All of our extensions were to the problem solving component; the algebra package was able to handle the equations produced from our new problems without modification.

Other AI researchers have also worked in this area, in particular Gordon Novak [8], E. Tyugu [9], and Jill Larkin [6, 7]. Several of the examples described here are taken from Novak's and Larkin's papers; indeed, it has been our intention to try and solve the problems handled by other workers

in this domain.

In the remainder of this paper, we will describe how the statics problems have been represented and solved, and attempt to bring out the reasons for the success of our extensions.

### Representing Statics Problems

We shall now discuss how an example problem is represented, and in the next section we shall show how it is then solved. This problem is taken from [8].

A scaffold 10 ft long is supported by ropes attached at each end. The scaffold weighs 100 lbs. A man weighing 150 lbs stands on the scaffold 4 ft from one end while another man weighing 175 lbs stands on the scaffold 2 ft from the other end. What is the tension in each of the ropes supporting the scaffold?

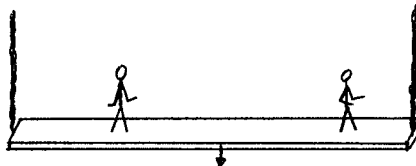


Figure 1: A Scaffold Problem

The problem contains various objects, such as the ropes, the scaffold, and the men; and certain quantities, such as tensions and masses. We represent these entities, their properties, and the relations between them by a set of predicate calculus assertions.

To start with, we need to represent a number of part/whole and connection relations. The scaffold has end points and internal points where the men are standing, while the ends of the ropes are attached to the end points of the scaffold. We represent this by idealizing the ropes as strings and the scaffold as a rod, both of which are types of LINE-like object. The men are treated as particles, which are types of POINT-like object. Points on the ropes and scaffold are also POINT's. All the required relations can then be represented using existing primitives for LINE-like and POINT-like objects. Figure 2 shows how the problem from Figure 1 can be broken down in this way, and introduces some names for the objects. This information is represented by assertions such as:

```
isa(string,rope1)
end(string,bottom1,right)
isa(rod,scaffold)
end(rod,leftend,left)
fixed_contact(bottom1,leftend,period1)
```

The concepts of sub/super-LINE's, POINT's of LINE's, and connections between POINT's can be used to represent uniformly a class of very common relations over a wide range of objects. Even abstract objects, such as periods of time and paths (trajectories of particles in motion), make use

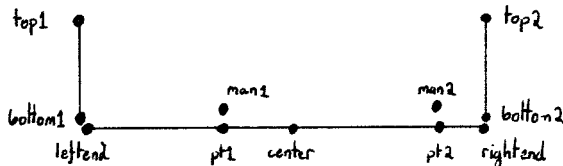


Figure 2: LINE/POINT relations for the Scaffold Problem

of the same underlying relations.

The objects also have properties like inclinations, masses, lengths, and tensions. These are represented as relations between the objects and corresponding symbolic quantities (which will have types like *angle*, *mass*, *distance*, and *force*). Examples of such assertions would be:

```
mass(man1,m1,period1)
tension(ropel,t1,period1)
separation(leftend,man1,d1,0,period1)
```

These assertions are input to the program and placed in the data base. They are then managed by an inference system (not described here, see [2, 4], which contains a wide range of inference rules about the mechanics domain. The inference system uses meta-knowledge about the properties of the predicates involved to control the use of these rules. It is capable of satisfying queries about the problem regardless of whether or not we have explicitly given the exact information the problem solver requires. This will often involve introducing new quantities not mentioned in the problem statement. In the current example all the required quantities are given, but this is not so for the examples given later.

#### Solving Statics Problems

The problem solver consists of inference rules which reason about the objects and relations in the problem, and about the strategies available which might relate 'sought' quantities to those 'given'. The quantities involved will be those mentioned in the problem statement, plus any new intermediate quantities introduced by the inference system.

Mechanics text books usually present formulae such as ' $F = M * A$ ' in their solutions to problems. Using such a formula involves knowledge about when it is applicable and how to correctly apply it, taking into account all the relevant objects and quantities. Our strategies consist of facts and rules that provide this information. For example, the 'resolution of forces' strategy (based on the above formula) contains knowledge about what types of quantity it relates (forces, masses and accelerations in this case), and rules that state exactly which forces etc need to be considered when dealing with particular object configurations.

The program currently has about fourteen such strategies which deal with a wide range of areas (such as principles of motion and energy). Only three of these are used for the problems in this paper. However, all of the strategies remain in the program and could be used if deemed applicable. A general goal directed method controls the application of

these strategies (see [4]).

Let us follow program's operation on the example from Figure 1. The problem is to solve for the two tensions in the ropes in terms of the masses and distances given in the problem statement. Considering the first tension, it is known to be a force involving the string rope1. The program now looks for a strategy that will relate the force tension1 to other quantities, and finds the resolve-forces strategy. The program's next goal is to find a situation in the problem to which the strategy can be successfully applied, in other words, to instantiate the variables in the strategy. One of the available rules states that if the force is a tension in a string, and the string has a point in contact with a rigid body (a rod), then resolving about the complete rod should be considered. This is the case in the current example, and the application of the strategy will now invoke rules which state that force contributions from all objects attached to points on the scaffold need to be taken into account. Finding these force contributions involves considering the two tensions and all the masses. The mass of the scaffold is treated as acting at the center of the scaffold. Thus, reasoning from general properties and rules has lead to the particular inter-dependencies of the quantities in the problem, which are now expressed in the form of an equation.

No new unknown quantities have been introduced, so the remaining goal is to solve for the second tension. The reasoning used above will now fail because the resolve-forces strategy will have already been applied to the scaffold. Another strategy, that of summing the turning-moments, also relates forces to other quantities. Similarly, another rule states that if there is a rigid body attached to a point of the string, then summing the turning-moments on that body (about some point other than where the string is attached) should be considered. Applying the new strategy will invoke rules about force contributions and distances. The inter-dependencies discovered can again be expressed as an equation, and so the result of this problem solving will be two simultaneous equations which can be solved by the equation solver to produce a final answer.

#### Further Examples

The following problem is taken from [6]:

Block B in Figure 3 weighs 160 lbs. The coefficient of static friction between block and table is 0.25. Find the maximum weight of block A for which the system will be in equilibrium.

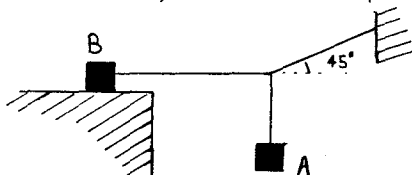


Figure 3: Another Statics Problem

This problem is more complex than the previous one, in that solving for the mass of the hanging particle requires that the program successively

introduce a number of new quantities not mentioned in the problem statement, namely the tensions in each of the strings, and the reaction between the table and Block B. This makes full use of the general problem solving strategies. Our program employs a backward goal-directed search strategy rather than the forward chaining strategy used by Jill Larkin. However, the actual qualitative reasoning involved in the problem solution is none-the-less similar.

A related domain that was investigated was that of spring problems. Solving such problems requires the addition of several new formulae, e.g. Hooke's law, along with rules for applying them. Hooke's law should not be used for inelastic strings. Therefore, a predicate 'elastic' is also defined, and the rule for applying Hooke's law includes a condition that the law be used only for elastic strings. Another formula specifies that the extension of an inelastic string is zero. A typical problem of this sort, taken from [1] p 153, is as follows.

Two springs AB and BC are joined together end to end to form one long spring. The natural lengths of the separate springs are 1.6 m and 1.4 m and their moduli of elasticity are 20 N and 28 N respectively. Find the tension in the combined spring if it is stretched between two points 4 m apart.

In the process of solving the problem, Mecho had to create several new unknowns: the tension in spring2, and the extensions of both springs.

Mecho can now solve quite a wide range of problems involving rods, strings and particles. The expanded version of this paper [5] contains examples of other problems and the assertions representing them.

### Conclusions

In this paper we have described a number of statics problems that Mecho can now solve. There are certainly problems that cannot be solved, either because the program's inference rules are inadequate (for example, there is not enough geometric knowledge), or because the problem solving strategies are inadequate (many situations are not covered, e.g. heavy strings and composite objects). The program's reasoning about its strategies could also be improved -- impossible problems, for example, should be rejected as such without trying all possible solution paths!

Nevertheless, we have been greatly encouraged by the ease with which statics problems have been incorporated into the program. The following features of the program have played an important part in making this possible:

- The modularity of the assertions and inference rules.
- The uniformity of certain underlying representational tools (such as the POINT and LINE relations), which took care of a central core of relations that needed to be represented.
- The decoupling of the problem representation from the rules of the problem solver by a sophisticated inference system. Since the program is willing to perform complex inferencing from the input assertions, it is able to bridge the gap between what it is

told and what it needs to know.

- The general method for solving for quantities by using formulae which relate these quantities to other quantities in the problem. New formulae (such as Hooke's law) can be simply "plugged in".

The generality of these approaches has enabled us to successfully extend Mecho to solve problems from a new domain.

#### Acknowledgements

We would like to thank Torbjon Wikstrom, who also worked on the extensions described in this paper. The Mecho project is supported by SRC grant No. GR/A 57954. Alan Borning is supported by a NATO Postdoctoral Fellowship from the National Science Foundation, USA.

#### REFERENCES

- [1] Bostock, L. and Chandler, S.  
Applied Mathematics.  
Stanley Thornes, 1975.
- [2] Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R. and Palmer, M.  
Solving Mechanics Problems Using Meta-Level Inference.  
IJCAI, Tokyo, 1979.  
Also available from Edinburgh as DAI Research Paper No. 112.
- [3] Bundy, A. and Welham, B.  
Using meta-level descriptions for selective application of multiple  
rewrite rules in algebraic manipulation.  
DAI Research Paper 121, Edinburgh, 1979.
- [4] Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R. and Palmer, M.  
Mecho: A program to solve Mechanics problems.  
DAI Working Paper 50, Edinburgh, 1979.
- [5] Byrd, L. & Borning, A.  
Extending Mecho to Solve Statics Problems.  
DAI Research Paper 137, Edinburgh, May 1980.
- [6] Larkin, J.H. and McDermott, J.  
Re-representing Textbook Physics Problems.  
Carnegie Mellon University, 1978.
- [7] Larkin, J.H.  
Models of Strategy for Solving Physics Problems.  
Carnegie Mellon University, 1979.
- [8] Novak, G.  
Computer understanding of Physics problems stated in Natural  
Language.  
NL30, Dept. Computer Science, Univ. of Texas, Austin, 1976.
- [9] Tyugu, E.H. "A Computational Problem-solver" in Michie, D. et al.  
(ed.), Machine Intelligence 9, 1979.

AUTOMATIC ANALYSIS OF ITALIAN

A. Capelli, G. Ferrari, L. Moretti, I. Prodanof, O. Stock

Istituto di Linguistica Computazionale - CNR - Pisa

ABSTRACT

ATNSYS, an automatic syntactic analyser, has been used for a number of experiments with Italian texts. It is provided with a heuristic mechanism based on probability evaluation. A 'verb frame' representation is introduced. Both these aspects are discussed and the results of our experiments are considered.

1. Natural language processing seems to be in a phase of experimentation, often more concerned with performance problems than with the proposal of new models.

Our own contribution deals with experiments on the analysis of complex Italian sentences, taken from descriptive and narrative texts. This type of text was chosen as it offers a large sample of linguistic phenomena. Limitations in the parsing techniques and the linguistic theories assumed, thus, tend to be evidenced. We have used an ATN parser, because of its characteristics of modularity, perspicuity and flexibility [Woods, 1970]. Moreover, the capability of ATN to maintain the original distribution of the elements of the input string makes it a suitable model for the analysis of rather free order languages like Italian. It also offers a model of the processes of sentence comprehension, as perceptive strategies can be formalised in the arcsets of the network [Kaplan, 1972].

2. Our experiments aim at the improving of the search in the network both by the use of heuristics and by the strengthening of conditions on the arcs based on lexical expectations and constraints. We have used ATNSYS, an ATN parser designed and implemented by O. Stock [Stock, 1976] and written in MAGMA-LISP, a dialect of LISP developed at the "Istituto di Elaborazione dell'Informazione" of the Italian National Research Council (CNR), Pisa [Asirelli et al., 1975].

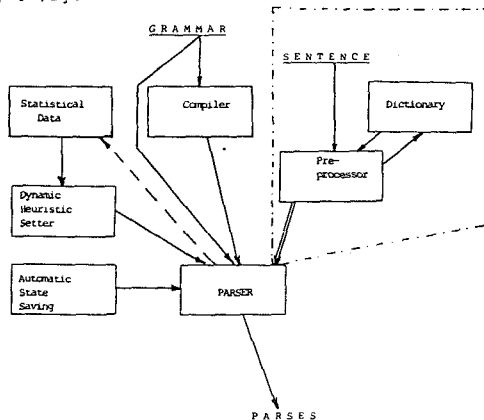


Fig. 1



The system, shown in Fig.1, has the following characteristics:

a) The lexical component (dictionary and preprocessor) is kept separate from the rest of the parser. This permits the use of larger dictionaries (at present we are adopting a dictionary of about 20,000 forms).

b) The grammar entered to the system as input can be partially or totally compiled in LISP functions. Each state is coded as a LAMBDA containing, in sequence, the series of arcs controlled by the non-deterministic structure. Compilation may be only partial as during the experiments certain parts of the grammar may be consolidated before others. As for the treatment of non-determinism ATNSYS takes advantage of ND-LISP [Montangero et al., 1976], a facility of MAGMA, compilation here is less complex than as described in [Burton and Woods, 1976].

c) ATN is a strongly non-deterministic model in which the arcs exiting from each state are tried in the order in which they are found. In ATNSYS, search is "depth-first" and the parser interacts with a heuristic mechanism which orders the arcs according to a probability evaluation. This probability evaluation is dependent on the path which lead to the current node and is also a function of the statistical data accumulated during previous analyses of a "coherent" text.

The mechanism can be divided into two stages. The first stage consists in the acquisition of statistical data; i.e. the frequency, for each arc exiting from a node, of the passages across that arc, in relation to the arc of arrival: for each arriving arc there are as many counters as there are exiting arcs.

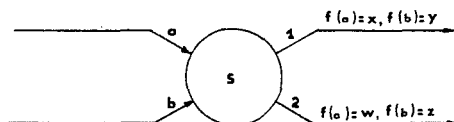


Fig.2

In this way, in Fig.2 arc 1 has been crossed  $x$  times coming from  $a$  and  $y$  times coming from  $b$ . In the second stage, during parsing, in state  $S$ , if coming from  $a$  and  $w > x$ , arc 2 is attempted first.

3. A group of experiments was aimed at verifying the statistically based heuristic mechanism. The first experiment consisted in analysing a coherent text, a page of a science fiction novel. An accumulation of descriptive statistical data might provide valid material to be used in the formulation of a model for linguistic analysis, taking also into account performance aspects. (We refer to the hypothesis by which the order of the arcs in ATN can adequately represent the set of sentence comprehension strategies in the order of increasing complexity).

The introduction into the parser of this heuristic mechanism has given good results. We have achieved both an appreciable reduction in the execution time and in the computational load necessary for parsing. The computational load is measured in terms of the length of the path followed through the

network including the number of blind alleys. Some average results relating to the strategies for the text specified above are shown in Fig.3.

Sentence	Original network			Reordered network					
	time I parse	fails	arcs	time I parse		fails		arcs	
1	5,335	92	231	5,444	+0,109	84	-8	232	+1
* 2	3,866	77	185	3,699	-0,167	60	-17	165	-20
3	4,956	88	212	4,600	-0,156	70	-18	188	-24
* 4	3,910	75	186	3,825	-0,085	67	-8	185	-1
5	2,455	47	105	2,289	-0,166	36	-11	101	-4
6	2,454	46	105	2,242	-0,212	36	-10	102	-3
7	3,652	72	167	3,429	-0,229	55	-17	146	-21
8	4,087	71	187	3,958	-0,129	63	-8	185	-2
9	6,100	135	332	7,946	-0,154	106	-29	310	-22
10	1,891	30	82	2,917	+1,026	54	+24	125	+43
11	1,620	33	81	1,769	+0,149	30	-3	61	-
* 12	6,370	107	273	11,480	+5,11	267	+160	553	+280
13	4,154	78	189	4,465	+0,311	68	-10	188	-1

Fig.3

Columns 1 to 3 give respectively the times, number of failed arcs, and number of traversed arcs when the text is parsed with the original grammar. Columns 4 to 9 show the same figures referred to a reordered grammar together with the differences. Lines stressed by an asterisk indicate those sentences in which the semantically acceptable parse appears as first only after reordering.

A formal analysis of the statistical data in relation to a grammar, with the introduction of significant mathematical measures, may constitute a unitary method for measuring linguistic facts. In this sense, this collection of statistical data may lead, for instance, to the identification of a particular syntactic style in relation to a general grammar, and to give a more precise concept of the performance adequacy of a grammar for a text [Ferrari and Stock, 1980].

4. Another group of experiments has been aimed at verifying a linguistic model which, while taking advantage of the possibilities offered by the structural representation, also develops the functional relationships of the components of a sentence. Functional labels were first given to the arguments belonging to the "verbal frame", i.e. the predicative structure that is assumed to be associated to every verb in the dictionary. Then, when possible, labels were associated to the other arguments of the sentence. The association of labels to arguments is achieved by operations of intersection upon lexical information of the following kind:

a) the verbal frame (its argumental structure) and the requisites (selective features) that the "heads" of the NP's must have in order to enter into that structure. This information is represented in the form

(<verb> (<preposition> (<selective feature> (<functional label>)+))\* where the preposition is the one which on the surface realises the particular

argument;  
b) the specifications associated to each preposition for the functions that it can realise as a surface element, and the relative selective features in the form

(<selective feature> <functional label>)+

c) the list of selective features which can be associated to every noun.

The analysis of the verb frame works as follows:

- if the arguments precede the verb: i) a list of functional labels is assigned by intersection of b and c. Given for instance the preposition "A" described as ((LUOGO MOTOA) (PERSONA DATIVO)) and a noun described as LUOGO, the function MOTOA is selected; ii) the verb verifies these hypothesized labels. Thus the verb GIUNGERE having in its frame (A (LUOGO MOTOA)) accepts the label selected above.

- if the arguments follow the verb: i) the verb tentatively indicates a list of hypotheses by the described intersection operation; ii) these are verified by the presence in the surface of a preposition and by the features of the head.

The mechanism we have described falls entirely into the classical ATN formalism. As we need information coming from different points of the sentence, we have heavily used SENDE and LIFTR, actions of ATN which sidely pass information from one level to another. This corresponds to a model of analysis which involves a predetermination of the configuration where the information will be processed. We think it would be more correct to ensure a model in which information are retrieved when needed. We are now working on this topic.

Figures 4 and 5 show parses with functional labels assigned according to the frames of the main verbs: GIUNGERE ((A (LUOGO MOTOA)) and DIRE ((A (PERSONA DATIVO))).

(IL PE DICEVA COSA BRUTTE ALLE SIGNORINE A TAVOLA)

```

S  TIPO  DCH
  GN  DEF  ART  IL
      N    RE
      GE  MAS
      NU  SING
  AUS  TEMPO PAS
      MODO IND
  GV  V  DIRE
      GN  N  COSA
          ATTR GA  AGG BRUTTO
          GE  FEM
          NU  PLU
  GP  PREP A
      GV  N  SIGNORINA
          GE  FEM
          NU  PLU
          FUNZIONE DATIVO
  GP  PREP A
      GN  N  TAVOLA
          GE  FEM
          NU  SING
          FUNZIONE MOTOA
          STATIN
  
```

Fig.4

# CAPELLI-5

(ERA IL TERZO ANNO DELL'ULTIMA DECADE DEL VENTESIMO SECOLO. QUANDO NEL MONDO IL CORSO DELLA CIVILTÀ GIUNSE A UN'IMPROVVISA FINE)

PARSE: PARTIAL TIME 5 . 410 SECONDS.

S COORD  
 S TIPO DCH  
 GN IMPLICIT  
 AUS TEMPO IMPERF  
 MODO IND  
 GV VM COPULA V ESSERE  
 PRED GN DET ART IL  
 ORD TERZO  
 N ANNO  
 GE MAS  
 NU SING  
 GP PREP DI  
 GN DET ART IL  
 N DECADE  
 ATTR GA AGG ULTIMC  
 GE PEM  
 NU SING  
 SP PREP DI  
 GN DET ART IL  
 ORD VENTESIMO  
 N SECOLO  
 GE MAS  
 NU SING

S TIPO TEMPORALE  
 GN DET ART IL  
 N CORSO  
 GE MAS  
 NU SING  
 GP PREP DI  
 GN DET ART IL  
 N CIVILTÀ  
 GE PEM  
 NU SING  
 AUS TEMPO PAS  
 MODO IND  
 GV V GIUNSE  
 GP PREP IN  
 GN DET ART IL  
 N MONDO

GE MAS  
 NU SING  
 GP PREP A  
 GN DET ART UN  
 N FINE  
 ATTR GA AGG IMPROVVISO  
 GE PEM  
 NU SING  
 FUNZIONE MOTOA  
 STATIN

Fig.5

REFERENCES

- Asirelli P., Lani C., Montangero C., Pacini G., Simi M., Turini F., "MAGMA-LISP Reference Manual", NT C75-13 IEI-CNR, Pisa, 1975.
- Burton R.R., Woods W.A., "A Compiling System of Augmented Transition Networks", COLING 76 Preprints, Ottawa, 1976.
- Cappelli A., Ferrari G., Moretti L., Prodanof I., Stock O., "An Experimental ATN Parser for Italian Texts", Draft, Pisa, 1977.
- Cappelli A., Ferrari G., Moretti L., Prodanof I., Stock O., "Parsing an Italian text with an ATN parser", NT, LLC-CNR, Pisa, 1978.
- Ferrari G., Stock O., "Strategie selection for an ATN Syntactic Parser", The 18th annual meeting of the Association for Computational Linguistics, Philadelphia, 1980
- Kaplan R., "Augmented Transition Networks as Psychological Models of Sentence Comprehension", in Artificial Intelligence, 3, 1972, pp. 77-100.
- Montangero C., Pacini G., Turini F., "ND-LISP Reference Manual", NT C76-3, IEI-CNR, Pisa, 1976.
- Stock O., "ATNSYS: un sistema per l'analisi grammaticale automatica delle lingue naturali", NI B76-29 IEI-CNR, Pisa, 1976.
- Woods W.A., "Transition Network Grammars for Natural Language Analysis", in Communications from ACM, 13-10, 1970, pp. 591-602.
- Woods W.A., "An Experimental Parsing System for Transition Network Grammars", in Rustin (ed.), Natural Language Processing, New York, 1973.
- Woods W.A., Kaplan R., Nash-Webber B., "The Lunar Sciences Natural Language Information System: Final Report", BBN Report 2378, Cambridge (Mass.), June, 1972.

ANALYSING ENGLISH TEXT:  
A NONDETERMINISTIC APPROACH WITH LIMITED MEMORY.

A.W.S.Cater, Computer Laboratory, University of Cambridge  
 Present address: Computer Science Department,  
 University College Dublin  
 Belfield, Dublin 4, Eire.

### Introduction

The paper describes a sentence analyser which operates as part of a story understanding system. In broad outline, the system has three components: the sentence analyser, which produces analyses into a conceptual representation, i.e. a case-oriented representational language akin to Conceptual Dependency; an inference mechanism, which answers questions, and also, whilst producing inferences, resolves pronouns; and a sentence generator, which can express these inferences and the answers to questions, and can also paraphrase the individual sentences of the story.

The larger system embodies a novel approach to the integration of low-level and high-level knowledge, in which script-like and plan-like structures are indexed by, and thus subordinated to, the structures which encode and apply mundane knowledge; and also has a unified approach to the problems of question answering and pronoun resolution.

The analyser is of specific interest because it combines semantic processing with low-level syntactic processing, but has no need for an overall sentence grammar; and, whilst operating in a non-deterministic fashion, it retains plausibility because of the limited number of partial analyses it needs to remember.

The first section presents the principles which guided the design of the analyser, sketching the derivation of these principles from earlier work on analysis. The second section presents the basic mechanisms used to put these principles into effect, presenting a sample dictionary entry, and showing how only small numbers of partial analyses need be kept. The third section shows the application of these mechanisms to specific linguistic phenomena: conjunctions, relative clauses, lexical and structural ambiguity.

### 1) Principles

The principles on which the analyser is based were derived mainly from consideration of earlier work on analysis by Riesbeck[74], Wilks[75] and Woods[76]. Woods adopted the classical approach, deriving semantic readings from prior syntactic analyses, whilst both Riesbeck and Wilks attempted to parse directly into a semantic representation.

Woods used the now-familiar ATN mechanism to produce syntactic analyses of sentences; these were then checked semantically. The ATN formalism allows great flexibility in the handling of syntactic structures, and is well suited to the task of searching the space of possible analyses; however it is not an ideal vehicle for semantic analysis, since this depends so strongly upon the actual words in the sentence. The strength of the ATN is in capturing regularities; syntactic regularity is commonplace, semantic regularity is not.

Riesbeck's analyser was driven by requests - test/action pairs - which constituted the definitions of the words in the dictionary. In rough outline, as a word was taken from an input sentence, as many existing requests as possible were satisfied; when no more of these could be satisfied, any requests attached to the new word were added to the list of active requests (possibly for immediate use). When no more requests could be satisfied, the next word was considered.

This scheme was purely left-to-right, and used virtually no syntax; this caused some problems, notably with detecting the end of a noun phrase. Problems also arose because there was no means of destroying requests which were no longer needed, and because there was no mechanism for delaying the structural decisions which must be made in processing embedded clauses. It has been suggested (Eisenstadt[79]) that Riesbeck's parser may be reformulated as an ATN. Such a reformulation is indeed shown to be possible; however the spirit of Riesbeck's work is very different, especially with respect to nondeterminism and overt syntactic processing. The analyser discussed later can similarly be viewed as an ATN, or, more helpfully, as two ATNs in series: for instance, it has mechanisms which parallel the PUSH/POP operations performed in an ATN. Overall, it may be regarded best as a dynamically constructed ATN, further augmented with preferential ordering of arcs. It is in spirit far removed from ATNs as traditionally conceived.

Wilks's analyser, based upon preference semantics, first broke a sentence into fragments, and then mapped bare templates corresponding to meaningful message schemas onto these fragments. The templates were triplets of semantic primitives; the mapping process involved comparing these with the head elements of the formulas which defined word senses. When a successful match was found, the formulas for the words in the fragment were substituted in the bare template. The formulas also encoded preferences for semantic features of contextually related items; these preferences could be used for choosing word senses, making structural decisions and resolving (some) pronouns.

Wilks's notion of preference seems a valuable idea; however his implementation with repeated scans of many alternative analyses is highly implausible as a model of human language comprehension.

More recently, Marcus [79] has proposed a deterministic model of syntactic recognition for English sentences. Though this work was contemporary, and hence did not form part of the background to this work, there are some superficial similarities. Marcus finds that grammar rules may be written in a form which permit a syntactic parser to operate deterministically. This is achieved largely by permitting a 3-deep lookahead; thus some degree of nondeterminism is implied but hidden. As will be seen in section 2.3, the analyser discussed here has explicit limited nondeterminism. This analyser is markedly different in other respects, notably the parsing mechanism and the semantic analysis it provides, contrasting with the syntactic analysis provided by Marcus's system.

From consideration of the various merits and disadvantages of the earlier analysers, the following principles emerge:

i) The information to guide analysis of a sentence should be derived from the definitions of the words in the sentence; this should be packaged in a manner similar to requests - that is, test + action pairs.

ii) The requests used should be manipulable, capable of being removed when no longer applicable, and clean, not modifying the definitions of other words.

iii) The role of syntactic processing should be to locate constituent groups of words which may be treated as building blocks for a larger sentence representation: neither a request mechanism, nor any keyword-based algorithm, should be expected to perform this task. Nor should an explicit syntactic analysis of whole sentences be sought, since a semantic representation naturally encodes that structural information which is important for comprehension.

iv) The analysers progress through a sentence should be predominantly left-to-right, but should not have to employ cumbersome stratagems when a given path of analysis proves fruitless.

v) There should be a mechanism for specifying and applying preferences.

An analyser has been constructed which adheres to these principles.

a) It uses an ATN parser to locate the basic building blocks of sentences, namely simple noun groups, verb groups, prepositional phrases, commas, *wh*-forms and conjunctions.

b) It uses a set of requests, classified into six types, and associated both with individual words in the dictionary and with standard operations, such as relative-clause processing.

c) It uses a control mechanism for applying these requests, which follows a preference-based strategy for controlling a nondeterministic analysis, and requires only a small memory for partial analyses.

## 2) The basic components.

### 2.1) The ATN

An ATN is used here to isolate syntactic constituents of a sentence, rather than to build a structure which explicates the syntactic relationships between the constituents. The constituents it recognises are simple noun phrases, verb groups, prepositional phrases, conjunctions and *wh*-forms. These constituents are recognised by subnets which simply find syntactic constituents. This contrasts with the use of subnets in the PLANES system (Waltz[73]) for finding semantic constituents.

Verb groups are not treated conceptually by the ATN, but merely analysed in terms of main verb, tense, voice, form and negation. Thus "was not being given" would be represented by the ATN as the constituent

(VP (tense PAST) (form PROG) (voice PASSIVE) (neg T) (verb GIVE))

Simple noun phrases - possibly containing determiners, adjectives, possessives and quantifiers - are accepted by the NP subnet of the ATN. Their representation at this stage closely parallels the ultimate conceptual representation, though disambiguation of polysemous nouns and the attachment of modifiers are performed at a later stage. Prepositional phrases are treated similarly.

The treatment of conjunctions and commas by the ATN is trivial, since its task is merely to locate and identify constituents. Unknown words, or words which fit into no higher structure (such as dangling prepositions) are marked as isolated words.

Since some words have several syntactic categories, there are occasions where it is possible to find more than one constituent in one place. It is then appropriate to build a constituent tree - akin to a lattice - to represent the possible constituents.

This use of an ATN has two principal effects. Firstly, it provides a convenient mechanism for treating words with more than one syntactic category, especially since frequently one category only will be considered because of the influence of the surrounding context. Secondly, it provides a motivated and isolable fragmentation scheme, which can assist the identification of the key elements of a sentence (verb and head nouns). This may be contrasted with Wilks's fragmentation algorithm, which was based largely upon key words, and which could not be separated from the rest of his system. It may also be contrasted with Riesbeck's approach, which relied upon complex requests associated with determiners, adjectives and auxiliary verbs.



## 2.2) The requests

A set of requests, each having one action part and two predicate parts, is used to guide the analysis of sentences. The first predicate determines whether the action part is to be executed, while the second determines whether the request should be discarded if it is not used. These two predicates are named respectively the MAIN and the KEEP predicate. The requests are used to specify what should be expected, in both syntactic and semantic terms, and to determine the utilisation of such elements when they are found.

The requests come from two sources: firstly from the definitions of words in the dictionary, principally from verbs and conjunctions; and secondly from places in the program where a standard situation has been recognised, for instance, the beginning of a sentence, relative clauses, or sentences of the form "X do Y to do Z".

The requests are classified into six types, each corresponding to a step in the cycle of processing a constituent. The characteristics of the six types which make it useful to differentiate between them will be explained in section 2.4 below, when the control mechanism and theories, the objects the mechanism manipulates, have been described. Briefly, the six types are:

- i) USE - use a constituent found by the ATN; if a constituent can be used, any program associated with the head word is executed. In the case of verbs, this leads in the requests which will process the rest of the sentence.
- ii) TIMING - provides for a specified delay, in terms of constituents seen, before performing some action. (These requests are very rare.)
- iii) TRIVIAL - the programs associated with verbs normally load requests of this type, whose purpose is to build up structures, insert time information and negation, and to add further requests.
- iv) USE-REG - handle constituents which have been placed in registers, a form of local storage. Usually this is just for placing the subject into the structure built by a verb sense.
- v) UNEMBED - insert the structures corresponding to embedded clauses, relative clauses and prepositional modifiers, into a predetermined place in the higher-level clause. Also resets structure, registers and requests.
- vi) EMBED - Save existing structure, registers and requests, and prepare to process an embedded clause of some kind.

## 2.3) The control mechanism: theories; and memory limitations.

Based on the notion of preference, a control mechanism for the application of requests has been developed which, even in a non-deterministic environment, is found to need only a small finite memory for partial analyses.

These partial analyses, I call theories: when tested these will usually generate new theories. Each has seven components:

- 1) A numeric score which may be modified by the satisfaction or violation of preferences, but also by behind-the-scenes activity, such as unembedding (see section 2.6). This score is a simple integer, and in practice lies roughly in the range -10 to 50.
- 2) A partial semantic structure.
- 3) A set of requests.
- 4) An indication of the request-class to be tried.
- 5) The remaining constituent-tree (product of ATN processing)
- 6) A current constituent (if the request-class is USE)
- 7) A set of registers, a form of local storage

The basic mechanism is simple, and is related to the Graph Traversal (Doran[65]), Woods's "theories" (Woods[77]) and the RNL scheduler queue (Bobrow et al[77]). The theories are maintained in order of score, and the highest-scoring theory is chosen; requests of the right class are selected and tried. New theories, with modified structures, registers etcetera, are generated; in making new theories, the used requests are discarded, as is any selected request whose KEEP predicate fails. The generation of a new theory requires specification of a request class. This follows the pattern shown in figure 1. GET does not label any requests at all, but is the stage of processing where a new constituent is found, usually from the constituent-tree returned by the ATN (but see section 3.2).

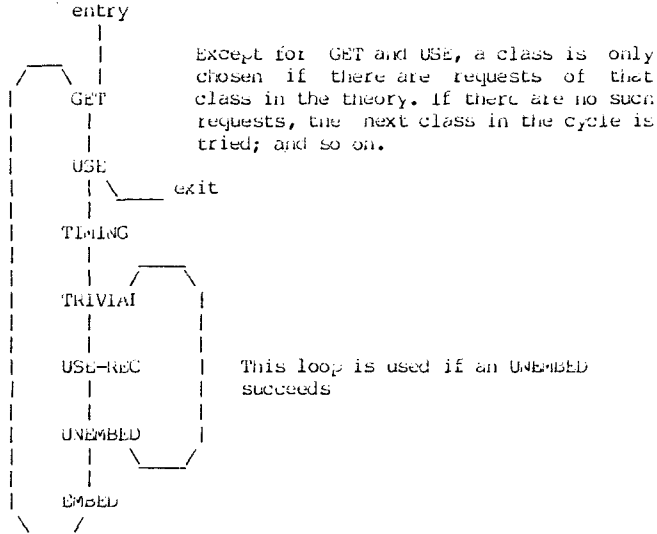


Figure 1: The constituent processing cycle

The loop shown from UNEMBED to TRIVIAL applies only when an UNEMBED has been done, permitting multiple unembeddings without consuming more constituents, and also permitting an UNEMBED to influence TRIVIAL or USE-REC requests directly (though this is seldom needed). When no UNEMBED is done, EMBED is selected next; or in the absence of EMBED requests, GET.

The operation of the mechanism as described is modified in one particular case. I have said that a theory is selected from the top of the stack of theories, on the basis of preference. In fact, all theories with the same preference as the top theory are separated from the stack, and processed in turn. This averts problems where one theory generates new theories with a slightly higher preference, causing the next theory, intrinsically of the same value, to be obscured.

The mechanism is also usefully restricted by taking account of an empirical fact which, I believe, also has implications for the relation between memory and linguistic faculties. It has been observed that, despite the apparently non-deterministic operation of this analyser, the depth of its stack of theories need only be small; the size of stack needed compares with the postulated size of human short-term memory. Limiting the size of this stack to seven theories, by simply discarding any theories which fall off the end, actually improves the performance of the analyser for complex sentences; and

in fact the largest stack needed for any sentence which has been processed using my test vocabulary has been only 5 elements long. This result is an empirical observation, based on the successful analysis of several hundred diverse sentences, and does not depend on any fine-tuning of the preference manipulations. (It may turn out that the addition of large numbers of highly-polysemous words may require adoption of a wait-and-see strategy, such as those used by Hayes[77] and Marcus[79]. However, though the dictionary does contain a number of polysemous words, no problems of this nature have been encountered.)

Marcus[79] proposes a "determinism hypothesis", and describes an analyser which also has a limited memory. In his system, the memory consists of a stack of partial syntactic structures, and the rules of the grammar are permitted to inspect only a small number (usually 3) of these structures. However, Marcus does admit that this analyser is unable to deal with complex sentences, and must seek advice from some other, more powerful, component of the larger system. It seems to me that this is tantamount to accepting that grammar rules are inadequate when so constrained; and his analyser does not attempt to provide a semantic analysis either.

This concept of a limitation on memory leads to a plausible solution of the question "When is a sentence structurally ambiguous?", if we include the hypothesis that alternative readings violate no more noun-feature preferences than does the first reading found.

#### 2.4) The stages of processing.

The stages of processing are naturally described in terms of the classes of request to which they correspond, since these classes, with the exception of GET, are in fact based on processing stages. GET is discussed more fully in Section 3.2, where the treatment of questions and relative clauses is examined; for the moment we may simply note that GET retrieves a constituent from the tree given by the MAIN; for each retrieved item, a new USE theory is constructed, with the appropriate current constituent and constituent-tree.

##### i) USE

USE requests have a MAIN predicate which normally specifies the syntactic class expected. Their action parts, in addition to specifying the use to be made of the constituent they accept, will often contain preference manipulations based upon the features associated with nouns: for this purpose, features are arranged in a hierarchy.

For each sense of the head word of a constituent, each USE request is applied: and for each successful application, where the MAIN predicate succeeds, a new theory is created: the class-*ta*; being determined according to the rules given in Section 2.3 above. The request applied, and any other requests of the same type whose KEEP predicate fails, are removed. (As noted earlier, this is true of all request classes.) Thus USE requests may branch in two different ways; for different senses of words, and for different uses of the constituent.

If a constituent is encountered for which no request succeeds, it is tested to see if it can be *trapped*; *traps* - which handle conjunctions, some commas and certain higher-level structures - are discussed in section 3.3.

##### ii) TIMING and TRIVIAL

TIMING and TRIVIAL requests are both simple, causing the creation of only one new theory. Any number of such requests may be present; if they are present, their MAIN predicate determines whether the action should be performed, but the KEEP predicate is immaterial. TIMING requests are rare, being used only for a few conjunctions. TRIVIAL requests are extremely common, being the vehicle for the introduction of other requests from the dictionary. These requests are often used to build conceptual structures, add time and negation information, and load the other requests which will analyse the remainder of a clause.

iii) USE-REG

USE-REG requests take a constituent, usually the subject, from temporary storage in a register. Since the constituent has been placed there by a USE request, it corresponds to only one sense of the head noun. However, it sometimes happens that more than one USE-REG request exists, and in this case all the available USE-REG requests are applied, producing one new theory each. This situation occurs mostly for the passives of indirect-object verbs, where the subject may fill either the slot corresponding to the direct object or the indirect object in an active sentence.

iv) EMBED

EMBED requests deal with embedded clauses, relative clauses and prepositional phrase modifiers. An EMBED operation is usually optional, so one theory will result from this stage which has not been embedded, and one or more which have. These theories will have the current state of the registers, the current set of requests, and the current semantic structure, all saved on an otherwise blank list of registers. When this has been done, the actions of the applicable EMBED request are executed. These will always add new requests to deal with the embedded clause, and will often set a few registers. For the latter purpose, there is temporary access to the saved registers, which is necessary to handle the tensing structures which occur in embedded clauses, and those clauses which borrow the subject from a higher-level clause.

An obvious point is that an analysis cannot be allowed to terminate while still in an embedded state.

v) UNEMBED; and constraints

UNEMBED requests are the converse of EMBED. There can only be one UNEMBED request present at any time. Requests of this class restore the registers, requests and semantic structure before their action parts are executed. However, the MAIN predicate on the UNEMBED request is not regarded as sufficient to permit an unembedding. A further mechanism, the specification of constraints, must be considered. A general constraint on unembedding is that there must be a semantic structure which has been built. Certain verbs, however, specify further constraints - saying such-and-such a request must be used. If any of these constraints have not been satisfied, no unembedding can take place. ( This mechanism applies also to accepting the end of the sentence, which is seen as a special case of unembedding. )

An UNEMBED request is also optional; thus one theory will always be created which corresponds to a failure to unembed.

2.5) The dictionary entries

Requests can be fairly complex objects, involving two predicates and an arbitrary set of actions, and their full specification on each verb definition would be a tedious and repetitive process. Advantage can be taken of the repetitiveness of simple requests however, by defining a set of macros with which such requests can be built up. Several such macros have been provided, greatly easing the definition of new verbs. Figure 2 illustrates some of these.

The macros available include:

PLACE-SUBJ    for using the subject: generates a USE-REG request  
 PLACE-OBJ    for using a noun-phrase: generates a USE request  
 PLACE-PP     for using a prep-phrase: generates a USE request

PLACE-CLAUSE, PLACE-SUBJECTLESS-CLAUSE, PLACE-OBJECTIVE-CLAUSE  
 all look for various sorts of embedded clause.

```

(DISPI EASE
((requests
  (ADDREQ
    (TRIVIAL
      T
      NIL
      (BUILDS
        ((CAUSE
          (ANTECEDENT
            (EVENT (ACTOR DUMMY-HUMAN1)
              (ACT DO)))
          (RESULT
            (STATE (STATENAME JOY)
              (THING DUMMY-HUMAN2)
              (VAL (I OVERBY 2)))))))
        (TIME-PLACE)
        (VERBS-DUMMIES
          ((CAUSE ANTECEDENT EVENT ACTOR) DUMMY-HUMAN)
          ((CAUSE RESULT STATE THING) DUMMY-HUMAN))
        (COND
          ((ACTIVE)
            (PLACE-SUBJ (CAUSE ANTECEDENT EVENT ACTOR) ANYTHING)
            (PLACE-OBJ (CAUSE RESULT STATE THING) BEAST)
            (PLACE-SUBJECTLESS-CLAUSE
              BY
              (CAUSE ANTECEDENT)
              (AND (FORM-IS PARTICIPLE)
                (TENSE-IS PRES))
              subj)))
          ((PASSIVE)
            (PLACE-SUBJ (CAUSE RESULT STATE THING) BEAST)
            (PLACE-CLAUSE
              BY
              (CAUSE ANTECEDENT)
              (AND (TENSE-IS PRES) (FORM-IS PARTICIPLE)))
            (PLACE-P
              BY
              (CAUSE ANTECEDENT EVENT ACTOR)
              ANYTHING)
            (PLACE-SUBJECTLESS-CLAUSE
              BY
              (CAUSE ANTECEDENT)
              (AND (PASSIVE) (FORM-IS PROG))
              subj)
            (PLACE-SUBJECTLESS-CLAUSE
              AT
              (CAUSE ANTECEDENT)
              (AND (PASSIVE) (FORM-IS PROG))
              subj)))))))

```

Figure 2: Dictionary entry for DISPIEASE

In this example, there is only one definition for the verb, ie. only one sense is being considered. The part of the definition which is to be executed is labelled by 'requests', with the function ADDREQ used to add a request. The one request concerned here is of class TRIVIAL, and its main predicate is T (TRUE): therefore it will succeed. Its actions are:

BUILDS . . . . . to build a semantic structure  
 TIME-PLACE . . . to insert temporal information in a standard way  
 VERBS-DUMMIES. . to create new 'tokens' to fill in parts of the structure:  
 this corresponds to assuming features of objects in default. The  
 destinations within the structure are denoted by the paths  
 (CAUSE ANTECEDENT EVENT ACTOR) and (CAUSE RESULT STATE THING).  
 The request-macros also use paths to determine the destinations of the  
 substructures they create.  
 COND . . . . . the LISP conditional, having here two branches; the  
 selection is made by the functions ACTIVE and PASSIVE, which inspect  
 registers which have been set up at the time the verb group was encountered.  
 The macros build up requests which effectively specify case-frames; these  
 are then fitted by (limited) exploration of possibilities.

In the ACTIVE branch, (PLACE-SUBJ . . . . ANYTHING) adds a USE-REG request  
 which will look at the 'subj' register. This register has been set, by one  
 of the initial requests, to contain a conceptual representation of the  
 subject; this request now inserts that representation into the semantic  
 structure, at the end of the path (CAUSE ANTECEDENT EVENT ACTOR); also it  
 will apply, in this case, the very weak preference for the feature ANYTHING  
 (weak, since ANYTHING is very near the top of the feature hierarchy).

(PLACE-OBJ . . . . BEAST ...) adds a USE request which seeks a noun phrase,  
 preferring the feature BEAST to be present: when this has been found,  
 (PLACE-SUBJECTNESS-CAUSE BY (CAUSE ANTECEDENT) X...X subj) adds a further  
 request, which corresponds to an EMBED. This will only be used if the word  
 BY is located, in which case an embedded clause is expected, which will lack  
 a subject, and whose verb is a present participle. The current contents of  
 the register 'subj' (ie whatever did the displeasing) is to be used again as  
 the subject of the embedded clause. The conceptual representation of the  
 entire embedded clause is to be placed at the end of the path (CAUSE  
 ANTECEDENT), replacing the default DO event.

In the PASSIVE branch, there is again a PLACE-SUBJ macro call, and some  
 PLACE-SUBJECTNESS-CAUSE macros which differ only in the word they expect as  
 a cue. The new items are:

(PLACE-CAUSE BY . . . . X...X), which looks for a complete clause cued by  
 the word BY, whose verb is , again, a present participle. This will  
 correspond to sentences like "Fred was displeased by Mary going to the  
 shops"

(PLACE-PP BY . . . . ANYTHING) creates a USE request which looks for a  
 prepositional phrase, with the preposition BY. This will correspond to  
 sentences like "Fred was displeased by Mary"

## 2.6) Preference manipulations

Preference, the numeric score associated with a theory, is important because  
 it provides the basis for the selection of theories in a nondeterministic  
 environment. It has been noted above (Section 2.3), in connection with the  
 observation that only a small stack is needed for theories, that this did  
 not depend upon finely tuning the preference manipulations. There are  
 nevertheless several different reasons for performing such manipulations,  
 most of which are strategic rather than merely tactical in nature.  
 Preference manipulation is required to deal with:

### i) Feature matches and mismatches

This corresponds in spirit to Wilks's preference mechanisms; however,  
 whereas Wilks uses only direct matches between individual semantic  
 primitives (with some primitives acting as classifications), I have found it  
 useful to allow two distinct types of matching operation. Firstly, Wilks's

scheme is generalised by building a hierarchy of primary features, all of which correspond to Wilks's "class" primitives; a request which prefers feature X will accept a noun-sense with feature Y if Y is anywhere on the branch of the hierarchy headed by X. (By "accept a noun-sense", I mean the preference will be applied.) Secondly, there are secondary features, such as MASSY, PROPERNAME and RELATION, which do not occur on the hierarchy, and with which only direct matches are considered.

Normally, these preferences are applied positively: if there is a match, the 'score' of the next theory generated is increased by 2. Simultaneously, all USE requests decrease the score of the next theory by 2. This effectively means doing nothing dramatic when a noun sense has the expected features, but inhibiting its use if the expected features are not present; thus any competing sense of that noun will be preferred (preference only makes sense when dealing with competing alternatives). Further, for those elements to which no preference is applied, such as verb groups and conjunctions, competing theories are given a chance to catch up and increase their 'score' if they can.

#### ii) Embedded clauses, and unembedding.

When an EMBED request may be applied, two theories are created which correspond to embedding and failure to embed respectively, having the same preference. However, the macros from which these requests are usually generated specify a number of USE requests on the lower level, and the first of these will, if successful, increase preference by 2. The effect of this is to stimulate an embedded path which has located a component which it needs.

UNEMBED requests will modify the preference of the unembedded analysis, decreasing it by 3 if some other component is expected, by 1 otherwise. The general disinclination to unembed reflects the observation that ambiguously placed conjunctions are most often taken to conjoin clauses at the most deeply embedded level. For instance, a sentence given by Cullingford[76] (which he quotes from a newspaper) illustrates this point nicely: "A New Jersey man was killed Friday evening when the car in which he was riding swerved off Route 59 and struck a tree". Naturally this heuristic sometimes fails; but it has been observed that wrongly-attended analyses almost invariably cannot accept the next constituent, and so the correct (unembedded) analysis path will be tried quickly.

It should be noted that no explanation is given here for the difficulty of garden-path sentences such as

"The boat floated down the river sank"

"The horse raced past the barn fell"

#### iii) Cues

The presence or absence of determiners is a cue for the disambiguation of nouns, such as JOHN and BILL, having one sense which is a proper noun, and one which is not; possessors provide the same cues. In general, the selection of a noun sense is penalised if these cues are inappropriate. An exception occurs when a relative clause follows, for instance "the Bill to whom I spoke". In such a case, no penalty is applied, and the usual feature-matching procedures must work unassisted.

#### iv) Overwriting

Once a constituent has been placed into the semantic structure, any attempt to replace that part of the structure will be penalised; in practice, this permits verbs to be more easily defined.

#### v) Constituents which cannot be used.

When no USE request will accept a constituent, and no traps can be invoked to use it, the analysis path is effectively terminated. For testing and development purposes, a new theory is created with a score decreased by 25: an attempt to consider this theory is regarded as an error.

### 3) linguistic phenomena: further details

#### 3.1) Treatment of ambiguity

I wish to distinguish three levels of ambiguity, and to present the manner in which the analysis model described above attempts to handle them.

##### i) Multiple senses of a syntactic constituent.

It should be noted that the use of semantic primitives tends to blur some fine sense distinctions.

The disambiguation techniques used differ according to the syntactic category.

Verbs with more than one sense are normally distinguishable on the basis of their case-frames, though occasionally there is an interaction with the features of surrounding nouns. There is one example, even in the small vocabulary of this system, of a sense discrimination which cannot be effected by this technique: this is the choice of GET = FETCH or GET = ACQUIRE, which I must leave as an open problem.

Nouns with several senses are normally distinguishable by their features, but occasionally CUES are useful: for instance, proper names are not normally introduced with determiners. This information is communicated to the overall request mechanism by inspecting CUES in the USE phase; this causes behind-the-scenes preference changes.

Prepositions usually have several senses, but each sense is associated with a particular usage. This is handled by associating appropriate requests with the prepositions in different ways; most commonly, prepositional phrases are located by PLACE-PP macro requests, and the definitions of the prepositions are immaterial. Most prepositions only need individual definitions to handle their use in postmodifiers.

Adjectives may have many senses; the current program does not attempt to disambiguate these, but assumes each adjective has one sense only.

##### ii) Multiple syntactic categories of words.

An explicit syntactic analysis, whether of complete sentences or, as here, of well-formed constituents, goes a long way towards handling this form of ambiguity. In this model, the AN performs this function, but sometimes is unable to determine the correct constituent (as noted in section 2.1). The application of requests, which correspond to expectations, will in these cases make a decision on the basis of surrounding context. If any constituent has multiple senses, these are treated by the same mechanisms as outlined above.

##### iii) Structural ambiguity of sentences.

The analyser presented here is intended as a front-end to an inference mechanism, and attempts to find the most plausible reading of a sentence. A sentence is reanalysed only if the inference component determines that the first reading found is ludicrous. In this case, the analyser will repeat its analysis, but reject the first reading encountered. If the next reading is also found to be ludicrous, the analyser cycles round again until either the inference component finds an acceptable reading, or the analyser can find no more readings.

This is not an acceptable long-term solution to the problems of genuine structural ambiguity; but the discussion in section 2.3 may point the way.

#### 3.2) Postponement mechanisms: questions and relative clauses.



wh-questions and relative clauses both exhibit a superficial transformation of the normal word order, where the implied position of the relativised object or the wh-form is, in traditional terms, a trace. In the analyser described here, such a constituent is held in a register, and is explicitly reintroduced into the set of syntactic constituents handled by the requests. This is done whenever the processing cycle along any analysis path reaches the GEI stage.

GEI also deals with yes/no questions, and with wh-questions not focussing upon the subject, which both split the auxiliary from the main verb group. Sentences like "has it John who annoyed Mary" can be seen as grouped with these if we allow "was" to function as both a fronted auxiliary and the entire verb group.

In section 2.4, the operation of the GEI phase was simply described as taking the next set of syntactic constituents, and creating new theories for each of them. To handle the phenomena just mentioned, GEI must do more than this. The more complex operations it actually carries out are controlled by the settings of registers (local to an analysis path), and include

i) Inserting an auxiliary to make a complete verb group

This is done for yes/no questions and some wh-questions. The auxiliary is taken from the register in which it is stored, and placed at the beginning of the remaining words of the sentence. The substring so formed is reanalysed by the ATN constituent grammar.

ii) Replacing a noun phrase constituent, or prepositional phrase constituent

This is common to relative clause and wh-question processing, and involves simply generating a new GEI theory which must use the constituent. Wh-words in relative clauses are replaced by the selected sense of the relativised object. Examples of the situations where this replacement operation may be done occur in the sentences:

Who did John annoy?      Who annoyed Mary?

The person John annoyed hit him.

The banana which John gave to Mary was rotten.

The person to whom John gave the rotten banana became sick.

Sometimes a relative clause, or wh-question, is deeply embedded. By this, I mean that a further clause is embedded within the relative clause (or question), and the trace of the relativised object is to be found within this embedded clause; for instance, "The car your brother said he was expecting us to tell Jane to buy" (an example from Winograd[72], which he calls downrel). This often happens with verbs, such as 'say', which normally expect a clause optionally introduced by 'that'; however when the embedded clause contains a trace of a relativised object, 'that' seems quite out of place. Consider:

"The banana Fred said the monkey ate" and

"The banana Fred said that the monkey ate"

When such an embedded clause is processed, the semantic structure will indicate the position of the relativised object (or queried object) within itself, and this must be combined with the position of the embedded clause to give a true representation.

It is necessary when processing relative clauses to make sure that quantifiers, if present, are properly scoped. In the representational language used here, this requirement is simply met by placing the semantic representation of the relative clause inside any existing qualifying information.

iii) Combining a noun phrase constituent with a dangling preposition.

This operation is very similar to the straightforward replacement outlined above, but is performed (additionally) if the next word in the remaining part of the sentence is a preposition, and the constituent to be replaced is

not a prepositional phrase (eg "TO AND").

### 3.3) Traps: dealing with the unexpected.

It has been explained that verb definitions provide a set of case-frames, which denote the expected gross syntactic elements of a clause. However, items such as conjunctions are never expected, but will themselves determine how they are to be used. They are equipped with traps, which, like requests, have a predicate and an action.

In the case of conjunctions, the predicate will normally test to see whether a clause can be terminated; the actions will normally

- remove all existing requests except UNEMBED requests
- build a new semantic structure, having the existing structure as a part
- forbid further conjunctions from operating at this level
- Initiate the processing of a new clause, EMBEDDED into the newly-built structure.

The treatment of AND, BECAUSE, BEFORE and AFTER follows this outline. However, BECAUSE, BEFORE and AFTER may also begin a clause; for instance

"because Mary went to the park, John felt unhappy."

These conjunctions therefore have several traps, whose predicates distinguish the particular cases. The action-parts for these senses will consume a comma if it occurs at the end of the first clause.

Similarly, the construction '<clause> to <oo> ...' is handled by a trap; again, there is a test to ensure that the first clause is complete, and other requests are discarded.

Commas, apart from being expected at the end of certain classes of clause, can be handled by a trap. This trap looks at the next constituent, to see if it too will be handled by a trap. Thus conjunctions may always be preceded by commas, as may 'to ...' constructions.

### 4) Conclusion

As noted at various points, the analyser so far implemented is far from being able to handle the full range of syntactic constructions. Its performance for the sentences so far supplied is very effective, and the ease with which its range has been progressively extended has been most encouraging. It is my belief that the system can be further extended, relying only on the existing techniques, to deal with such phenomena as reflexive pronouns, abstract nouns, participial premodifiers and appositive clauses.

### References

Bobrow et al[77]

D.G.Bobrow, T.Wilnojad, and the KIL research group.

Experience with KIL-4; one cycle of a knowledge representation language. in IJCAI5, pp213-222

Cullingford[78]

R.E.Cullingford

Script application: computer understanding of newspaper stories. Research Report 116, Dept. of Computer Science, Yale University.

Doran[65]

J.Doran

An approach to automatic problem-solving.

in Machine Intelligence 1, pp105-123

Eisenstadt[79]

M.Eisenstadt

Alternative parsers for conceptual dependency: getting there is half the fun.  
in IJCAI6, pp236-240

Hayes[77]

P.J.Hayes

Some association-based techniques for lexical disambiguation by machine.  
TR 25, Dept. of Computer Science, Univ. of Rochester.

Marcus[79]

M.P.Marcus

An overview of a theory of syntactic recognition for natural language.  
AI Memo no 531, Artificial Intelligence Laboratory, MIT

Riesbeck[74]

C.K.Riesbeck

Computational understanding: analysis of sentences and context.  
Report No. AIM-238, Computer Science Dept, Stanford University.

Waltz[78]

D.Waltz

An English language question answering system for a large relational  
database.  
CACM, vol 21 no 7 1978, pp526-539

Wilks[75]

Y.Wilks

An intelligent analyzer and understander of English.  
CACM, vol 18 no 5 1975, pp264-274

Winograd[72]

T.Winograd

Understanding natural language.  
Edinburgh University Press

Woods[76]

R.A.Woods

Transition network grammars for natural language analysis.  
CACM, vol 13 no 10 1970, pp591-605

Woods[77]

R.A.Woods

Shortfall and density scoring strategies for speech understanding and  
control.  
in IJCAI5 pp18-26

IJCAI5 = Proceedings of the Fifth International Joint Conference on  
Artificial Intelligence, held in MIT in August 1977

IJCAI6 = Proceedings of the Sixth International Joint Conference on  
Artificial Intelligence, held in Tokyo in 1979.

CACM = Communications of the Association for Computing Machinery

## A RATHER INTELLIGENT LANGUAGE TEACHER

Stefano Cerri  
 Istituto di Scienze dell'Informazione  
 University of Pisa  
 Corso Italia 40, 56100 Pisa, ITALY

Joost Breuker  
 Center for Research into Higher Education (COWO)  
 University of Amsterdam  
 Oude Turfmarkt 149, 1012 GC Amsterdam, Holland

Abstract

A semi-intelligent CAI program for teaching the use of conjunctions in a number of foreign languages is presented. The representation of well known confusions in the use of these conjunctions form the basis of tutorial strategies to correct the student. The program is written as a try-out of DART, an ATN-based system for authoring intelligent CAI lessons on the PLATO-system.

Emphasis is put on the fact that intelligence in CAI is not all-or-none: the degree of intelligence required is dependent on the variances permitted and expected in the student's responses. Misconceptions are one of the major sources of variance. DART is proposed as facilitating the transition between traditional and intelligent CAI.

1. DART and Computer Assisted Instruction (CAI)

DART (Didactic Augmented Recursive Transition network) is an authoring system for CAI on the PLATO-system. PLATO is a CAI dedicated system, using a central computer that is at the heart of a network of up till 200 special terminals. In contrast to other CAI author languages, in DART the representation of subject-matter (data) is separated from the teaching program. This separation of data and program increases the *generative* capacity of CAI programs: several programs may operate on the same data, or the same didactics (program) can be used for different domains of knowledge, represented by different data-bases.

Moreover, DART is designed for *intelligent* CAI. A major obstacle in traditional CAI is the restricted nature of the processing of the students response (cf section 3). In DART the development of intelligent programs is facilitated by the use of two well tested conceptions from AI: semantic networks for the representation of the structure of the subject-matter, and ATN's (Augmented Transition Network) for the teaching program. ATN's are not only extremely suited for parsing student input, but also for planning or controlling the initiatives in the tutorial dialogue (cf Brown e.a., 1977). Although ATN's have a bias for top-down processing, which may lead to strongly teacher oriented tutorial strategies, DART does not exclude bottom-up procedures.

2. A second language learning experiment2.1. The use of conjunctions

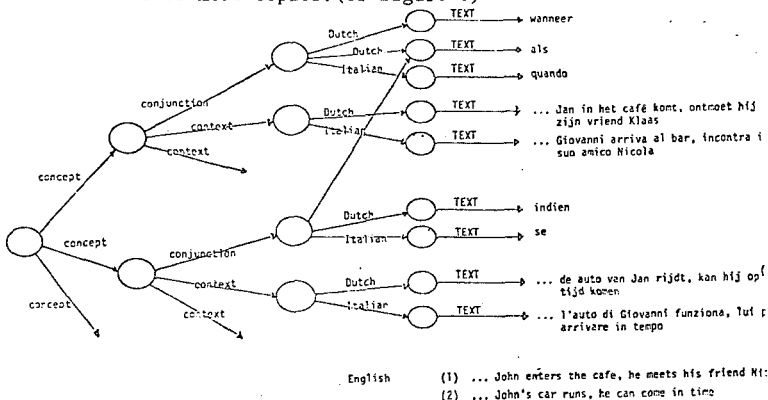
The first operational program written in DART teaches the conjunctions of subordinate clauses in a foreign language (at present: Dutch, English, French and Italian). It is in fact a try-out in a domain which seems at first sight ill suited for a demonstration of intelligent teaching. In traditional CAI foreign word teaching exemplifies "dumb" drill and practice programs, which seem to involve no more than simply putting new labels to old concepts. We have opted for this domain, however, because some well known and tenacious errors can be attributed to conceptual confusions.

The meaning of conjunctions is complex and their use is highly context dependent. What makes things complicated in learning a foreign language, is the fact that different languages do not always make the same distinctions in expressing temporal, conditional, causal etc. relations.

Conceptually, conjunctions are one of the means to express inter-propositional relations, i.e. relations between facts. In many theories these relations are represented by a limited set of primitives, like PURPOSE, ENABLEMENT, REASON, etc. (cf Schank & Abelson, 1977; Norman & Rumelhart, 1975). Temporal relations are generally denoted by time labels. Although there is a correspondence between conjunctions and these relations, there is in general no one to one mapping. "Because" may indicate a REASON relation, whereas "although" implies pragmatic aspects, referring to expectations that should be invoked by the receiver. This makes a correspondence with one of these primitives definitely context dependent.

In Westeuropean languages there is a considerable overlap in the use of conjunctions, but there are differences as well, leading to errors. For instance, the Dutch word "als" has both a conditional ("if") meaning and a temporal ("when") meaning. In Dutch the distinction between these conceptions appears to be less emphasized than, for instance, in Italian, where one cannot use "quando" instead of "se" as a connective for an impossible condition. On the other hand, the Dutch conjunction "sinds" has a strictly temporal meaning, while the English homophone "since" may also denote reasons. This lack of distinctive use leads to errors: e.g. Dutch native speakers using "quando" where only "se" is appropriate. English native speakers using "sinds" in a causal context.

These distinctions can be easily represented in a network. A node ('topic') stands for the discriminative use of conjunctions across languages. If in a particular language the use of a conjunction is less restricted, it is related to two or more topics. (cf figure 1)



The network emphasizes differences in meaning. The meaning itself is operationalised by relating topics to contexts: sentences in which the use of the conjunctions related to that topic is more or less exclusively, or at least highly preferentially, allowed. A context consists of a subordinate clause, -without the conjunction- and a main clause, e.g. "... my car will start, we may be able to reach the city in time", which allows only conditional conjunctions (if, se, si, indien, als). In some cases, the use of other conjunctions cannot be completely excluded, but then a non-standard interpretation of the context is necessary.

This short analysis of the knowledge domain to be taught and of the misconceptions that are common in students is primarily meant as an illustration of the point that intelligent CAI does not necessarily involve full understanding by the program of all aspects of the teaching and learning process.

## 2.2. Assessment of misconceptions

The program teaching these conjunctions consists of three main phases: presentation, assessment and test. Particularly in the assessment phase intelligent didactics are indispensable. Where in the presentation the understanding is mainly up to the student, in the assessment phase the teacher/system has to figure out in what way the student may have (mis) conceived the presented subject matter.

In the program, the student does not receive direct feed-back in the assessment phase. In many cases, the student will lack the insight to understand what misconception is at the base of an error, and will therefore not be able to self-correct, when only presented with the right answer. If the student gives a wrong answer in the assessment phase, the program exhaustively generates hypotheses on the nature of the error by generating a confusion matrix. The student is not informed about this. The cells in the confusion matrix are evaluated by succeeding presentations of relevant items (an item consists of a sentence in the source language, followed by a translation in the target language, where the appropriate conjunction is missing). The evaluation of the confusion matrix generally takes two or three items. The misconceptions can be very superficial: the student may simply mix up the translation of a conjunction. But if the misconception is conceptual, i.e. the indiscriminative use of conjunctions, more steps are involved. Of course, the hypotheses testing may require more steps, when new and inconsistent errors appear, but this is quite improbable after the presentation phase.

Generally, such an algorithmic approach is not possible in diagnosing misconceptions. In this domain, the subject-matter is simple and transparent. But for more complex subject matter misconceptions cannot be represented by confusion matrices. A first complication is the fact that the size of the set of wrong responses can easily lead to combinatorial explosions. Secondly, the pattern of errors may appear to be inconsistent, but the underlying misconception can be coherent: it can be based on context sensitive rules (cf Brown, e.a., 1977). And thirdly, misconceptions cannot simply be considered as a pattern of errors, generated by a variation on a correct model, where e.g. a rule is missing. Misconceptions reflect the use of distinct and consistent other models (cf Stevens, Collins & Goldin, 1979; Stevens & Collins, 1978). These two last complications require more than straightforward computation: they require guided or 'pattern-directed' inference. Psychologically, misconceptions are often difficult to trace by teachers or researchers, because they may contain elements which are correct, but not a specific context. As Collins & Stevens (1978) showed misconceptions may have at their base many naive preconceptions. These preconceptions are conditions, not causes of misconceptions. Preconceptions are, rightly or wrongly, invoked if the processing of a text which presents the new information exceeds the capacity of working memory (Kintsch & van Dijk, 1978). Dominant preconceptions easily seduce the reader in unwarranted interpretations of new information: this is not the consequence of "prejudices", but of the fact that well established preconceptions supply evidence for coherence testing (Breuker & van Dijk, 1980).

### 3. Intelligence in CAI

Intelligence is a much desired feature in CAI mainly for understanding student responses (Camstra, 1977). In the presentation of new subject matter pre-specified texts, called 'frames' in CAI, will certainly be preferred to computation, as long as no text producing systems have become available with extensive paraphrasing capacities and a variety of didactic strategies, so that the presentation can be adapted to the individual needs, capacities and pre-existing knowledge.

The intelligence needed for response understanding is mainly dependent on the variations expected or allowed to the input. The following categories of variations play a role:

1. Typing and spelling variations or errors. Some CAI systems, including PLATO have correction facilities at their disposal.
2. Paraphrases. The student can express the same conceptualization in many different ways. There are three solutions to this problem. The most intelligent one is to provide a CAI system with natural language front-ends. But no general and manageable front ends will be available in the near future. Another solution is the use of subject-matter, which has a simple syntax and well-defined symbols for its expressions. Mathematics and programming are favourite subjects in ICAI, because the meaning of every statement in itself is easily found. This does not imply that the meaning of combinations of statements is trivial. On the contrary. Understanding the successive steps of a student's solution of a mathematical problem may require lots of inferences. Thirdly, the vocabulary and syntax of the student can be restricted. There are indications that these restrictions will not hamper the student's performances (Kelly & Chapanis, 1977). On the other hand, students seem to have problems in paraphrasing their responses, even when their first responses may differ widely (Burton & Brown, 1979).
3. The amount of initiative permitted to the student in topic selection and specification in the dialogue. One of the advantages of ICAI over traditional CAI is that the student is enabled to pursue his own interests because the system should be able to trace his explorations and guide his steps when desired. Apart from the fact tracing requires even more inference capacity than understanding paraphrases, we have found that mixed-initiative dialogues are rather an exception than a rule in tutorial dialogues. From protocols of students communicating with teachers by way of teletypes, we found that students only took initiatives when explicitly invited and were using strategies to return it as soon as possible to the teacher. An explanation for this phenomenon can be that the student is quite aware of the fact that he is less informed than the teacher. This is also indicated by the fact that students try to avoid well specified answers and that about 60 % of the interactions refer asking for specifications by the teacher.
4. The expected range and types of errors. The complexities in diagnosing misconceptions have been pointed out before. These may even cast a doubt on the feasibility of intelligent CAI, and to the educational enterprise itself, because human teachers do fail also in many cases (Brown e.a., 1977). A strategy may be to use simple and well understood domains of knowledge or to accept a more heuristic approach, which can be updated by collecting data from running programs or their simulations (Collins, Warnock & Passafiume, 1975).

If intelligence is not an all or nothing characteristic of CAI, the sharp division between ICAI and CAI appears to be accidental. ICAI has emerged from work in AI, which means that many accomplishments are in fact models. But wider applications of these models is impeded, because the instruments of AI are not easily accessible in educational settings. In order to introduce and disseminate conventional CAI it took a lot of effort to design habitable authoring languages. These languages are not suitable for symbol

manipulation, analogous to many other computer languages that do not lend themselves easily to AI problems.

We think that DART provides an instrument that can facilitate the transition between ICAI and conventional, frame oriented CAI within an established educational setting, i.e. PLATO-users.

This transition is facilitated by the following characteristics:

1. The data structures of DART can be used both for frame manipulation and for operations on structures for representing knowledge. Any node in the data network can refer to an input/output feature of PLATO (text, pictures, audio, etc). This references to output can vary from simple characters to complete frames. Therefore the more 'shallow' the controlling 'semantic' network, and the larger the textual units for presentation the more a DART data-structure resembles conventional CAI. An example of this shallowness is the use of contexts to operationalise the meaning of conjunctions.
2. The use of an ATN-dialect in DART programs has a number of advantages. ATN's are not only very suitable for parsing, eg the student's response, but they can be used for planning too, (For more arguments for the use of ATN's in ICAI cf Brown e.a., 1977).
3. The DART package also includes an editor specifically designed for not very experienced authors, and an authoring guide, which is a (conventional) CAI program teaching some basic principles of AI. Moreover the author has access to a library (file) of tried out routines.

#### References

- Breuker, J.A. & van Dijk, T.A. Tekstverwerking II: betekenisselectie en hechtheid (Textcomprehension II, selection of meaning and coherence) Subsidieaanvraag ZWO, 1980
- Brown, J.S., Burton, R.R., Hausman, C., Goldstein, I., Huggins, B. & Miller, M. Aspects of a theory for automated student modelling, BBN, Boston, BBN-Report 3549, ICAI, no 4, 1977
- Burton, R.R. & Brown, J.S. Toward a natural language capability for computer assisted instruction. In H.F. O'Neil (ed) Procedures for Instructional Systems Development, New York, Academic Press, 1979
- Camstra, B. Make CAI smarter. Computers and Education, 1, 177-183, 1977
- Collins, A., Warnock, E.H., & Passafiume, J.J. Analysis and synthesis of tutorial dialogues. In G.H. Bower (ed) The Psychology of Learning and Motivation, Vol 9, New York, Academic Press, 1975
- Kelly, M.J. & Chapanis, A. Limited vocabulary natural language dialogue Int.J. Man-Machine Studies, 9, 479-501, 1977
- Kintsch, W & van Dijk, T.A. Towards a model of text comprehension and production. Psychological Review, 85, 363 - 394, 1978
- Norman, D.A. & Rumelhart, D.E. Explorations in Cognition, Freeman, San Francisco, 1975
- Schank, R.C. & Abelson, R.P. Scripts, plans, goals and understanding. Erlbaum, Hillsdale, 1977
- Stevens, A.L. & Collins, A. Multiple conceptual models of a complex system BBN-Report 3923, Boston, 1978
- Stevens, A.L., Collins, A.M. & Goldin, S.E. Misconceptions in students understanding. Int. J. Man-Machine Studies, 11, 145-156, 1979



THE EFFECT OF MOTION CONTRAST ON SURFACE SLANT AND EDGE DETECTION

W.F. Clocksin  
Department of Artificial Intelligence  
University of Edinburgh  
Forrest Hill  
Edinburgh, Scotland EH1 2QL

Abstract

We present here three results obtained from our investigations of a computational model of the effect of motion contrast on the perception of physical (3D) surfaces. The first result is a set of equations that describes the relationship between fixed bounded surfaces and the motion contrast information given to a moving observer. It is shown how to extract surface slant and the types of bounding edges. The second result is a computer program that implements the theory. The third result is a set of graphs, derived from the equations, that depicts the psychophysical thresholds for slant and edge detection under a variety of conditions for human subjects and for computer programs. The graphs show the precise conditions under which slants and edges can be extracted from motion contrast information.

Introduction

Our investigations are concerned with the role of optical motion contrast on the visual perception of physical surfaces. Although we would expect that visual perception in general is derived from the operation of a variety of subsystems, it is useful to know the detailed properties of individual subsystems and computational models of them. Such information can tell us the conditions under which the subsystem can respond to stimuli, and whether it is feasible to construct such subsystems.

Our investigations employ certain assumptions (or 'constraints') about the nature of the surfaces to be perceived. The widely-known assumptions of surface continuity and uniqueness (Marr and Poggio, 1976) are observed. Furthermore, it is assumed that surfaces are rigid, fixed, opaque, and bounded. The bound of a surface is either some sort of corner or a depth discontinuity. The observer moves with non-zero velocity through the environment. When imaged by the observer, the projection of a surface is covered in luminance discontinuities (optical texture). It is assumed that the optical texture is fixed to the surface, so that, unlike 'glare' or shadow boundaries, it does not move with respect to the surface when the observer moves. Finally, the observer is equipped to recover the projected velocity of optical

texture elements as his movements force them to travel across the retina. At this point the assumptions end, and the theory can be derived. We will not examine the assumptions themselves here; they are accepted in all the optical flow studies cited in this paper, and are examined elsewhere (Turvey, 1977, 1979).

It has been previously shown that, providing the above assumptions are met, that the resulting optical velocity field (optical flow) can be easily described formally (Whiteside and Samuel, 1970; Lee, 1974). Furthermore, computations on the optical flow can yield information about object contours (Nakayama and Loomis, 1974) and self-movement (Pradzny, 1979). In the previous AISB/GI Conference, Clocksin (1978) suggested that gradual and abrupt motion gradients informed about surface slant and edges respectively, and showed how surface slant in principle could be extracted. In this paper we show that different kinds of surface edges can also be distinguished by optical flow information, and we give the results of a computer implementation showing the precise conditions under which surface slant and edges can be detected by a human or machine observer. The same coordinate system and derivation used in the previous paper will also be assumed here.

### Slant and Edges

From Clocksin (1978), the angular velocity  $f$  of a point at range  $r$  and eccentricity  $b$  appeared to the observer moving at speed  $S$  as  $f = S * (\sin b) / r$ , provided the above constraints were observed. Next, a sufficient representation for the local surface slant (Figure 1) with respect to the direction of locomotion was given by the pair  $(\tan s, \tan t)$ , where:

$$\tan s = \cot b - d/db \log f ; \tan t = -d/da \log f.$$

Now we show that the types of edges can also be extracted from optical flow patterns by noting the correspondence between types of edges and the optical flow that arises from the edge when a flow line crosses it. Without loss of generality, the second derivative of flow with respect to a meridian or eccentricity will inform, by the distribution of singularities, what kind of edge the meridian or eccentricity crosses (if any). Such an operator can be considered as providing a "signature" in the pattern of singularities that result. Each type of edge has its own signature, as shown in Figure 2.

Four different types of edges are distinguished. The convex and concave edges are common edges (Clowes, 1971) that typically occur at corners. There are two types of eclipsing edges -- occluding edges and disoccluding edges. An occluding edge gradually covers a background surface, and a disoccluding edge gradually uncovers a background surface as the observer moves past. The essential feature of the common edge is the discontinuous tangent in velocity with respect to a flow line crossing the edge. The sign of the feature specifies whether the edge is concave or convex. The corresponding feature of the eclipsing edge is the discontinuity in velocity with respect to a flow line, and the sign of the feature specifies whether the edge is covering or uncovering the background.

Edges and slants are computed with respect to a single  $a$ - or  $b$ -

coordinate. Edges crossing a coordinate line will be detected, but those lying the length of a line will not be. Also, no feature can be detected when both  $a$  and  $b$  are 0, because  $f$  is by definition 0 at that point. The types of edges can be distinguished providing that the assumptions mentioned in the introduction are met. If the ambient visual information falls below thresholds discussed in the last section of this paper, then the observer would require other means (Clowes, 1971) to discriminate types of edges.

### Implementation

For the purposes of explicit models of slant and edge-label detection, the following general issues are important: the discrete nature of the computations to be performed; a method for sensing the optical velocities; and establishing the  $(a,b)$  coordinate system from the data. The first issue suggests that it is necessary for the purposes of implementation to derive a discrete approximation to the theory by using differencing. All values thus obtained will be finite, so the edge signature points are now very large instead of singular. Thus, to discriminate edges from non-edges, a mathematical analogy to the psychophysical threshold must be considered. Predictions resulting from this are presented later. Second, it is necessary to represent the optical velocities. Many investigations have provided evidence that the perception of motion has a neural basis (Sekuler, 1975). The implementation discussed here makes no further contribution to our knowledge of velocity sensitive mechanisms, but only assumes the existence of mechanisms with particular properties. Third, since the  $(a,b)$  coordinate system used for representing flow fields bears no relation to retinal position, the implementation could simulate receptive fields lying along  $a$  or  $b$  flow coordinates. It is a simple matter to check if a receptive field lies along a flow line: a  $b$ -receptive field contains only flow vectors with the same direction as the field's orientation, and an  $a$ -receptive field contains only flow vectors perpendicular to the field's orientation. Any field not meeting this constraint is not oriented along an  $a$ - or  $b$ -coordinate, so it is discarded from consideration.

We implemented a computer simulation of the above slant and edge equations. The input to the program was a depth map and some parameters for observer movement. The depth map was used only as a simple way to generate the velocity flow field, and was then discarded. The program used a receptive field model to produce the local slant or edge label when the receptive field passed over the simulated terrain. The output was either in the form of a slant measure, or a label taken from the set {convex, concave, occluding, disoccluding, contour}, where a contour edge joins a surface with empty space.

The program always returned near-perfect results for two main reasons. First, the slant equations are trigonometric identities derived from the geometric basis of the problem (Clocksin, 1978). Likewise, the edge-labelling scheme has been derived from first principles, except for the template matching of the signatures. Second, the input data is simulated, and hence within roundoff error of being perfect. One would have misgivings about this: some programs, when given test data, are known to fail when given "real" data. However, psychophysical studies

of relative velocity thresholds lead us to believe that the human visual system is sufficiently sensitive to the optical flow to be able to carry out the computations we propose, and to return reproducible results. We tested this hypothesis by experimenting with the computer program. We raised the program's relative velocity threshold from  $5 \times 10^{-9}$  radians per second (roundoff error) to  $1.5 \times 10^{-4}$  rad/sec, a relative velocity threshold in humans under good conditions (Graham, Baker, Hecht, and Lloyd, 1948). As a result, we found that computations of surface slant by the program varied plus or minus 4 degrees from the correct value. This compares well with judgements of surface slant by humans, whose estimates vary between plus or minus 4 and 7 degrees (Gibson, Gibson, Smith and Flock, 1959; Flock, 1964) under good conditions.

Other artefacts result from the discrete nature of the computations performed. The program cannot determine the slant of surfaces that subtend less than 5 quantization units of arc, and all non-contour edges are located to within 2 units of arc from the correct position. Contour edges are located to within 3 units of arc of the correct position. Edge types are correctly labelled, except for edges within a unit of arc from the optical centre of expansion ( $a=b=0$ ), where edges cannot be detected. Slant values are smoothed over the centre of expansion.

Further implementation details are not appropriate here. A complete derivation of the theory, and a physiological model together with other items relevant to vision research are published elsewhere (Clocksin 1980a). Results will separately appear on a new statistical algorithm for extracting velocity vectors from images without the use of correlation or correspondence.

#### Specifying the conditions for perception

What are the conditions and limits under which the observer obtains effective information from optical flow? The fact that the Ames Window (Ames, 1951) is ambiguous even though it is a real moving surface means that there are thresholds which must be exceeded before the information is effective. The notion of a 'threshold', or the minimum amount of a stimulus that is necessary for its detection at least fifty percent of the time, is used throughout psychophysical research. Although thresholds have been determined for simple stimuli such as luminance and motions, such work has not previously been carried out for more "physical" stimuli such as surface slant and edges. We can use our slant and edge theory to derive the slant and edge detection thresholds for humans under a variety of conditions. When cast into threshold boundary graphs given below, we are able to predict the conditions under which humans can or cannot detect surface slant and edges, based on whether the motion contrast stimulus is above or below empirical thresholds. For simplicity, the assumption is made that motion contrast is the only source of information used, but in general we should not rule out cooperative effects from other sources of information. Thresholds can also be estimated for computer programs, so the tradeoff between resolution and performance can be determined in advance.

The conditions we can vary for the slant and edge judging tasks include

the distance to the surface and the speed of the observer. For the slant task we also vary the amount of slant, and for the edge task we vary the depth of the depth discontinuity. The observer's movement could be, for example, a side-to-side head movement. Given some set of conditions, all we need is to compute the motion contrast stimulation resulting from the conditions (see Figure 3), and determine whether the motion contrast is above or below some threshold. Stimulation above the threshold means that the slant or edge should be detectable. Although the exact value of the threshold is not crucial for our purposes, a typical relative velocity threshold of 0.5 minutes of arc per second was empirically determined by (Graham, et al, 1948), and depending on conditions, measured thresholds can range from 0.4 to 1.5 minutes of arc per second.

By comparing relative velocity thresholds with the motion contrast calculated to occur under a large combination of conditions, we computed the theoretical boundary between subthreshold and suprathreshold stimulus conditions for slant and edge detection. Graphs showing the resulting boundary curves for various conditions are shown in Figure 4. In both slant and edge graphs, the range to the surface (in meters) is represented by the abscissa, and curves are shown for several observer speeds (in meters/second). The ordinate represents amount of slant (in degrees) for the slant graph, and change in depth (in meters) for the depth graph. For a given speed, the appropriate curve describes the boundary between suprathreshold and subthreshold stimulus conditions. Points on the graph lying below the curve represent conditions for which motion contrast stimulation is below the threshold. Points above the curve represent suprathreshold conditions, which should be detected by human observers. Figures 4a and 4c refer to edge and slant thresholds for humans. For example, Figure 4a tells us that we should be able to resolve (50% of the time under good conditions) a 10 cm deep edge at a distance of 8 meters, provided that we (or the edge under certain conditions) travel at 10 cm/sec past the edge. Also, we should not expect to judge the 10 degree slant of a surface (by flow alone) unless we are moving at least 10 cm/sec and the surface is less than 2 meters distant. Motion contrast should become more effective as conditions well above threshold were observed.

In order to test predictions resulting from this study, we reviewed a number of experiments pertaining to slant and edge perception (including Gibson, et al, 1959; Smith and Smith, 1963; Flock, 1964; Rogers and Graham, 1979), and found that our predicted boundary curves accurately discriminate amongst experiments where subjects (on the average) did or did not obtain slant and edge percepts from motion contrast (Clocksin, 1980b). This would suggest that the effectiveness of motion contrast in slant and edge perception in the absence of other information depends on having conditions that provide sufficient amounts of motion contrast. We hope to have shown just what those conditions are.

This analysis can also evaluate computer vision techniques. Limb and Murphey (1975) determined, for their motion detection algorithm, the range over which the speed of a single object moving along the scan lines in the video image could be measured relative to a contrasting background. The lower bound was 0.25 pixels per frame. We converted this to a relative velocity threshold for real-time motion detection using a proposed frame rate of 60 frames/sec for a frame subtending 10

degrees of arc and a 256x256 quantisation. The resulting threshold, 0.010227 radians per second, is about two orders of magnitude higher (worse) than human vision. The edge and slant threshold boundary curves under these conditions are shown in Figures 4b and 4d. Under these conditions, the observer's speed must increase significantly to detect the same features: to measure surface slants, for instance, a robot equipped with such a vision system would need to move at rather distressing speeds exceeding 6 meters per second. However, by improving the velocity detection algorithm or by increasing resolution (the angle subtended by a pixel), the relative velocity threshold (and consequently, slant and edge thresholds) would decrease. Thus, the tradeoff between resolution and performance can be examined. Further ways to increase performance could make use of other information such as texture gradients, however, Braunstein (1976) has found that motion overrides static indicators of surface slant, completely dominating slant judgements by humans.

#### Acknowledgements

I thank Jim Howe for his support in this investigation.

#### References

- Ames, A J (1951), Visual perception and the rotating trapezoidal window, Psychological Monographs 65(7), 1-32
- Braunstein, M L (1976), Depth perception through motion, Academic Press
- Clocks in, W F (1978), Determining the orientation of surfaces from optical flow, Proc. Conf. AISB/GI, Hamburg, 93-102
- Clocks in, W F (1980a), Perception of surface slant and edge labels from optical flow: a computational approach, Perception 9, no. 3.
- Clocks in, W F (1980b), Computer prediction of visual thresholds for surface slant and edge detection from optical flow, Ph.D. Thesis in preparation.
- Clowes, M B (1971), On seeing things, Artificial Intelligence 2, 79-114
- Flock, H R (1964), Some conditions sufficient for accurate monocular perceptions of moving surface slants, J. Exp. Psych. 67, 560-572
- Gibson, E J, Gibson, J J, Smith, O W, and Flock, H (1959), Motion parallax as a determinant of perceived depth, J. Exp. Psych. 58, 40-51
- Graham, C H, Baker, K E, Hecht, H, and Lloyd, V V (1948), Factors influencing thresholds for monocular movement parallax, J. Exp. Psych. 38, 205-223
- Lee, D N (1974), Visual information during locomotion, in Perception: Essays in Honour of J.J. Gibson, R B MacLeod and H L Pick (eds), Cornell Univ. Press

- Limb, J O, and Murphey, J A (1975), Estimating the velocity of moving images in television signals, Computer Graphics and Image Processing 4, 311-327
- Marr, D and Poggio, T (1976), Cooperative computation of stereo disparity, Science 194, 237-238
- Nakayama, K, and Loomis, J M (1974), Optical velocity patterns, velocity sensitive neurons, and space perception: a hypothesis, Perception 3, 63-80
- Prazdny, K (1979), Egomotion and relative depth map from optical flow, Computer Science Department, University of Essex
- Sekuler, R (1975), Visual motion perception, in Handbook of Perception, Volume 5, E C Carterette and M P Friedman (eds), 387-430
- Rogers B and Graham M (1979), Motion parallax as an independent cue for depth perception, Perception 8, 125-134
- Smith, O W, and Smith, P C (1963), On motion parallax and perceived depth, J. Exp. Psych. 65, 107-108
- Turvey, M T (1977), Contrasting orientations to the theory of visual information processing, Psych. Rev. 84, 67-78
- Turvey, M T (1979), Commentary on motor-sensory feedback and geometry of visual space, The Behav. and Brain Sci. 2, 81-83
- Whiteside, T C D, and Samuel, G D (1970), Blur zone, Nature 225, 94-95

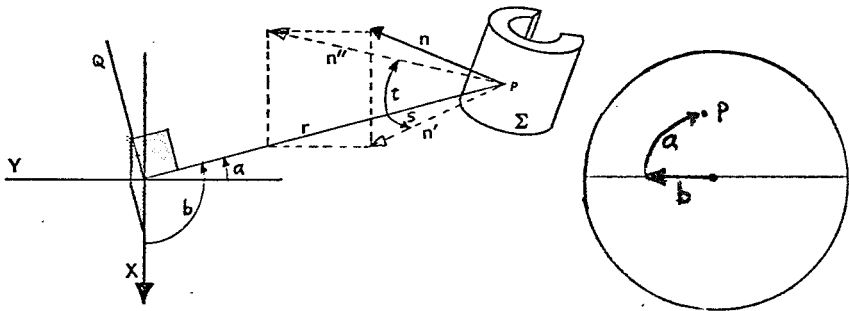


FIGURE 1: (a) The coordinate system centred on the observer  $O$  moves with speed  $S$  in the direction of the  $X$ -axis;  $a$  and  $b$  are angular spherical coordinates of meridian and eccentricity;  $n$  is the normal of surface  $\Sigma$  at point  $P$ ; the slant is  $(\tan s, \tan t)$ . Angle  $s$  is the angle between  $r$  and the projection  $n'$  of  $n$  into the  $RX$  plane. Angle  $t$  is the angle between  $r$  and the projection  $n''$  of  $n$  into the  $QR$  plane. The velocity flow from  $P$  is  $f = (S \sin b)/r$ . (b) The image of  $P$ , viewed from  $O$ , sighting along the  $X$ -axis.

FIGURE 2:

Edge Profile (range)	Edge Profile (velocity)	Edge Signature (singular points)

FIGURE 3:

Using  $f$  with  $b=90$  (movement perpendicular to line of regard), motion contrast is  $|f - f| = S(1/r - 1/r)$ .

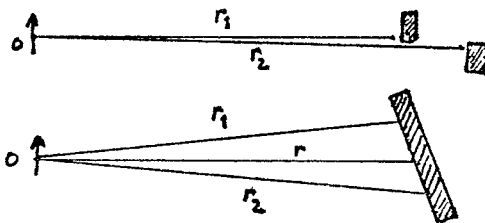




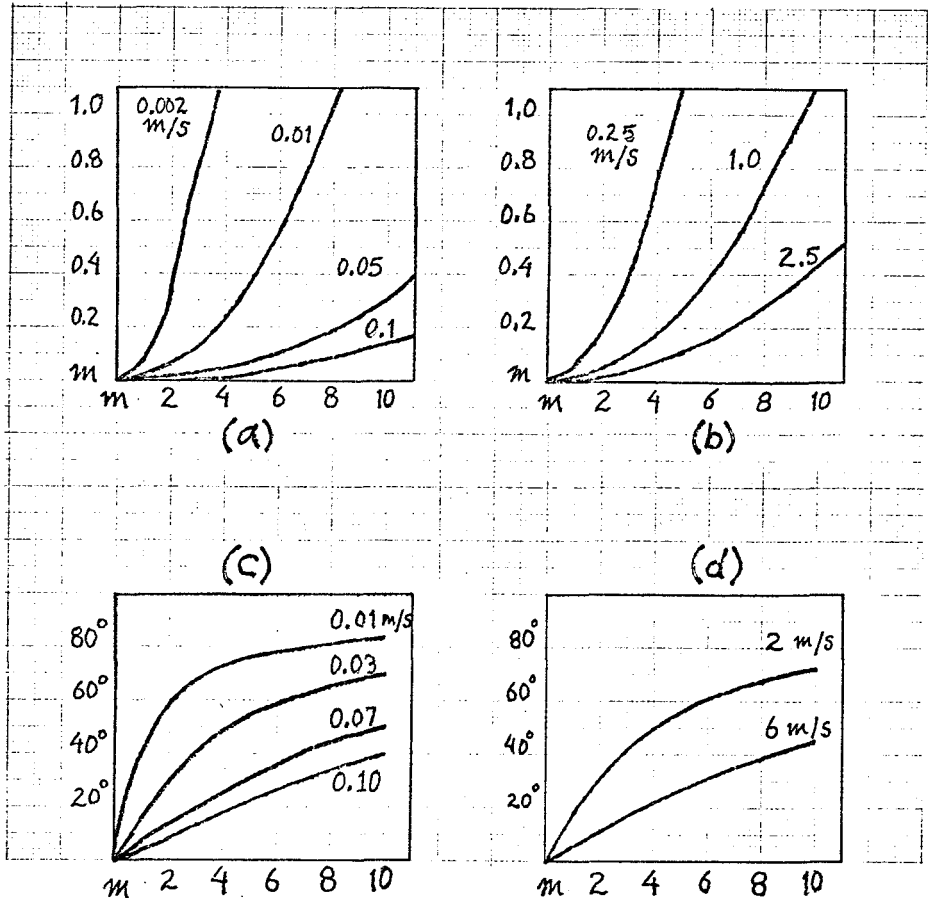
FIGURE 4:

(a) Predicted edge threshold boundary curves for a human observer moving at 2 mm/sec, 1 cm/sec, 5 cm/sec, and 10 cm/sec.

(b) Edge threshold curves for simulated computer vision system moving at 25 cm/sec, 1 m/sec, and 2.5 m/sec.

(c) Predicted surface slant threshold curves for a human observer moving at 1 cm/sec, 3 cm/sec, 7 cm/sec, and 10 cm/sec.

(d) Slant threshold curves for simulated computer vision system moving at the rather formidable speeds of 2 m/sec and 6 m/sec.



DRAPER-1  
Using models to augment rule-based programs

Stephen W. Draper  
Cognitive Studies Programme  
University of Sussex  
Brighton, Sussex, U.K.

Abstract

This paper discusses the design of a program that tackles the ambiguity resulting from the interpretation of line-drawings by means of geometric constraints alone. It does this by supplementing its basic geometric reasoning by means of a set of models of various sizes. Earlier programs are analysed in terms of models, and three different functions for models are distinguished. Finally, principles for selecting models for the present purpose are related to the concept of a "mapping event" between the picture and scene domains.

Introduction

The line-labelling scheme proposed by Huffman (1971) and Clowes (1971) in effect posed the problem of producing all the interpretations in terms of line-labels of a line-drawing that can be made on the assumption that it is a projection of a scene composed of polyhedra with trihedral vertices, plane surfaces, and straight edges, and also that the viewpoint does not give rise to any accidental alignments of vertices or edges (i.e. any "accidentals"). The line-labelling scheme offered only a partial solution, and left two major challenges for future program designs: to relax the non-accidental and trihedral restrictions (i.e. to allow accidental alignments and vertices with more or fewer than three surfaces), and to generate only those labellings that are geometrically consistent with the assumptions - Huffman himself showed that the line-labelling scheme was inadequate in this respect. Waltz' (1972) program did nothing towards the latter goal, and made only ad hoc attempts at the former by including a few hand-picked accidental and multihedral junction labellings (his real contribution was to the expression of knowledge about shadows). Mackworth's program Poly (Mackworth 1973) achieved the first and partly achieved the second goal, and work by the author on sidedness reasoning including a design for a program called Ellsid (Draper 1978, forthcoming) has completed this goal. It turns out, however, that the purely geometric constraints thus fully captured allow an enormous number of interpretations (many hundreds even for simple drawings of a cube or a tetrahedron) even though people see only one or two. Clearly the next task is to attempt to model the choice of interpretation made by us from among those geometrically possible.

Much of this ambiguity comes from allowing accidentals indiscriminately - in effect this treats each picture region as the projection of (part of) an opaque plate with no necessary point of contact with any other, and many of the interpretations represent such weird, disconnected scenes where the edges of floating plates are lined up in various unlikely ways. This does not however account for all the ambiguity since even excluding accidentals still allows numerous odd interpretations. It is now clear that the trihedral and accidental restrictions in the Huffman-Clowes scheme were responsible for keeping the interpretations produced in fairly close correspondence to human interpretations, even though their rigid application prevented the interpretation of some simple pictures. Kanade (1978) shows how even a slight relaxation of the restrictions (redefining the trihedral restriction to allow up to three surfaces at a vertex, which can be either laminae or faces bounding a solid volume) greatly multiplies the possible interpretations of simple pictures and that extra scene constraints must then be mobilised. However his proposed scheme, although an

interesting compromise between ambiguity and geometric competence, neither reduces the ambiguity to the point of corresponding to human perception, nor fully enforces the applicable geometric constraints. An obvious suggestion then is to look for a way to select the interpretation which conforms or most nearly conforms to the Huffman-Clowes restrictions while retaining the more general geometric powers of sidedness reasoning for use as and when necessary.

This paper outlines an approach which uses models of familiar objects and fragments of objects to implement, and hopefully to improve on, this suggestion. The program will be based on the sidedness reasoner from Ellsid, which is essentially rule-based, but will be augmented by the models whose role is to guide the interpretation - in effect resolving the ambiguity found by Ellsid alone. It is argued that this offers a method of capturing the good aspects of past programs (including the Huffman-Clowes scheme) particularly their abilities to choose the same interpretations as people, while overcoming their inadequate grasp of geometric constraints and inability to cope with accidentals when this is necessary. In addition it offers a way of modelling the effect of common or familiar objects or configurations on the perception of drawings.

### Model-based vision programs

Most, perhaps all, programs can be seen as model-based in some sense despite the fact that their general "feel" may be that of a bottom-up general purpose method. For instance Woodham's (1977) program for getting shape from shading by a local computation uses models of fragments of surface shape. In the domain of line-drawing interpretation, the succession of programs can be seen as having been based on successively smaller models. Roberts (1965) used complete, simple, convex polyhedra such as bricks and wedges as models. Line-labelling schemes are based on models of possible vertices and their appearances. Mackworth's Poly and the author's Ellsid both use planes as their basic element or model: they reason about how these may be fitted together to make up scenes.

Here it is useful to distinguish three different aspects of the use of models.

1. They can largely determine the way in which the scene (i.e. the interpretation) is described. This is most obviously true when models are used to achieve recognition of known objects - the interpretation may then consist almost entirely of the names of those objects. Likewise when models carry information which could not otherwise be deduced from the picture - such as lengths in Falk's (1972) program and hidden surfaces in Roberts' program - they have a large effect on the content of the scene description. Apart from this effect on its content, models may affect its structure by determining the elements of which it is made up: Roberts' program sees an L-beam as two bricks welded together whereas Poly sees it as planes meeting along edges with no sub-division into convex blocks.
2. Models can be used as hypotheses - sets of conclusions about the scene that are jumped to on the basis of slight evidence, though some checking may follow. Roberts' program has this flavour; line-labelling does not since it considers all possibilities and allows all consistent interpretations that survive all the checks it knows how to make. As we shall see, it is this aspect of models that is needed in the present application since they are to be the basis for going beyond the geometric constraints.
3. Models are often the basis for organizing the way knowledge is built into the program - primarily constructs to help the programmer organize the code, nuclei for structuring the program. It is in this sense that all the programs mentioned are model-based - they are all organized around some basic scene concepts. Loosely speaking, frames (Minsky 1975) are models in this sense since a frame brings together procedures as well as declarative information, and the idea is to organize the program round frames whether or not

it has a hypothesise-and-test flavour and however its scene descriptions are organized. In vision this is taken furthest by Freuder (1976) who organizes all knowledge around models but these appear in what are effectively four different networks of declarative information and have at least two sets of procedures associated with them (for use when activated in top-down and bottom-up modes respectively).

#### A program design using models to select interpretations

We are now on a better position to specify clearly what role the models are to play in the proposed program design. We do not want them for defining the scene description language - that is still to be done primarily by the underlying sidedness reasoning program using line-labels and its representation of planes and their relationships; neither are the models to be central units of the program organization. They are instead required to provide likely hypotheses about parts of the scene.

The basic sidedness reasoner is retained firstly to ensure that the final interpretation is geometrically consistent - and so it is used to check that the hypothesised fragments of scene description are mutually compatible. Each hypothesis must therefore be expressed in terms that the reasoner can deal with directly - as sidedness assertions about planes. Note that in terms of the underlying plane-based approach the models do not correspond to natural fragments of scene objects: for instance the all-convex labelling of a Y-junction, which in the Huffman-Clowes scheme is a complete model of a vertex, tells a plane-based system something about one corner of each of three surfaces and the way they meet each other there. In this respect these models are more like the M.I.F.s and M.U.F.s developed by Frank Birch (1978) in a letter recognition system than they are like Roberts' models. M.U.F.s (minimal unambiguous fragments) are combinations of strokes that have no meaning by themselves as letters but are valuable to a program as a combination that can belong to only one letter and can hence initiate some special processing. M.I.F.s (minimal impossible fragments) are stroke combinations that cannot be part of any single letter and hence signal that some stroke junctions must be undone. Both are models selected not for their significance in the resulting interpretation but for their usefulness to the process constructing the interpretation.

The second function of the sidedness reasoner is to fill in the gaps in the interpretation when there are parts of the picture not covered by hypotheses of familiar configurations - e.g. at accidentals. How many such "gaps" occur - that is, how much of the interpretation will not be covered by the models - depends partly on how extensive the set of models is, and partly on whether the control strategy is to search for the interpretation that has the maximum proportion supplied by models, or to grow an interpretation outwards from an initial model match, using other models if possible but only backtracking if forced to by a geometric inconsistency. The latter strategy will not always give the "best" interpretation and in general will depend on the order in which parts of the picture are tackled. Probably both should be explored to see which can be made to fit human performance best.

A second major design decision is to trigger the models by matching cues in the picture domain. It is possible to have a system where scene descriptions are generated by some means and then models of 3-D configurations are matched to them. This is sufficient for recognition systems, where the purpose of the models is to identify known objects, and it could be used here by preferring interpretations containing familiar 3-D configurations. However it seems unlikely to provide a good model of human preferences for several reasons. Firstly, such a system would behave like a paranoid: it would have strong ideas about what the interpretation "should" be and very slight evidence would be enough to "confirm" this - it would take no account of what those appearances

probably or usually indicate. What we really want is a system that assigns appearances a plausible interpretation and does not invoke unlikely links between appearances and interpretations unless it has to. This is achieved by having a stored set of picture configurations called "keys" or "cues" each of which triggers a particular fragment of 3-D interpretation called a model. It is these pairs of keys and models that are stored, and have been loosely referred to up to now simply as "models".

This decision fits well with previous programs, and with the aim of trying to recapture the insights contained in past work. Line-labelling is based on an insight first exploited by Guzman (1969) that picture junctions are good evidence about the scene (see Hochberg 1968 for evidence of the psychological reality of this): what we want to capture here is the idea that Y-junctions, say, normally have one of the three interpretations allowed by the Huffman-Clowes scheme although a lot more are possible. Similarly in Roberts' program models have keys associated with them that are good candidates for incorporation in the present program, as are the "line features" used by Grape (1973).

#### Principles for choosing model-key pairs for incorporation

Depending on the motives for constructing a version of the proposed program, various principles might be used in selecting the model-key pairs to be used. Interesting experiments could be made to see how well the ideas in previous programs will work when augmented by sidedness reasoning - for instance by using as models just the Huffman-Clowes set of junction labellings one could find out if it could now cope with multihedral vertices and accidentals while avoiding geometrically impossible interpretations and still generally producing only those interpretations which people see.

Another idea would be a learning program which had some method of analysing each picture-plus-dictated-interpretation in a training sequence and compiling a set of picture fragments plus interpretations to use as models, selected perhaps on the basis of frequency of occurrence. Alternatively a set could be compiled by hand, the aim being to model the interpretations people select over as large a set of pictures as possible.

The above program designs might possibly achieve a good approximation to human perceptual behaviour (within the limits of the power of line-labels to express 3-D interpretations) but they could never offer a theoretical explanation of why they worked. What than should the principles for selecting models be? A possible answer comes by extending Huffman's General Viewpoint idea (Huffman 1971 p.298): we want picture-to-scene hypotheses that are probable. To explore this we need to develop the picture/scene distinction emphasized by Clowes (1971).

The keys are defined in the picture domain - e.g. a Y-junction is defined by picture angles etc. independent of the labelling (interpretation) that may later be assigned to it. The models are defined in the scene domain - that is they specify aspects of the 3-D scene in a way that in principle is independent of the picture (this is completely true of Roberts' program, only partly true when the interpretation is specified by line-labels). In order to identify probable key-model pairs we must consider the relationship between parts of the scene and the appearances they generate - I shall call these pairings "mapping events". A good example of a mapping event is a T-junction generated by an edge being occluded. Not all T-junctions signal occlusion and not all occluded edges generate T-junctions. When trying to interpret a T-junction the question is: was occlusion the mapping event that generated it? If that is the hypothesis you adopt, it dictates certain features of the putative scene - a relative depth relationship for instance; in general several somewhat unconnected scene relationships are specified by the hypothesis of a given mapping event. Occlusion

is not a scene property - it is not part of a scene but a consequence of the association of a scene and a viewpoint, pairing a particular appearance (i.e. a picture) with the scene.

In order to pursue the idea of choosing key-model pairs that represent probable hypotheses, then, models should not be chosen by selecting convenient scene fragments nor keys by selecting convenient picture fragments. Rather we want to identify keys that have a good chance of leading to a correct piece of interpretation. It is mapping events like occlusion and accidentals that are more or less probable, and so it is these probabilities that should be taken into account in formulating hypotheses and in designing the model set that determines the hypothesis-making of a program. Key-and-model pairs correspond to mapping events and should therefore be selected for maximum probability of the correspondence. Some mapping events have a relatively high probability and are worth being stored explicitly as models, while others do not have a high enough relative frequency to warrant this. Ideally we would like to identify keys for which a subset of their possible interpretations account for the large majority of their preferred interpretations in practice. The non-accidental interpretations of a junction are an example of this. The selection should also be influenced by the effect of other constraints operating on a hypothesised interpretation - one can afford to consider fairly unlikely interpretations if they are (nearly) always ruled out by other constraints when they are incorrect (not preferred). A considerable further amount of theoretical work needs to be done on the behaviour of such systems, but practical experience with various sets of models in the proposed system may suggest important leads in this.

### Conclusion

The proposed program will have a competent geometric facility as its basis (the sidedness reasoner) to check the overall interpretation produced and to act as a medium for combining the contributions of other parts. This will be supplemented by a system of stored models of a range of sizes. These models are not included to provide the program's basic ability at interpretation but to generate good hypotheses for the reasoner to work on. They can be seen as partial results stored ready-made because they are frequently encountered, and as such they may save computation. However that is not their primary function. Like M.I.F.s and M.U.F.s, they are items useful to the program for controlling the computation rather than for making up large portions of the output. They are hypotheses in the sense intended by R.L. Gregory (e.g. 1974) when he characterised vision as a process of forming and checking hypotheses: they are used to resolve the ambiguity inherent in the picture. They cause the program to jump to a conclusion that goes beyond the evidence in the sense that, although it will check that no constraint refutes the hypothesis, the interpretation is partly determined by ignoring the possible alternatives.

Acknowledgements. The ideas put forward in this paper mostly originated in some form with my colleagues. Max Clowes showed me that most vision programs can be seen as based on models of some sort. Aaron Sloman has persistently argued for the simultaneous use of models of different sizes: an idea analogous to that put forward for natural language by Becker in "The phrasal lexicon". But most immediately important was talking to Dave Owen while he struggled to find principles for applying this idea in the Popeye program: it was he who made me realise that models may not form natural subdivisions in the scene domain.

The research of which this forms a part was supported by the Science Research Council, grant number GR/A79703.

References

- Becker J.D. 1975 "The phrasal lexicon" in TINLAP conf. proc. ed. Schank R. and Nash-Webber B.L. pp.70-73 (Cambridge, Mass.: Bolt, Berenek, and Newman)
- Birch F. 1978 "A (self-adapting) network for recognition of visual structures" Proc. A.I.S.B./G.I. conference, Hamburg.
- Clowes M.B. 1971 "On seeing things" Artificial Intelligence vol.2 no.19 pp.79-116
- Draper S.W. 1978 "Competence at interpreting line-drawings: a preliminary report on Ellsid" Report, Cognitive Studies Programme, Sussex University.
- Falk G. 1972 "Interpretation of imperfect line data as a three dimensional scene." Artificial Intelligence vol.3 no.2 pp.101-144
- Freuder E.C. 1976 "A computer system for visual recognition using active knowledge" AI-TR-345 (Cambridge, Mass.: M.I.T.)
- Grape G.R. 1973 "Model based (intermediate-level) computer vision" Stanford AI memo AIM-201 computer science department, Stanford University
- Gregory R.L. 1974 "Perceptions as hypotheses" in Brown S.C. (ed.) Philosophy of psychology (London: MacMillan) pp.195-210
- Guzman A. 1969 "Decomposition of a scene into three-dimensional bodies" in Grasselli A. (ed.) 1969 Automatic interpretation and classification of images pp.243-276
- Hochberg J.E. 1968 "In the mind's eye" in Haber R.N. (ed.) Contemporary theory and research in visual perception pp.309-331 (London: Holt, Rinehart, and Winston)
- Huffman D.A. 1971 "Impossible Objects as Nonsense Sentences" in Machine Intelligence 6 pp.295-323 ed. Meltzer B. & Michie D. (Edinburgh: Edinburgh University Press).
- Kanade T. 1978 "A theory of origami world" Technical report, dept. of computer science, Carnegie-Mellon university. CMU-CS-78-144
- Mackworth A.K. 1973 "Interpreting pictures of polyhedral scenes" Artificial Intelligence vol.4 no.2 pp.121-137
- Minsky M. 1975 "A framework for representing knowledge" in Winston P.H. (ed.) 1975 The psychology of computer vision (New York: McGraw-Hill) pp.211-277
- Roberts L.G. 1965 "Machine perception of 3-D solids" in Tippet et al. (eds.) Optical and electro-optical information processing pp.159-197 (Cambridge, Mass.: M.I.T. Press)
- Waltz D.L. 1972 "Generating semantic descriptions from drawings of scenes with shadows" MAC AI-TR-271 (Cambridge, Mass.: M.I.T.)
- Woodham R.J. 1977 "A cooperative algorithm for determining surface orientation from a single view" Proc. I.J.C.A.I.5 vol.2 pp.635-641

## AUTOMATIC IMPLEMENTATION OF ALGEBRAIC SPECIFICATIONS OF

## ABSTRACT DATA TYPES

H. Eigemeier, Ch. Knabe, P. Raulefs, K. Tramer

Institut für Informatik III

Universität Bonn

Postfach 2220

D-5300 Bonn 1, W. Germany

**Abstract.** The system ADTCOMP which constructs LISP-implementations from algebraic specifications of abstract data types is presented. ADTCOMP accepts specifications with conditional axioms and hidden operations (parameterized specifications are not yet accepted). Code is generated by applying programming knowledge codified in terms of production rules.

1. Introduction

A standard paradigm of software development is to start with abstract specifications, and to gradually expand, refine, and transform them into an appropriate linguistic level to obtain efficient implementations. Algorithm design should be done independently of machine representations.

Algebraic specifications [ADJ 76] of abstract data types (ADTs) allow to specify data types by characterizing the behavior of data objects under characteristic operations without using any particular representations of data objects. Hence, developing algorithms at the level of manipulating objects of algebraically specified ADTs provides an optimum of freedom from specific representations, facilitates the manipulation of algorithms (e.g. program transformations), and allows to prove properties of algorithms which are independent of representation details. The remaining task consists in implementing such *abstract algorithms* in terms of appropriate representations supported at the machine level.

Previous investigations [BAR 77, LON 77] in constructing concrete implementations of abstract algorithms have revealed that the problem primarily is how to represent and utilize programming knowledge. However, the "abstract" structures considered by PECOS [BAR 77] (and even more in [LON 77]) are *substantially* more "concrete" than algebraic ADT-specifications. ADT-specifications only provide functionalities of operations and equational axioms on which design decisions can be based on.

We present the system ADTCOMP which constructs LISP-implementations from algebraic ADT-specifications. Programming knowledge is codified in terms of production rules, contained in five different production bases. ADTCOMP constructs LISP-code by successively working through seven phases, each of which incorporates specialized programming knowledge and refines intermediate constructions obtained from the previous phase. ADTCOMP accepts specifications with conditional axioms and hidden operations (see e.g. [ADT 79]), but not yet parameterized specifications. So far, ADTCOMP has successfully implemented *all* ADT-specifications we could get hold of (about 70). The five production bases contain some 120 rules. There is an additional system of meta-rules for meta-reasoning and constraining search. ADTCOMP uses well-established AI-techniques such as agendas and dependency-directed backtracking.

It turned out that the choice of LISP as target language was not essential. Although this will not become clear from this paper, we could have easily changed our rules to generate implementations in terms of SIMULA-classes (say; in fact, such a redesign making ADTCOMP multilingual is currently under way).

This paper is aimed at introducing the basic machinery by which ADTCOMP works, at a level of detail that allows to reconstruct how simple and familiar data types are implemented. Section 2 gives a rough survey on ADTCOMP illustrated by a detailed walk through the implementation of a simple data type. Section 3 reviews decision making and coding illustrated by a variety of simple, but non-trivial examples.

ADTCOMP is implemented in INTERLISP without using features not being standard in LISP-systems (ADTCOMP even runs on LISP-F3 [NOR 79]). The system as well as a detailed reference manual is available from the third author.



## 2. A Survey of the ADT-Compiler

This section introduces the most important mechanisms of ADTCOMP. To show how the system works, this overview is illustrated by a particularly simple and familiar example, the unbounded stack abstraction.

### 2.1. Input Specifications and System Structure

**2.1.1. Input Specifications.** The input to ADTCOMP is an *algebraic data type specification* (S,E) which consists of - a *signature* (S,E) containing sorts and operations with their functionalities.  
- a set E of *equational axioms* describing the effect of operations.

Completeness and consistency checks are not yet included in ADTCOMP.

**2.1.2. Example.** We consider the specification for unbounded stacks:

```

DATATYPE      : STACK D;           D, X are sorts for STACK- and component-objects
COMPONENTSORT: X;
OPERATIONS    : PUSH: D X → D;
                POP : D → D;      these are the operations and their functionalities.
                TOP  : D → X;
AXIOMS: [A1] (POP D0) = UNDEF;
        [A2] (TOP D0) = UNDEF;
        [A3] (POP (PUSH D2 X2)) = D2;
        [A4] (TOP (PUSH D2 X2)) = X2;

```

So far, ADTCOMP is restricted to specifications of data types D s.t. all D-objects can be inductively generated from a basic D-object D0, which is the empty stack in our example. For all sorts, UNDEF is the error element of the respective sort.

**2.1.3. System Structure.** Upon encountering an input instruction, ADTCOMP works through seven phases that successively construct a LISP-implementation of the data type. Except for the first and last one, each phase incorporates specialized programming knowledge and refines the intermediate constructions obtained from the previous phase. These seven phases are:

- (1) *Initialization (I-) Phase.* The I-phase converts the *input specification* into a LISP-representation.
- (2) *Functionality (→) Phase.* The →-phase considers the *signature* of the input specification only. Conclusions from the operations' functionalities are drawn, and the structure of LISP-functions implementing these operations is established.
- (3) *Axiom (=) Phase.* The =-phase considers the *axioms* of the input specification only, and makes inferences from observations on the axioms.  
After the =-phase, the input specification is no longer considered.
- (4) *Representation (R-) Phase.* The R-phase combines conclusions of the previous phases to determine the data structure in which D-objects are represented.
- (5) *Compile (C-) Phase.* Using all information obtained so far, the C-phase constructs actual LISP-code to make optimizations and improvements.
- (6) *Cleanup Phase.* The Cleanup-phase refines and/or rearranges the LISP-code obtained from the C-phase to make optimizations and improvements.
- (7) *LISP-Phase.* The LISP-phase constructs the final LISP-code. As LISP does not support a module-construct such as SIMULA-class or ALPHARD-form, a corresponding mechanism is built up in the LISP-phase.

Phases (2)-(6), which actually apply programming knowledge, are production systems. Applications of production rules is controlled by meta-rules and attention focussing. These mechanisms, dependency-directed backtracking, and pattern matching are not discussed in this paper.

### 2.2. A Walk Through the Automatic Implementation of an Abstract Data Type Specification

To explain how ADTCOMP works, we present a walk through the implementation of the STACK-specification 2.1.2.

**2.2.1. Initialization (I-) Phase.** The input specification is converted into an internal LISP-representation, and the LISP-atoms D, X, PUSH, POP, TOP are associated with the following properties:

D	NAME	STACK	<i>name-property of D has value STACK.</i>
X	COMP SORT		<i>the COMP SORT-property of X has value TRUE (omitted).</i>

# EIGEMEIER-3

```

PUSH OP (IMPLEM)
POP OP (IMPLEM)
TOP OP (IMPLEM)

```

these operations are to be IMPLEMENTed,  
i.e. they are no hidden operations.

2.2.2. *Functionality (→) Phase.* →rules check functionalities of operations to make useful observations. In our case, the three rules →01, →02A, →08 apply:

```

→01. *OP1: D <1 *Y> ... → D. <1 *Y NOTEQUAL D>
    =>*OP1 CONTINUABLE SINGLE

    if *OP1 maps a D-object and possibly a *Y-object to a D-object and *Y is not D
    then *OP1 is a singly continuable operation

```

- Note. (1) An operation op is *n-fold continuable* iff op constructs from n D-objects another D-object.
- (2) The first clause of the above rule is matched against the OPERATIONS-part of the specification. \* prefixes match-variables. In '<1 \*Y>', 1 is a LISP-atom bound to TRUE iff \*Y is matched to something. 1 serves as a condition for the test '<1 \*Y NOTEQUAL D>' which is evaluated only if 1 is bound to TRUE. We call such a test a *conditioned test*.

```

→01:Result. PUSH CONTINUABLE SINGLE
            POP CONTINUABLE SINGLE

```

```

→02A. *OP1: D ... → X => *OP1 READS

    if *OP1 maps D-objects and possibly more arguments to component sort X
    then *OP1 reads some contents from D-objects.

```

→02A only fires for operation TOP which results in

```

→02A:Result. TOP READS

```

The third applicable rule sets up the structure of the LISP-functions that will implement the operations:

```

→08. *OP1 OP IMPLEM,
    NOT *OP1 CONTINUABLE DOUBLE,
    LET *LIST BE LIST,
    <1 *NA MEMORY 1>,
    <2 *NB MEMORY 2>,
    LET 3 BE NOT 1, 4 BE NOT 2,
    <7 *OP1 CONTINUABLE>,
    <8 *OP1 SETS>,
    *OP1:<D><5 *NA> <6 *NB> *LIST →
    => *OP1 FUNCT
    (D<1 *NA><2 *NB><3<5 *NA>><4<6 *NB>> *LIST.
    FORM <1<7 S>> <8 S> <3<7 D>>,
    *OP1 PARAMS <1 *NA><2 *NB>
    <3<5 *NA>><4<6 *NB>> *LIST)
    then
    this is the structure of the LISP-function
    implementing *OP1, entered under property
    FUNCT of *OP1,
    the property PARAMS of *OP1 records all
    parameters except for the first (for D-objects); the PARAMS-property is used in recursive calls.

```

```

→08:Result. PUSH FUNCT ((LAMBDA (D X) FORM D))
            PUSH PARAMS (X)
            POP FUNCT ((LAMBDA (D) FORM D))
            POP PARAMS
            TOP FUNCT ((LAMBDA (D) FORM))
            TOP PARAMS

```

2.2.3. *Axiom (⇒) Phase.* ⇒rules analyse the axioms of the specification.

```

→01. *OP1 READS TRUE, /(*OP1 (*OP2 ... X2 ...)) = X2 <1 IF *COND>/
    =>*OP1 READS <1 IF *COND>, *OP2 WRITES ELEM, *E DENOTES *OP2

    if *OP1 is a read-operation extracting information from a term as described by
    the axiom-pattern in slashes (possibly qualified by a condition *COND)
    then
    *OP1 is a read-operation (possibly under *COND), and *OP2 is the elementary
    construction operator for D-objects which is signalled by permanently binding
    the pseudo-variable *E to the value of *OP2.

```

*Note.* Including an axiom in slashes results in omitting the respective axiom from any further consideration after the respective rule has been applied successfully. The conditioned test <1 IF \*COND> is checked in the consequence-part of the rule if the test has been *established* in the precondition, i.e. the axiom which matches is a conditional axiom.

=01 applies to the axiom A4 (TOP (PUSH D2 X2)) = X2, and results in:

=01:Result. TOP READS PUSH WTIES (ELEM) *E DEMOTES PUSH	TOP reads from D-objects. PUSH writes elementary objects into D-objects. construct new D-objects. *E is a variable bound to PUSH. axiom [A4] will no longer be considered
---	---

=03. LET \*EXPR BE EXPR, /(\*OP1 \*EXPR ...) = UNDEF <1 IF \*COND>/,  
<2 \*EXPR EQUAL D0>  
=> \*OP1 UNDEF IF <1<2{AND}> <2{EQ D D0}> <1 \*COND> <1 2>>.

*if* \*OP1 applied to some expression \*EXPR yields UNDEFINED, possibly under conditions (a) \*COND, and (b) \*EXPR can be shown to be EQUAL D0  
*then* applying \*OP1 returns UNDEFINED under the conditions established in the precondition of this rule. The axiom involved in a successful application of the rule is deleted.

*Note.* A meta-rule makes sure this rule is never applied when neither condition (a) nor (b) holds.

=03 applies to the axioms [A1] and [A2], (POP D0)=UNDEF and (TOP D0) = UNDEF:

=03:Result. POP UNDEF IF (EQ D D0)  
TOP UNDEF IF (EQ D D0)  
axioms [A1] and [A2] are deleted

=06C. \*OP1 CONTINUABLE, \*E CONTINUABLE SINGLE, NOT \*OP1 WRITES,  
(\*OP1 (\*E ... D2 ...) ...) = <1(\*OP1...D2<1...)> <2 IF \*COND>  
=> \*OP1 DELETES <2 IF \*COND>

*if* \*OP1 is a continuable operation extracting a D-object from an object built up by the elementary construction operation E, possibly under condition \*COND  
*then* \*OP1 is a deletion operation whenever \*COND holds (if \*COND is established).

=06C applies only to axiom [A3] (POP (PUSH D2 X2)) = D2:

=06C:Result. POP DELETES

2.2.4. *Representation (R-) Phase.* There is a wealth of conclusions which can be drawn from the results of the previous phases. First, rule R01 decides to implement data types as lists if their elements are constructed by a singly continuable construction operation

R01. \*E CONTINUABLE SINGLE => D LIST  
R01:Result. D LIST

For operations which unconditionally read or delete, rule R04 concludes that their action affects the last (NEWEST) element of their argument which had been added by the elementary construction operation:

R04. LET \*ACTION BE !READS DELETES!, ! ... ! encloses alternatives (ORing)  
\*OP1 \*ACTION  
=> \*OP1 \*ACTION NEWEST  
*if* \*OP1 READS (DELETES) unconditionally  
*then* READING (DELETing) of \*OP1 affects the NEWEST element in its argument

R04 applies to operations POP and TOP:

R04:Result. POP DELETES (NEWEST)  
TOP READS (NEWEST)

There is another rule to detect the OLDEST element in a D-object as affected by some operation.

Although R01 has decided that stacks should be implemented as lists, it is still open what particular kind of list structure -such as singly or doubly linked lists- should be chosen. E.g., the doubly linked list structure is selected when reading is

done at one end, and a counting operation recursively works through the list from the other end. To prepare this decision, we have to find out about whether operations work at the front resp. back end.

<p>R11A. D LIST, NOT D DIRECTED, *OP1 OP IMPLM,          LET *PLACE BE !NEWEST OLDEST! OP1 DELETES PLACE&lt;1 IF *COND&gt;          =&gt; D DIRECTED, *OP1 DELETES FRONT &lt;1 IF *COND&gt;</p>
<p><i>if</i> D is to be implemented as a list, but no direction has been established yet,          and *OP1 is an operation to be implemented,          and *OP1 deletes the NEWEST (resp. OLDEST) elements from D-objects,          possibly under some condition *COND  <i>then</i> *OP1 deletes at the front end of D-objects under condition *COND (if established), and the lists representing D-objects are directed now.</p>
<p>R11A applies to POP:          R11A:Result. D DIRECTED, POP DELETES (FRONT)</p>

*Remark.* After having applied R11A to some operation, this rule is no longer applicable for any other operation, as D is tagged DIRECTED now. This is important because operations may work on opposite ends s.t. choosing one operation to work at the front end implies the other operation works at the back end.

With rule R11A, we have now established what is taken to be the front resp. back end of lists representing D-objects w.r.t. operation POP. The next two rules determine at which end the other operations work.

<p>R13A. D LIST, *OP1 DELETES FRONT, *OP1 DELETES /NEWEST/          =&gt; *E WRITES FRONT</p>
<p><i>if</i> the data type is represented by lists, and *OP1 DELETES the NEWEST element of D-objects, and the NEWEST element is in the FRONT position of the list  <i>then</i> the elementary construction operation *E WRITES into the FRONT position, and the NEWEST-value under property DELETES of *OP1 is deleted.</p>
<p>R13A:Result. PUSH WRITES (FRONT)          POP: NEWEST-value of property DELETES is erased</p>
<p>R15A. LET *ACTION BE !READS DELETES!, *OP1 *ACTION /NEWEST/, *E WRITES FRONT          *OP1 *ACTION FRONT</p>
<p><i>if</i> *OP1 READS (resp. DELETES) the NEWEST D-element,          and the elementary construction operation writes in the FRONT-position  <i>then</i> *OP1 READS (resp. DELETES) in the FRONT-position, and the NEWEST-value of property READS (DELETES) of *OP1 is removed.</p>
<p>R15A:Result. TOP READS (FRONT)          TOP: NEWEST-value of property READS erased</p>

We have now collected all information which makes the next rule choose the singly linked list structure to represent D-objects:

<p>R21. D LIST, EXISTS NO *OP: *OP DELETES BACK AND *OP OP IMPLM,          EXISTS NO *OP: *OP COUNTS BACK AND *OP OP IMPLM          =&gt; D LIST SL</p>
<p><i>if</i> D-objects are represented as lists, and there is no operation to be implemented which deletes or counts at the back end of lists  <i>then</i> D-objects are represented as singly linked lists.  <i>Note.</i> The value IMPLM of property OP of *OP rules out that *OP is a <i>hidden operation</i>.</p>
<p>R21:Result. D LIST SL</p>
<p>Finally, R29 establishes the LISP-representation of the basic D-object DØ(empty stack):</p>
<p>R29. D LIST SL          =&gt; DØ FUNCT (NLAMBDA (FRONT) (SETQ FRONT (CONS)) (RPLACA FRONT FRONT))</p>
<p><i>if</i> D-objects are represented as singly linked lists  <i>then</i> implement the basic D-element DØ by the above LISP-function.</p>

R29:Result. DØ FUNCT (NLAMBDA (FRONT) (SETQ FRONT (CONS)) (RPLACA FRONT FRONT))

2.2.5. *Compile (C-) Phase.* C-rules work on the FUNCT-property of operation names which is a pattern of the LISP-function to implement the respective operation. All strings in such a FUNCT-pattern that are not LISP-atoms are regarded as keywords to be *expanded* to LISP-code by C-rules. C-rules work from front to back ends of FUNCT-patterns. A C-rule inspects the next keyword after the current position. The §-sign indicates the current position pointer.

First, we work on operation PUSH.

C01. D LIST,  
LET \*ACTION BE !READS DELETES WRITES!, LET \*POSITION BE !FRONT BACK!,  
\*OPI \*ACTION \*POSITION, \*OPI FUNCT: /FORM/  
=> (COND UNDETERMINED (T \*ACTION \*POSITION)).

if D is a list, and \*OPI does \*ACTION=READS, DELETES, WRITES) at the  
\*POSITION=(FRONT, BACK) end  
then in the FUNCT-pattern of \*OPI, FORM is replaced with  
(COND UNDETERMINED (T ACTION POSITION))

Note. C-rules differ from all previous rules in that /FORM/ occurring in the precondition is replaced with the value of the consequence of the rule (shorthand notation).

C01:Result. PUSH FUNCT ((LAMBDA (D X) § (COND UNDETERMINED (T WRITES FRONT)) D ))

The next keyword after § is UNDETERMINED; it is simply deleted by rule C10B:

C10B. NOT \*OPI UNDEF IF \*COND, \*OPI FUNCT:/UNDETERMINED/ => .

C10B:Result. PUSH FUNCT ((LAMBDA (D X) (COND § (T WRITES FRONT)) D ))

C15 generates LISP-code from the observation "WRITES FRONT":

C15. D LIST SL, OPI:FUNCT:/WRITES FRONT/  
=> (REPLACE FRONT (CONS X FRONT)) (AND (EQ (CAR URD) D)(RPLACA URD FRONT))

C15:Result. PUSH FUNCT  
((LAMBDA (D X) (COND (T § (REPLACE FRONT (CONS X FRONT))  
(AND (EQ (CAR URD) D)(RPLACA URD FRONT)))) D ))

The next two keywords are both FRONT and are both expanded by rule C33:

C33. \*OPI FUNCT:/FRONT/ => (CDR D).

C33:Result. PUSH FUNCT  
((LAMBDA (D X) (COND (T (REPLACE (CDR D) (CONS X § (CDR D)))  
(AND (EQ (CAR URD) D)(RPLACA URD FRONT)))) D ))

We omit the rules for further expansions and show the final result of compiling PUSH:

C:Result(PUSH). PUSH FUNCT  
((LAMBDA (D X) (COND (T (REPLACE (CDR D)(CONS X (CDR D)))  
(AND (EQ (CAR URD) D)(RPLACA URD (CDR D)))) D ))

Next, we turn to operation POP. Rule C01(above) yields

C01:Result. POP FUNCT ((LAMBDA (D) § (COND UNDETERMINED (T DELETES FRONT)) D ))

Contrary to PUSH, POP is sometimes UNDEFINED, so rule C10B does not apply. Instead, C08 fires:

C08. \*OPI /UNDEFINED IF \*COND/, <1 \*OPI WRITES>, \*OPI FUNCT:/UNDEFINED/  
=> (\*COND (RETURN (QUOTE UNDEF))) <1 UNDETERMINED>.

C08:Result. POP FUNCT  
((LAMBDA (D) (COND § ((EQ D DØ)(RETURN (QUOTE UNDEF)))  
(T DELETES FRONT)) D ))

Our next rule refines the test (EQ D DØ), i.e. "is D the empty stack?":

C08B. LET \*ACTION BE !READS DELETES WRITES!, NOT \*OPI \*ACTION REFERSTO \*N,  
NOT \*OPI COUNTS BACK IF ..., \*OPI FUNCT:/(EQ D DØ)/  
=> (NULL FRONT)

C08B replaces the test (EQ D DØ) with (NULL FRONT):

C08B:Result. POP FUNCT ((LAMBDA (D) (COND(§ (NULL FRONT)(RETURN(QUOTE UNDEF)))  
(T DELETES FRONT)) D ))

Rule C23 expands DELETES FRONT into LISP-code:

```
C23. D LIST SL, *OP1 FUNCT:/(DELETES FRONT/
=> (AND (EQ (CAR URD) FRONT) (RPLACA URD D)) (REPLACE FRONT (CDR FRONT))).
```

Omitting several further steps, the C-phase finally produces the following code for POP:

```
C:Result(POP). POP FUNCT
((LAMBDA (D) (COND ((NULL (CDR D))(RETURN (QUOTE UNDEF)))
(T (AND (EQ (CAR URD) (CDR D)) (RPLACA URD D))
(REPLACE (CDR D) (CDR (CDR D)))))) D ))
```

To implement TOP, we need two additional rules:

```
C13. LET *POSITION BE !FRONT BACK!, *OP1 FUNCT:/(READS *POSITION/
=> (COMPONENT *POSITION)).
```

```
C36. D LIST SL, *OP1 FUNCT:/(COMPONENT/ => CAR.
```

In our example, these two rules determine that TOP reads the CAR-position of lists. Again omitting further steps, the C-phase finally implements TOP in this way:

```
C:Result(TOP). TOP FUNCT
((LAMBDA (D) (COND ((NULL (CDR D)) (RETURN (QUOTE UNDEF)))
(T (CDR D))))))
```

2.2.6. *Cleanup Phase.* The Cleanup phase again works through the FUNCT-property of operation names, and makes optimizations and final expansions. In our example, the following rules apply:

```
CLEAN04. LET *VAR1 BE LIST, *OP1 FUNCT:/(COND (T *VAR1))/ => *VAR1.
any COND-form having only a T-branch is replaced with the T-branch.
```

The following rules replace REPLACE applied to a CAR- resp. CDR-form with RPLACA resp. RPLACD:

```
CLEAN01. LET *VAR BE EXPR, *OP1 FUNCT:/(REPLACE (CAR *VAR)/ => RPLACA *VAR.
```

```
CLEAN02. LET *VAR BE EXPR, *OP1 FUNCT:/(REPLACE (CDR *VAR)/ => RPLACD *VAR.
```

2.2.7. *LISP Phase.* Simulating a CLASS-construct in LISP, the LISP-phase returns the following LISP-code:

```
(PRINT 'STACK)
<DEFINEQ
<STACK
<NLAMBDA (OP DLIST) (SETQ DLIST (EVLIS DLIST))
(PROG ((URD (CAR DLIST)))
(RETURN (NEVALA (APPLY (EVAL OP) DLIST STACK)>
>
<SETQC STACK
(<TOP LAMBDA (D)
(COND ((NULL (CDR D)) (RETURN 'UNDEF)) (T (CAR (CDR D>
(POP LAMBDA (D)
<COND ((NULL (CDR D)) (RETURN 'UNDEF))
(T (AND (EQ (CAR URD) (CDR D)) (RPLACA URD D))
(RPLACD D (CDR (CDR D)> D)
(PUSH LAMBDA (D X)
(RPLACD D (CONS X (CDR D)))
(AND (EQ (CAR URD) D) (RPLACA URD (CDR D)))) D)
(DØ NLAMBDA (FRONT) (SETQ FRONT (CONS)) (RPLACA FRONT FRONT>
```

As an example, the empty stack is generated by

```
(SETQ S (STACK DØ)).
```

Pushing 17 onto the empty stack S is done by

```
(STACK PUSH S 17), and (STACK PUSH S 25) PUSHes 25 onto S so that we obtain
from (STACK TOP S) the value 25, and (STACK POP S), (STACK TOP S) yields 17.
POPPing S twice, i.e. doing (STACK POP S), (STACK POP S) yields UNDEF.
```

### 3. List and Tree Representations of Abstract Data Types

ADTCOMP implements ADT-specifications in terms of a variety of given data structures. This section briefly reviews decision making and coding for three simple structures: (3.1) doubly-linked lists, (3.2) association lists, and (3.3) binary trees.

3.1. *Doubly-Linked Lists*. After the R-phase has established that objects of the data type D are to be represented as lists (i.e. 'D LIST (TRUE)'), one or both of the rules R19A,B may apply:

```
R19A D LIST TRUE, *OP1 OP IMLEM, *OP1 DELETES BACK => D LIST DL
R19B D LIST TRUE, *OP1 OP IMLEM, *OP1 COUNTS BACK IF ... => D LIST DL

if D-objects are to be represented as lists, and there is an operation *OP1 to be
implemented which deletes (counts) at the back end
then D-objects are to be represented as doubly-linked lists.
```

R27 establishes the LISP-implementation of doubly-linked lists by generating code for the basic D-object DØ:

```
R27 D LIST DL
=> DØ FUNCT (NLAMBDA (FRONT) (SETQ FRONT (CONS (CONS)))
(RPLACA (CAR FRONT) FRONT) FRONT)).
```

The following C-rules compile LISP-code for various kinds of operations:

```
C36 D LIST DL, *OP1 FUNCT:/BACK/ => (CAR (CAR D)).
the BACK-end is reached by CAAR.
```

```
C37 LET *REPKIND BE !DL ASSOC!, D LIST *REPKIND ...,
LET *LIST BE LIST, *OP1 FUNCT:/COMPONENT *LIST/
=> (CDR (CAR *LIST)).
the FRONT-element is obtained by applying CDAR.
```

The following four rules generate code (up to further refinements) for operations writing (deleting) at the FRONT-/BACK-position:

```
C16 D LIST DL, *OP1 FUNCT:/WRITES FRONT/
=> (REPLACE FRONT (CONS (CONS D X) FRONT))
(COND ((NULL (CDR FRONT)) (RPLACA (CAR URD) FRONT))
(T (RPLACA (CADR FRONT) FRONT))).
```

```
C19 D LIST DL, *OP1 FUNCT:/WRITES BACK/
=> (REPLACE BACK (CDR (RPLACD BACK (CONS (CONS BACK X)(CDR BACK)))).
```

```
C24 D LIST DL, *OP1 FUNCT:/DELETES FRONT/
=> (COND ((NULL (CDR FRONT))(RPLACA (CAR URD) D)) (T (RPLACA (CADR FRONT) D)))
(REPLACE FRONT (CDR FRONT)).
```

```
C26 D LIST DL, *OP1 FUNCT:/DELETES BACK/
=> (REPLACE BACK (CAAR BACK)) (COND ((EQ URD D)(RPLACD BACK))
(T (RPLACD BACK D))).
```

Compile-rule C03 constructs a schema for implementing operations that count recursively from the back end:

```
C03 D LIST DL, LET *ACTION BE !READS DELETES WRITES!, LET *POSITION BE !FRONT BACK!
LET *MODE !NONE OVER!, NOT *OP1 *ACTION *POSITION, NOT *OP1 *ACTION REFTO *N,
*OP1 COUNTS BACK IF *COND3, <1 *OP1 *ACTION *MODE IF *COND2>,
*COND1 IMPLIES (NOT *COND2), <1 *COND2 IMPLIES (NOT *COND3)>, *OP1 FUNCT:/FORM
=> (COND UNDETERMINED (*COND1 *ACTION BACK) <1 (*COND2 *ACTION *MODE BACK)>
(T (APPLY* *OP1 BACK PARAMS))).
```

*Example:* Two-way list. An algebraic specification of a two-way list is:

```
DATATYPE: TWO-WAYLIST D; COMPONENTSORT X; OTHER SORTS: A = NAT;(NAT=nat'Z numbers)
OPERATIONS: ADD1: D X → D, ADD2: D X → D, REMOVE1: D → D, REMOVE2: D → D,
READ1: D A → X, READ2: D A → X;
AXIOMS: (ADD1 DØ X) = (ADD2 DØ X), (REMOVE1 DØ) = UNDEF, (REMOVE2 DØ) = UNDEF,
(ADD1 (ADD2 D2 X2) X) = (ADD2 (ADD1 D2 X) X2),
(REMOVE1 (ADD2 D2 X2)) = D2 IF (EQ D2 DØ)
= (ADD2 (REMOVE1 D2) X2) IF (NOT (EQ D2 DØ)),
(REMOVE2 (ADD2 D2 X2)) = D2, (READ1 DØ A) = UNDEF, (READ2 DØ A) = UNDEF.
```

```

(READ1 (ADD2 D2 X2) A) = X2 IF (AND (EQ D2 D0) (EQ A 1)),
                      = (READ1 D2 A) IF (AND (NOT (EQ D2 D0)) (EQ A 1)),
                      = (READ1 (REMOVE1 (ADD2 D2 X2)) (SUB1 A) IF (NOT (EQ A 1)))
(READ2 (ADD2 D2 X2) A) = X2 IF (EQ A 1),
                      = (READ2 D2 (SUB1 A)) IF (NOT (EQ A 1));

```

This specification is compiled to the following LISP-code:

```

<DEFINEQ <TWOWAYLIST
<NLAMBDA (OP DLIST) (SETQ DLIST (EVLIS DLIST))
                  (PROG ((URD (CAR DLIST)))
                        (RETURN (NEVALA (APPLY OP) DLIST) TWOWAYLIST>>
>
{SETQ TWOWAYLIST
  (<READ1 LAMBDA (D A)
    (COND ((EQ (CAR (CAR D)) URD) (RETURN 'UNDEF))
          <(EQ A 1) (CDR (CAR (CAR (CAR D)
    (T (APPLY* READ1 (CAR (CAR D)) SUB1 A)
  <READ2 LAMBDA (D A)
    (COND ((NULL (CDR D)) (RETURN 'UNDEF))
          <(EQ A 1) (CDR (CAR (CDR D)
    (T (APPLY* READ2 (CDR D) (SUB1 A)
  (REMOVE2 LAMBDA (D)
    <COND ((NULL (CDR D)) (RETURN 'UNDEF))
          (T (COND ((NULL (CDR (CDR D))) (RPLACA (CAR URD) D))
                (T (RPLACA (CADR (CDR D)) D)))
            (RPLACD D (CDR (CDR D) D)
  (REMOVE1 LAMBDA (D)
    (COND ((NULL (CDR D)) (RETURN 'UNDEF))
          (T <RPLACA (CAR D) (CAAR (CAR (CAR D)
            (COND <(EQ URD D) (RPLACD (CAR (CAR D)
              (T (RPLACD (CAR (CAR D)) D)) D)) D)
  (ADD1 LAMBDA (D X)
    <RPLACA (CAR D) (CDR (RPLACD (CAR (CAR D))
            (CONS (CONS (CAR (CAR D)) X) (CDR (CAR (CAR D) D D)
  (ADD2 LAMBDA (D X)
    (RPLACD D (CONS (CONS D X) (CDR D)))
    <COND ((NULL (CDR (CDR D))) (RPLACA (CAR URD) (CDR D)))
          (T (RPLACA (CADR (CDR D)) (CDR D) D D)
  (D0 NLAMBDA (FRONT) (SETQ FRONT (CONS (CONS)) (RPLACA (CAR FRONT) FRONT) FRONT)))

```

3.2. *Association Lists.* If components of a data type D are accessed via an index sort, then D-objects are represented as association lists. This decision is made by rules of the  $\rightarrow$  and  $\rightarrow$ -phase. As an example, we show how ADTCOMP implements the flexible record abstraction.

*Example:* Flexrecord. In our specification, a predefined operation LEGAL? checks whether an object of the componentsort X may be entered into a record under a particular index.

```

DATATYPE: FLEXRECORD D; COMPONENTSORT: X; OTHER SORTS: I = NAT, B = BOOL;
OPERATIONS: ASSIGN: D I X  $\rightarrow$  D, DROP: D I  $\rightarrow$  D, READ: D I  $\rightarrow$  X;
GIVEN OPERATION: LEGAL?: I X  $\rightarrow$  B;
AXIOMS: (READ D0 I) = UNDEF,
        (READ (ASSIGN D2 I2 X2) I) = X2 IF (EQ I I2)
        = (READ D2 I) IF (NOT (EQ I I2)),
        (ASSIGN (ASSIGN D2 I2 X2) I X) = (ASSIGN D2 I X) IF (AND (EQ I I2) (LEGAL? I X)),
        = UNDEF IF (NOT (LEGAL? I X)),
        (ASSIGN D0 I X) = UNDEF IF (NOT (LEGAL? I X)),
        (DROP D0 I) = UNDEF,
        (DROP (ASSIGN D2 I2 X2) I) = D2 IF (EQ I I2),
        = (ASSIGN (DROP D2 I) I2 X2) IF (NOT (EQ I I2));

```

This specification is implemented by ADTCOMP in this way:

```

<DEFINEQ <FLEXRECORD
<NLAMBDA (OP DLIST) (SETQ DLIST (EVLIS DLIST))
                  (PROG ((URD (CAR DLIST)))
                        (RETURN (NEVALA (APPLY (EVAL OP) DLIST) FLEXRECORD>>
>

```



```

<SETQ FLEXRECORD
  (<READ LAMBDA (D I)
    (COND ((NULL (CDR D)) (RETURN 'UNDEF))
      (<EQ I (CAAR (CDR D)) (CDR (CDR D)
        (T (APPLY* READ (CDR D) I)
          (DROP LAMBDA (D I)
            (COND ((NULL (CDR D)) (RETURN 'UNDEF))
              (<EQ I (CAAR (CDR D)) (RPLACD D (CDR (CDR D)
                (T (APPLY* DROP (CDR D) I))) D)
            (ASSIGN LAMBDA (D I X)
              (COND ((NOT (LEGAL? I X)) (RETURN 'UNDEF))
                (<NULL (CDR D)) (RPLACD D (CONS (CONS I X)
                  ((AND (EQ A (CAAR (CDR D))) (LEGAL? I X))
                    (RPLACD (CAR (CDR D)) X) D)
                (T (APPLY* ASSIGN (CDR D) I X))) D)
              (DØ NLAMBDA NIL (CONS)>:

```

3.3. *Binary Trees*. An ADT-specification is implemented in terms of trees if there are  $n$ -fold continuable operations (see 2.2.2) with  $n \geq 2$ . We consider  $n=2$ , i.e. binary tree representations.

R-rule R30 generates code for the empty binary tree:

```
R30. D BINTREE = DØ FUNCT (NLAMBDA () (CONS)).
```

C04 sets up the structure of tree-operations to be expanded subsequently:

```

C04. D BINTREE, LET *ACTION BE !READS DELETES WRITES!,
  LET *MODE BE !NONE ACROSS FROM!, <1 *OPI UNDEFINED IF *B1>,
  <2 *OPI *ACTION IF *B2>, <3 *OPI *ACTION *MODE IF *B3>,
  *OPI CHOPS P1 IF *B4, *OPI CHOPS P2 IF *B5,
  <1 <2 *B1 IMPLIES (NOT *B2)>, <1 <3 *B3 IMPLIES (NOT *B3)>,
  <1 *B1 IMPLIES (NOT *B4)>, <1 *B1 IMPLIES (NOT *B5)>,
  <2 <3 *B2 IMPLIES (NOT *B3)>, <2 *B2 IMPLIES (NOT *B4)>,
  <2 *B2 IMPLIES (NOT *B5)>, <3 *B3 IMPLIES (NOT *B4)>, <3 *B3 IMPLIES (NOT *B5)>
  *B4 IMPLIES (NOT *B5), *OPI FUNCT:/FORM/
=> (COND <1 (*B1 (RETURN (QUOTE UNDEF)))>
    <2 (*B2 *ACTION FRONT)>
    <3 (*B3 *ACTION *MODE FRONT)>
    (*B4 (APPLY* *OPI (CAR FRONT) PARAMS))
    (T (APPLY* *OPI FRONT PARAMS))).

```

*Note.* Nodes in trees are accessed via prefix-closed strings. An operation that "works through a tree" proceeds from node to node by CHOPPING off a character from the access-string. Such an operation has the property CHOPS.

*Example.* As an example, we present the implementation of the usual binary tree specification which contains a doubly continuable hidden operation.

```

DATATYPE: BINTREE D; COMPONENTSORT X; OTHER SORTS: A = STRING(NAT);
OPERATIONS: INSERT: D A X + D, DELETE: D A - D, READ: D A + X;
AUXOPERATION: CONS: D D X + D;
AXIOMS: (INSERT DØ A X) = (CONS DØ DØ X) IF (EQ (FIRST A) Ø),
        = UNDEF IF (NOT (EQ (FIRST A) Ø)),
  (INSERT (CONS D12 D22 X2) A X)
    = (CONS D12 D22 X) IF (EQ (FIRST A) Ø),
    = (CONS (INSERT D12 (REST A) X) D22 X2) IF (EQ (FIRST A) 1),
    = (CONS D12 (INSERT D22 (REST A) X) IF (EQ (FIRST A) 2),
  (DELETE DØ A) = UNDEF,
  (DELETE (CONS D12 D22 X2) A) = DØ IF (EQ (FIRST A) Ø),
    = (CONS (DELETE D12 (REST A)) D22 X2)
      IF (EQ (FIRST A) 1),
    = (CONS D12 (DELETE D22 (REST A)) X2)
      IF (EQ (FIRST A) 2),
  (READ DØ A) = UNDEF,
  (READ (CONS D12 D22 X2) A) = X2 IF (EQ (FIRST A) Ø)
    = (READ D12 (REST A)) IF (EQ (FIRST A) 1),
    = (READ D22 (REST A)) IF (EQ (FIRST A) 2);

```

*Note.* FIRST, REST are string operations.

From this specification, ADTCOMP generates the following LISP-code:

```
<DEFINEQ <BINTREE
<NLAMBDA (OP DLIST) (SETQ DLIST (EVLIS DLIST))
              (PROG ((URD (CAR DLIST)))
                    (RETURN (NEVALA (APPLY (EVAL OP) DLIST) BINTREE>>
>SETQ BINTREE
  (<READ LAMBDA (D A)
    (COND ((NULL (CDR D)) (RETURN 'UNDEF))
          <(EQ (FIRST A) 0) (CAR (CAR (CDR D)
            ((EQ (FIRST A) 1) (APPLY* READ (CAR (CDR D)) (REST A)))
            (T (APPLY* READ (CDR D) (REST A)
  (DELETE LAMBDA (D A)
    (COND ((NULL (CDR D)) (RETURN 'UNDEF))
          ((EQ (FIRST A) 0) (RPLACD D))
          ((EQ (FIRST A) 1) (APPLY* DELETE (CAR (CDR D)) (REST A)))
          (T (APPLY* DELETE (CDR D) (REST A) D)
  (INSERT LAMBDA (D A X)
    (COND ((AND (NULL (CDR D)) (NOT (EQ (FIRST A) 0))) (RETURN 'UNDEF))
          <(AND (NULL (CDR D)) (EQ (FIRST A) 0)) (RPLACD D (CONS (CONS X)
            ((EQ (FIRST A) 0) (RPLACA (CAR (CDR D)) X) D)
            ((EQ (FIRST A) 1) (APPLY* INSERT (CAR (CDR D)) (REST A) X))
            (T (APPLY* INSERT (CDR D) (REST A) X))) D)
  (DO NLAMBDA NIL (CONS>
```

#### 4. Final Remarks and Conclusions

4.1. *Interactive System Development and Theory Formation.* The development of ADTCOMP has been evolutionary, driven by considering more and more examples, and feeding the programming knowledge necessary to implement the examples into the production bases. To support codifying and incorporating programming knowledge, ADTCOMP has an extensive interactive user interface. This interface helps a user to find out why ADTCOMP failed, coding new rules and experimenting with them, and finally changing meta-rules and the production control unit after inserting new rules into the production bases. The user interface is extremely important for *upgrading* the production bases: whenever a new portion of programming knowledge has been fed to ADTCOMP, it is usually in order to condense rules into fewer and more general rules. *Rule condensation* actually contributes to the formation of a *theory of programming*, and we expect such a theory to eventually evolve from our investigation.

4.2. *Accomplishments.* Previous systems start at a much more concrete level of data structures such as sets, collections, etc., and implement them at the somewhat more "concrete" level of lists, arrays, etc. ADTCOMP generates code from representation-free *axiomatizations* of all structures previous systems could implement, and in addition succeeded for about 60 other axiomatizations. ADTCOMP generates clusters of routines which may be mutually recursive. Previous systems could not introduce recursions.

4.3. *Automatic Programming Environment.* ADTCOMP is part of the automatic programming environment PEN currently under development. In PEN, a user specifies structures in terms of equations about the behavior of data objects under operations intended to manipulate them. The system analyses such specifications, and guides the user to make them consistent and complete. The user then develops algorithms in terms of self-defined or library-supplied data abstractions, obtaining abstract algorithms to be compiled into efficient and machine-executable code by ADTCOMP.

4.4. *Current Work.* ADTCOMP is currently being extended in three directions:

(1) Generating code in other target languages; (2) an extension to generate code from parameterized ADT-specifications; (3) compiling abstract algorithms.

#### 5. References

- [ADJ 76] Goguen, J.A., J.W. Thatcher, E.G. Wagner. An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types. IBM Research Rept. RC-6487 (1976).
- [ADJ 79] Thatcher, J.W., E.G. Wagner, J. B. Wright. Data Type Specification: Parametrization and the Power of Specification Techniques (Revised). IBM Res. Rept. (1979).
- [BAR 77] Barstow, D.R. Automatic construction of algorithms and data structures using a knowledge base of programming rules. Rept. STAN-CS-77-641(77), Stanford U.
- [LON 77] Long, W.J. A program writer. Rept. TR-187. MIT Lab. for Comp. Sci. (1977).
- [NOR 79] Nordström, M. LISP-F3 Reference Manual. Uppsala Computing Center (1979).

TOWARDS AN AUTOMATED DEBUGGING ASSISTANT FOR NOVICE PROGRAMMERS

Marc Eisenstadt  
Joachim Laubsch

The Open University  
Milton Keynes, ENGLAND MK7 6AA

ABSTRACT

This paper describes the design philosophy underlying a debugging assistant which will help novice programmers debug arbitrary programs of their own design. The assistant synthesizes plans which are suitable for carrying out a particular set of intentions. The intentions may either be specified in advance (by us), or obtained from the student at debugging-time. The plans are represented in an abstract plan-language, and are compared against a similar representation of the student's own code in an attempt to find anomalies. The system focusses mainly on telicological bugs (unachieved intentions), but can also help to pinpoint the source of more complex conceptual bugs, even though it cannot recommended specific patches.

1. INTRODUCTION AND OVERVIEW

This paper describes work currently in progress on the design and implementation of a 'debugging assisant' for psychology students learning about artificial intelligence. The approach is novel in that it combines three aspects of software design: (a) friendly user-aids of the 'do-what-I-mean' variety [Teitelman, 1974; Wertz, 1978]; (b) expert debuggers which know the algorithm for which the student is writing code [Ruth, 1974; Adam and Laurent, 1977]; (c) domain-independent program understanders which try to 'make sense' of programs on the student's own terms [Rich and Shrobe, 1978; Lukey, 1978]. The debugging assistant will be described in terms of the way we envisage it working, but it has not yet been implemented.

Background

A course on cognitive psychology at the Open University (UK) requires each of its 400 students to do some elementary artificial intelligence programming. The students are given access to a software environment called SOLO, which can be thought of as a variant of LOGO, with a small number of primitives for manipulating semantic networks rather than lists. SOLO procedures work by side-effecting a data base of assertions, which all have the form of associative triples. Thus, SOLO's primitives correspond to the usual ASSERT, ERASE, and FETCH of assertional data base packages. As an illustration, the simple SOLO program shown below, when invoked by typing FRIENDTEST FRED, will assert the triple (FRED ISFRIENDLYWITH SAM) if and only if the triple (FRED DRINKSWITH SAM) is present in the data base. The variable Y, below, would be bound to SAM as a result of the pattern-match at step 1:

```

TO FRIENDTEST /X/
1 CHECK /X/---DRINKSWITH---> ?Y
  1A If present: NOTE /X/---ISFRIENDLYWITH---> *Y; EXIT
  1B If absent: PRINT "I GIVE UP"; EXIT
DONE

```

Students are presented with a handful of cognitive modelling tasks to challenge their skills as cognitive scientists. For instance, they are asked to write a program to perform two-column subtraction. Since SOLO has no numerical primitives, the students must invent a representation for numbers in the data base, invent all of the primitive operations for working with pairs of numbers (e.g. fetching the result of  $X \text{ MINUS } Y$ ), and then tackle the problems of borrowing which arise in multi-column subtraction.

### Design philosophy

SOLO itself is a relatively friendly environment for beginners, since it traps or pre-empts a large proportion of lexical and syntactic errors. Our philosophy is that students should spend their time working only on 'interesting' errors in their programs. We foresee students explicitly requesting help when they are in trouble. Our debugging assistant would intervene, and interact with the student. The purpose of the interaction would be to determine what the student was trying to achieve, and to help to isolate the culprit.

The debugging assistant is organized into four main modules: an intent-specifier, an instantiator, a coder, and a translator. The job of the intent-specifier is to depict the overall aim of the code which a student is trying to debug. This aim is typically expressed in terms of changes to a global data base. The intent-specifier has access to a plan library, which includes a repertoire of low-level plans for achieving particular ends, and higher-level plans which specify certain intentions known to be relevant to problems which our students will be working on. If we know precisely the problem the student is working on, then we can supply the intent-specifier with useful domain-dependent knowledge (e.g. knowledge about subtraction). If we don't know this, then the intentions must be obtained from the student at debugging-time.

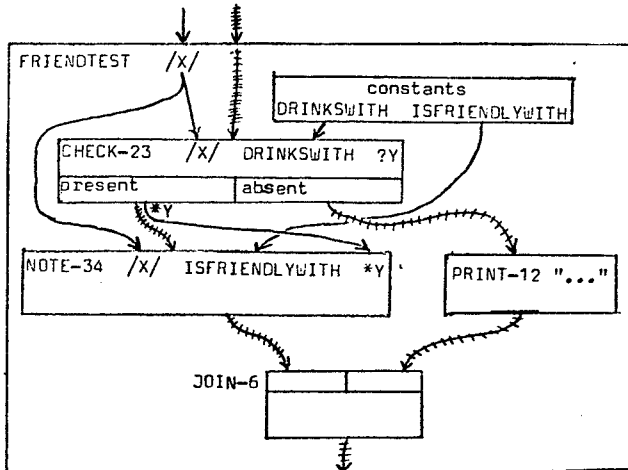
The job of the instantiator is to work out how best to achieve a particular intention, given the database representation which the student has chosen to use. The student's data base is analyzed in some detail, and the instantiator proposes several possible plans for carrying out the intentions handed down by the intent-specifier.

The coder takes the output of the instantiator, and uses knowledge of SOLO to produce a formal plan suitable for execution by a SOLO virtual machine (expressed in terms of the conceptual FETCH and ASSERT primitives, rather than in actual SOLO syntax). This plan is expressed in a plan-language very similar to that developed by Rich and Shrobe [1976]. The plan-language itself uses a KRL-like notation [Bobrow and Winograd, 1978].

The translator works in the other direction, taking actual SOLO code produced by a student, and translating it into the plan-language. This enables the student's code to be compared directly with the abstract plan, allowing minor perturbations to be ignored. Mismatches between the 'ideal' and the 'actual' plan are then articulated for the student's benefit.

2. PLANS AND INTENTIONSPlan diagrams

A language-independent graphical notation for describing control and data flow was first used by Rich and Shrobe [1978] to describe LISP and FORTRAN programs. Our SOL0 debugging assistant translates student programs into a list structure version of that diagram language. The diagram below depicts our debugging assistant's representation of the FRIENDTEST procedure described in section 1. Control flow is indicated by cross-hatched arrows, while data flow is shown by plain arrows:



Each box above depicts what is known as a 'segment'-- these are modules which have behavioral descriptions which specify both their effects and any sub-segments which are responsible for producing those effects. These behavioral descriptions are specified in our plan-language notation, a portion of which is illustrated below:

```

friendtest =
[ a segment with
  name: friendtest
  steps: (check-23 note-34 join-6 print-12)
  indata: (x)
  boundvars: (y)
  constants: (drinkswith isfriendlywith)
  entrystep: check-23
  exitstep: join-6]

check-23 =
[ a check-unbound with
  indata: (a triple with [source: x] [rel: drinkswith] [target: ?y])
  occursin: friendtest
  effect:
    (cases
      [present (x drinkswith ?) => successor: note-34]
      [absent (x drinkswith ?) => successor: print-12])]

```

This KRL-like notation is used to describe user programs as well as library plans of common modules. From this notation the debugging assistant derives the effect of a procedure, considering the effect of each step which is part of the procedure.

The semantics of primitive SOLO-steps is represented in a similar fashion, e.g.:

```
check-call = [ a SOLO-primitive-call with
               name: check
               predecessor: (a step) (not (a check-call))]

check-unbound = [ a check-call with
                  indata: (a triple with
                           source: (One-of [my occursin's constants])
                                      (One-of [my occursin's parlist])
                                      (One-of [my predecessor's boundvars]))
                  rel: ...
                  target: ...)
                ; each argument is either a constant or a parameter
                ; of the embedding procedure, or dynamically bound
                ; at least at the preceding step
                effect: (cases
                        [present(my indata) => successor: (a step)]
                        [absent (my indata) => successor: (a step)])]
```

One kind of check-call is a check-unbound. The description above tells how to recognize it, and what effect it has. Each occurrence of CHECK in a SOLO program is an instance of one type of check-call. Some bugs can be detected already, if a primitive-call does not match its schema, e.g. if CHECK had been called with a parameter not occurring in this procedure.

#### A library of domain-independent plans

Frequently used sequences of steps are described as abstract schemata and stored in a plan library. An instance of a library plan can be recognized in a particular user program by comparing the effect descriptions. This is independent of the way a user has broken up the plan into segments or has specialized it by using constants in place of variables.

An example of a domain-independent plan is 'assignment' which has the effect:

```
(make (The target from (a triple with [source: x] [rel: p]))
      (The target from (a triple with [source: y] [rel: q])))
```

where the 'source', 'rel', and 'target' of a triple refer respectively to the first, second, and third elements of an associative data base structure such as (FRED DRINKSWITH SAM).

There are several possible SOLO programs which all have this effect. Some are specializations which arise through substitution of the plan-variables (x, y, p, q) by constants or bound variables.

The effect of a segment can be symbolically derived from the effects of its parts. This may reveal to the SOLO-specialist the existence of some bug if the segment has no or only error effects. As an example, consider a procedure from an actual buggy user program in the subtraction domain.

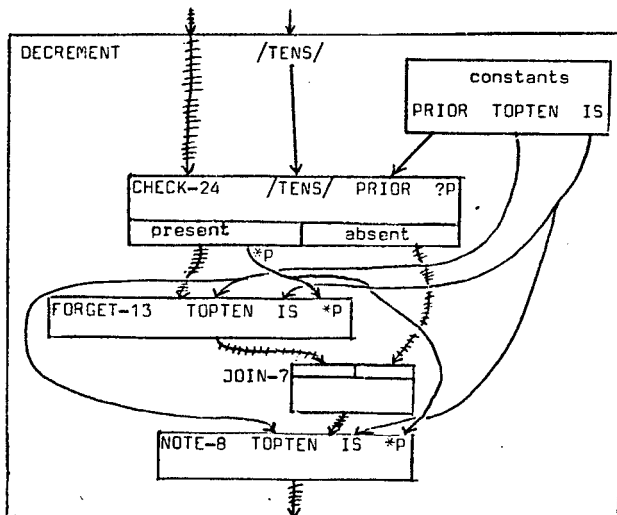
```

TO DECREMENT /TENS/
1 CHECK /TENS/---PRIOR---> ?P
  1A If present: FORGET TOPTEN---IS---> *P; CONTINUE
  1B If absent: CONTINUE
2 NOTE TOPTEN---IS---> *P
DONE

```

This procedure looks up the 'predecessor' of some number in the data base. If found, it is bound to the variable P. DECREMENT then erases the triple (TOPTEN IS \*P) from the data base at step 1A, only to re-assert it again at step 2. In other words, it has no actual effect if the 'predecessor' is found. Even worse, if the predecessor is not found, the procedure carries on to step 2 anyway (via step 1B) only to find that P is now unbound.

The figure below depicts the plan-diagram representation of this procedure:



The effect of this segment found by symbolic evaluation is:

```

(cases [present (/TENS/ PRIOR ?>p) =>
  (cases [present (TOPTEN IS ?=p) => nop]
    [t => error])])
[t => error]

```

Thus, there is either no effect at best ('nop'), or an error at worst, and the inevitability of this can be pointed out to the student. From the plan diagram for an entire program, symbolic evaluation can reveal inconsistencies between a segment's actual effect and its intended effect.

In this example, the intended effect can be obtained from the student interactively. This is done by first asking the student to specify a typical invocation of the troublesome procedure:

PLEASE ILLUSTRATE A TYPICAL USAGE OF DECREMENT, E.G.  
 DECREMENT FOO  
 WHEN YOU SEE THE '>>' PROMPT BELOW:

>>DECREMENT 7

Then the student is asked to specify the expected effect in SOLO notation:

WHAT TRIPLES (IF ANY) WOULD YOU EXPECT TO BE NOTED INTO THE  
 DATABASE BY DECREMENT?

>>TOPTEN---IS---6

Step NOTE-8 in the user's program does in fact match the expected effect:

(make (The target from (a triple with [source: TOPTEN] [rel: IS])) 6)

The first step (CHECK-24) retrieves p=6 as

(The target from (a triple with [source: /TENS/] [rel: PRIOR]))

If the triple (7 PRIOR 6) is present in the student's data base (as it should be), the standard assignment plan can be instantiated with x=TOPTEN, y=/TENS/, p=IS, q= PRIOR, and used to supply the missing step.

Several of the programming schemata that Shrobe et.al. [1979] identified in the analysis of Lisp programs have also been found in beginner's SOLO programs. Some others, particular to programming with associative nets (e.g. testing for a common target from two source nodes) also have to be included in the plan library.

### 3. ANALYZING A PLAN FOR SUBTRACTION

#### A sample task

One sub-task required of students working on multi-column subtraction is to write a procedure DIFFERENCE, which finds the difference of two integers X and Y, each in the range 0-9, provided that X is greater than or equal to Y. The actual code turns out to be self-evident, once the student has decided upon a data representation. One solution arrived at by some of our students is to use SOLO's relational network to construct subtraction tables. This representation constrains the code which the student must generate to solve this sub-task, as illustrated by the sample database and procedure shown below:

7	TO DIFFERENCE /X/ /Y/
'---0--->7	1 CHECK /X/---/Y/---> ?D
'---1--->6	1A If present: NOTE ANSWER---IS---> *D; EXIT
'---2--->5	1B If absent: PRINT "I GIVE UP"; EXIT
'---etc.--->etc.	DONE



Some students realize that using numbers as relation names can be rather limiting in the long run, because they may want to start writing procedures to do addition. In this case, they need a way to discriminate between addition and subtraction tables. One solution is to use nodes such as '7SUBTABLE' and '7ADDTABLE' as follows:

7SUBTABLE	7ADDTABLE
'---0--->7	'---0--->7
'---1--->6	'---1--->8
'---2--->5	'---2--->9
'---etc.--->etc.	'---etc.--->etc.

The only trick is to get from /X/ (one of the inputs to DIFFERENCE) to the appropriate table. This can be done with the following representation:

7	6
'---SUBTRACTION--->7SUBTABLE	'---SUBTRACTION--->6SUBTABLE
'---ADDITION--->7ADDTABLE	'---ADDITION--->6ADDTABLE
etc.	

Since an extra link must be traversed to get from a node (e.g. '7') to the appropriate table (e.g. '7SUBTABLE'), we will call this representation an 'indirect table', in contrast to the 'direct table' representation presented first. There are several other possible representations, but we will just focus on these two. The 'indirect table' representation, once chosen, constrains the code which the student must generate to solve the problem. First the table must be fetched from the input /X/, and then that table must be used to perform the table lookup. So an additional step is necessary, as shown below:

```

TO DIFFERENCE /X/ /Y/
1 CHECK /X/---SUBTRACTION---> ?T
  1A If present: CONTINUE
  1B If absent: PRINT "UH OH"; EXIT
2 CHECK *T---/Y/---> ?D
  2A If present: NOTE ANSWER---IS---> *D; EXIT
  2B If absent: PRINT "MAYBE" /Y/ "IS BIGGER THAN" /X/; EXIT
DONE

```

The new step 1 fetches the indirect table, which is used at step 2 as the 'base' from which an indexed retrieval of the difference D is performed. The important point here is the way in which representation and code interact. The code derives its meaning from the interrelations among structures in the data base. Our instantiator is only given a high-level intent-specification for DIFFERENCE, which it may choose to instantiate in one of several ways. It does this by developing a plan which is suitable for the data representation which has been chosen by the student. This representation is analyzed using specific knowledge about associative networks, as described in the next two sections.

A formal description of the data base

The concepts of 'direct table' and 'indirect table' have the following formal descriptions:

```
direct-table = [a table with
  name: (an integer)
  base: (my name)           ;base and name are coreferential
  content: (an alist with
    typical-pair: [left: (an integer)
                  right: (an integer)])
  variety: (one-of add sub mult)
  constraint: apply [(my variety)
                    [(my base)
                     (The left from
                      (any (pair = p) of (my content))))]]
               = (The right from p) ]
```

```
indirect-table = [a table with
  name: (The target from (a tableptr with source: (my base)))
  base: (an integer)
        (The source from (a tableptr with target: (my name)))
  content:
  variety: } as in direct-table
  constraint: }
```

Using the direct and indirect subtraction tables for the number 7 shown earlier, we may interpret these descriptions as follows: a direct table has a number base which happens to be the same as its own name ('7', in this example). The direct table consists of an association list, which is just pairs of integers. A constraint which must be fulfilled is that if we apply a function (e.g. 'sub' or 'add') to two arguments-- the base (e.g. '7') and the left element of some pair (e.g. '2')-- then the result should be equal to the right element of that pair (e.g. '5', since  $SUB(7,2)=5$ ). The interpretation of indirect-table is almost identical, with one crucial exception-- the relationship between its 'name' and 'base' slots. The name is some item which is specified by saying that it is the target (i.e. the thing pointed to) from some particular tableptr (defined below) whose source is the integer base ('7', in this example). A tableptr is an instance of a triple which relates an integer to a table. Thus, the relational structure (7 SUBTRACTION 7SUBTABLE) is an instance of a tableptr.

Instantiating an appropriate plan

In analyzing the student's data base each subclass of table is hypothesized in turn as a possible data representation, and confirmation of these hypotheses is then sought. Suppose that in this example, our student is in fact using an indirect-table representation. The instantiator focusses on the 'content' slot of a data base object whenever possible. Confirmation is initially sought by looking for nodes containing pairs of integers dangling from them. Such nodes are indeed found, and examination of the other slots eventually leads to a disconfirmation of the 'direct-table' hypothesis and a confirmation of the 'indirect-table' hypothesis.

Next, the instantiator examines the representation of DIFFERENCE, which primarily depicts its intentions. It is the job of the instantiator to generate a particular manifestation of the detailed plan, based upon an analysis of the data representation the student has already chosen. If our student used indirect-tables, the instantiator would generate an idealized plan which used the indirect style of fetching exhibited by the SOLO DIFFERENCE procedure shown earlier.

The intent-specification of a difference-plan is shown below:

```
difference = [a plan with
  input1: (an integer) = X
  input2: (an integer) = Y
  output: (The (access-of Y) from
    (a table with
      variety: sub
      base: X))]
```

The expression (access-of Y) means that simple slot-retrieval is not possible, but instead there is some procedure attached to the 'access-of' slot of the table which will specify how to perform the retrieval. For direct and indirect tables the 'access-of' method attached to their schemata is:

```
access-of i => (The right from
  (The pair from
    ((my content) with left: i)))
```

To illustrate the use of this method, suppose the direct-table representation for '7' includes the pair (2 5). If we want to access some element indexed by '2', then we do that by retrieving the right half of the pair whose left half is '2'. In this case, the pair (2 5) has a right half '5', so if we sent the message 'access-of 2' to the instance of direct-table whose name was '7', the result would be '5'.

From this method the SOLO coder first constructs a description of the intended effect of accessing each type of table. For a direct table the intended effect is:

```
(The target from
  (a triple with source: X rel: Y))
```

For an indirect table the intended effect is:

'outer form'
<pre>(The target from   (a triple with     source: (The target from       (a tableptr with source: X))     rel: Y))</pre>
'inner form'

The SOLO coder knows that an instance of fetch will handle the 'outer form' shown in the rectangle above, so when it processes the intended effect of an indirect-table, it produces the following virtual machine specification:

```
F1 = [a fetch with
  input1: (The target from (a tableptr with source: X)) = T
  input2: Y
  output: (a node) = A ]
```

The 'inner form' is processed analogously, leading to the following virtual machine specification:

```
F2 = [a fetch with
      input1: X
      input2: SUBTRACTION
      output: (a node) = T ]
```

The SOLO coder knows that the final output from a plan can be expressed either with a print or a data base 'assert' operation, so one possible final representation for the idealized plan includes, in its 'effect' slot, the following:

```
The effect from
(an assert with
  arg1: (a constant)
  arg2: (a constant)
  arg3: (The output from (a fetch with
                                input1: (The output from (a fetch with
                                                                input1: X
                                                                input2: SUBTRACTION)))
                                input2: Y)))
```

The translator can map the student's program into a comparable notation. Mismatches between the student's program and the idealized program are then dealt with in the same way as those for the DECREMENT example presented in section 2.

#### 4. DEBUGGING MULTI-COLUMN SUBTRACTION PROGRAMS

The techniques of plan-instantiation described above can be applied equally well to more complex problems, such as multi-column subtraction. In this case, the intent specification is depicted as a segment consisting of about 15 subsegments, representing intermediate plans such as 'FINDCOLUMN', 'PROCESSCOLUMN', 'BORROW', 'DIFFERENCE' etc. As in the simple 'DIFFERENCE' example discussed above, the students may implement one of several possible algorithms. Fortunately, each solution is tied to a particular data base representation. Thus, just as the instantiation of an ideal 'DIFFERENCE' plan was dictated by the student's chosen data structures, so is the instantiation of an ideal 'BORROW' plan dictated by data structures already present when the student asks for help.

The debugging assistant views particular segments not only as instances of domain-independent plans (such as the 'assignment' plan), but also as instances of domain-dependent plans (such as the 'processcolumn' plan). The data flowing in and out of segments inherit properties from the more abstract plan description. One benefit of attaching semantic descriptions to the data flow is the ability to detect type errors. Another benefit is that it enables recognition of the user-intended effect of other modules.

In case the student is working on a subproblem for which the intent-specification is unknown, the debugging assistant tries to obtain such a specification, in SOLO notation, through a dialogue with the student. Typically, this can only be done for a single sub-procedure at a time.

Once the intentions are known, the debugging assistant can then generate a refutation, by symbolic evaluation of the plan description, which shows why the sub-procedure is destined to fail. If the intent-specification has been supplied by us in advance, then it is also possible for the debugging assistant to generate counterexamples which illustrate the procedure working improperly.

In case there are segments missing from an otherwise correct program, or items are accidentally left out of the data-base, the assistant could give a counterexample in which the student's solution fails to consider this case. For instance, the student may not have included cases for  $x - x = 0$  or  $x - 0 = x$  in his tables, and be asked to augment DIFFERENCE to include these cases or change his tables accordingly.

So far we have been talking about teleological bugs [Model, 1979]. These are bugs where the intentions are known (residing in the plan library or being obtained from the student at debugging time), but the code fails to fulfill the intention. A far more difficult class of bugs are 'conceptual' bugs, where the intentions themselves are improperly conceived. Our debugging assistant, through the use of an annotated back-trace, can help the student to see where things have gone wrong. Performing actual reasoning about these wrongly-specified intentions is still well beyond the scope of our project, but we see this as an exciting and important area for future research.

## REFERENCES

- Adam, A. and Laurent, J.P. "Transformation de Programmes et Correction de Programmes." Coll. sur l'Intelligence Artificielle, Strasbourg, CNRS, 1977, 41-83.
- Bobrow, D. and Winograd, T. "An overview of KRL-0." Cognitive Science, Vol. 1, 1978.
- Lukey, F. "Understanding and debugging simple computer programs." Ph.D. Thesis, Univ. of Sussex, 1978.
- Model, M.L. "Monitoring system behaviour in a complex computational environment", Xerox, Palo Alto Research Center, Ca., CSL-79-1, 1979.
- Rich, C. and Shrobe, H.E. "Initial report on a Lisp programmer's apprenticeship." IEEE Transactions on Software Engineering, SE-4:6, 1978, 456-467.
- Ruth, G.R. "Analysis of algorithm implementations." MIT, MAC-TR-130, 1974.
- Shrobe, H.E., Waters, R.C. and G.J. Sussman "A hypothetical monologue illustrating the knowledge underlying program analysis." MIT, AI-Memo 507, 1979.
- Teitelman, W. "Interlisp reference manual." Xerox PARC, Palo Alto, 1974.
- Wertz, H. "Un systeme de comprehension, d'amelioration et de correction de programmes incorrects." These de 3eme cycle, Univ. Paris 6, 1978.

## THE MARKGRAF KARL REFUTATION PROCEDURE

N. Eisinger, J. Siekmann, G. Smolka  
 E. Unvericht, Chr. Walther  
 Institut für Informatik I  
 Universität Karlsruhe  
 D-7500 Karlsruhe 1  
 Postfach 6380  
 West Germany

*Abstract: The current state of a Theorem Proving System (The Markgraf Karl Refutation Procedure) at the University of Karlsruhe is presented. The goal of this project can be summarized by the following three claims: it is possible to program a theorem prover (TP) and augment it by appropriate heuristics and domain-specific knowledge such that*

- (i) it will display an 'active' and directed behaviour in its striving for a proof, rather than the 'passive' combinatorial search through very large search spaces, which was the characteristic behaviour of the TPs of the past. Consequently*
- (ii) it will not generate a search space of many thousands of irrelevant clauses, but will find a proof with comparatively few redundant derivation steps.*
- (iii) Such a TP will establish an unprecedented leap in performance over previous TPs expressed in terms of the difficulty of the theorems it can prove.*

*The results obtained thus far corroborate the first two claims.*

## 0. INTRODUCTION

The working hypothesis of this TP project [DS77, DS79], first formulated in an early proposal in 1975, reflects the then dominating themes of artificial intelligence research, namely that TPs have attained a certain level of performance, which will not be significantly improved by:

- (i) developing more and more intricate refinement strategies (like unit preference, linear resolution, TOSS, MTOSS, ...), whose sole purpose is to reduce the search space, nor by
- (ii) using different logics (like natural deduction logics, sequence logics, matrix reduction methods etc)

although this was the main focus of theorem proving research in the past.

The relative weakness of current TP-systems as compared to human performance is due to a large extent to their lack of the rich mathematical and extramathematical knowledge that human mathematicians have: in particular, knowledge about the subject and knowledge of how to find proofs in that subject.

Hence the object of this project is to make this knowledge explicit for the case of *automata theory*, to find appropriate representations for this knowledge and to find ways of using it. As a testcase and for the final evaluation of the project's success or failure, the theorems of a standard mathematical textbook [DE71] shall be proved mechanically.

In the first section of this paper we give a general overview of the system as it is designed, albeit not completed. The second section concentrates on those parts of the system, whose implementation is finished and evaluated. In the third section experimental results are given and the final two sections present an evaluation based on the present findings.

## 1. OVERVIEW OF THE SYSTEM

### *A Bird's-eye View*

Proving a theorem has two distinct aspects: the creative aspect of how to find a proof, usually regarded as a problem of *psychology*, and secondly the logical aspect as to what constitutes a proof and how to write it down on a sheet of paper, usually referred to as *proof theory*.

These two aspects are in practice not as totally separated as this statement suggests (see eg. [SZ69]), however we found it sufficiently important to let it dominate the overall design of the system:

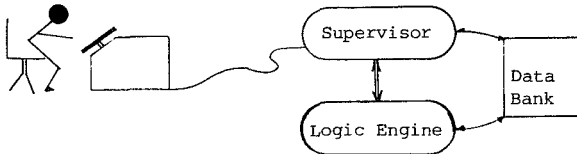


Figure 1

The *Supervisor* consists of several independent modules and has the complex task of generating an overall proposal (or several such) as to how the given theorem may best be proved, invoking the necessary knowledge that may be helpful in the course of the search for a proof and finally transforming both proposal and knowledge into technical information sufficient to guide the Logic Engine through the search space.

The *Logic Engine* is at heart a traditional theorem prover based on Kowalski's connection graph proof procedure [K075], augmented by several components that account for its strength.

The *Data Bank* consists of the factual knowledge of the particular mathematical field under investigation, i.e. the definitions, axioms, previously proved theorems and lemmata, augmented as far as possible by local knowledge about their potential use.

### *A View from a Lesser Altitude*

The diagram of figure 2 sufficiently refines figure 1 to gain a feeling for the overall working of the system:

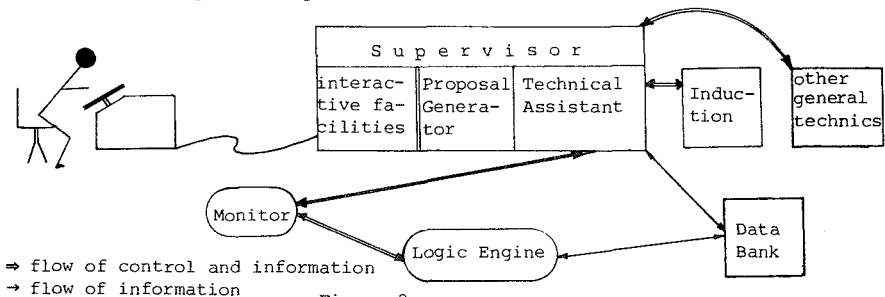


Figure 2

The user types in his problem, gives any possible additional information and advice and then specifies some of the many options open for the final protocol of this session. On the basis of this information the *Proposal Generator* (PG), which is part of the supervisor, may decide that the theorem is best proved by a general technique like induction, in which case the Supervisor loads the induction modules [CW79] and hands over control. Another instance for the activities of the PG is the following: in the case of the presence of equality literals, it discovers this fact and causes the set of clauses to be transformed into a more appropriate set and hands over control to the *Paramodulation Module*. The transformation involves the deletion of certain equality literals (clauses), which are selected for the rewrite-rule-module [HU77], [LA77], [DS79]; other axioms are deleted since they are taken care of by the unification module [DS77], whereas certain other axioms are added (e.g. the functional reflexive axioms) to ensure that the resulting set is E-unsatisfiable iff the previous set was unsatisfiable.

If none of the general techniques apply, the Proposal Generator generates a *proposal* (or several such), which contains heuristic information, of how the theorem may best be proved (e.g. it contains the definitions and lemmata which may be relevant, it suggests some proof techniques which are likely to be appropriate, it suggests which assertion or subproblem is best proved first, it tries to generate a top level plan for the proof if possible and finally it decides which of the permanent heuristics are to be activated).

The proposals are given one at a time to the *Technical Assistant* (TA), whose task is to transform this information, which up to this point is intelligible for a human user, into technical advice and code, which will then govern the top level behaviour of the Logic Engine and is for that reason passed on to the Monitor.

The *Monitor* governs and controls the global behaviour of the Logic Engine: immediately after activation it checks for an easy proof using the terminator heuristic (see section 2) and only upon failure activates the full machinery of the Logic Engine. Typical *control* tasks are detecting constant reapplications of the same lemma, detecting a circular development in the search space and keeping track of the 'self resolving' clauses. A good example how the monitor *governs* the top level behaviour is in its prevention of the unsteady behaviour which the system showed during earlier experimentation: The selection heuristics constantly suggest 'interesting' steps to take and forced the system to vacillate between different parts of the search space - very unlike the behaviour of people, who, if put into the same situation, would tackle an interesting path until they either succeed or become somehow convinced that it was a blind alley.

Up to this point the decisions and activities of the PG and the TA are to a large extent based on the semantics of the theory under investigation and knowledge about proofs in this theory and their top-goal may be formulated as: to be helpful "*in finding a proof*". Once they have done so, the top-goal becomes "*to derive a contradiction (the empty clause)*" and although this goal is of course identical to the previous one, it implies that different kinds of information may be useful: the original information provided by the PG based on the *semantics* (which is by now coded into various parameters, priority values and activation modules) as well as information based on the *syntax* (of the connection graph or the potential resolvent).

It may be objected that this is the main goal of a traditional TP also. While this is of course true, there is the important difference that a traditional (resolution based) TP is not directly guided towards this goal in a step by



step fashion, as no refinement [LO78] specifies which literal to resolve upon next. For example linear resolution reduces the search space as compared to binary resolution, but within the remaining space the search is as blind as ever.

The PG, the TA and the Monitor are currently under development and not implemented at the time of writing.

## 2. THE LOGIC ENGINE

The *Logic Engine* is based on Kowalski's connection graph proof procedure [KO75], which has several advantages over previous resolution based proof procedures: there is no unsuccessful search for potentially unifiable literals, every resolution step is done once at most and the deletion of links and subsequently of clauses leads to a remarkable improvement in performance, which is heavily exploited in the Deletion-Module. Most crucial to our approach however is the observation that since every link represents a potential resolvent, the *selection* of a proper sequence of links leads to the alleged active, goaldirected behaviour of the system.

### *Input, Output, Simplification and Evaluation*

The *interactive facilities* are too numerous to account for here and instead a protocol of a typical session is presented at the conference. An interesting point to note is that the interaction at this level was only designed for the intermediate stages of development. It is now to an increasing degree taken over by the Supervisor as it develops, with the intention to move the interface with the user altogether to the outside and to make the Supervisor take most of the low level decisions. Two sets of instructions however are to stay: the *IN-Module* is used to set up (and to read) the Data Bank in a way easily intelligible for the user. It also performs a *syntactical* and *semantical* analysis of the Data Bank, which is of considerable practical importance in view of the fact that it eventually contains a whole standard mathematical textbook.

The *PO-Module* provides several facilities for tracing the behaviour of the system at different degrees of abstraction in order to cope with its complexity.

Once the Logic Engine is set into "prove-mode", the *CSS-Module* converts the *activated* part of the Data Bank into Skolemized clausal normal form and performs various splitting and truth functional simplification tasks [BL71]. The resulting set of clauses is passed on to the *CG-Module*, which constructs the connection graph and if possible performs an evaluation of terms, reductions or algebraic simplifications of terms with the aid of the *ERS-Module*. After these activities the *initial connection graph* is set up and now the search for a proof within this graph commences.

This search is locally controlled by the *Control-Module*, which decides which link to resolve upon next, based on information provided by the Heuristic-Module, the Strategy-Module, the Deletion-Module and others. The Control-Module turns the *initial representation* of clauses (the connection graph) into a *proof procedure* (based on connection graphs) as it defines a particular selection function, which maps graphs to links. This mapping is complex and based on information provided by several modules, but for clarity it is entirely contained in the Control-Module.

We shall now digress from the description of the actual search and first present the heuristic selection functions contained in the *Heuristic-Module*.

### The Heuristic Selection Functions

In contrast to the global search strategies and global heuristics, the heuristic selection functions of the *Heuristic-Module* are based on *local syntactical information* about the graph or the resolvent (paramodulant) respectively [SS80]. These heuristics were obtained with two types of experiments: In the first experiment a human testperson is asked to prove a given set of formulas by resolution *without any* information on the intended meaning of the predicate or function symbols. Then the same set is proved by the system and in case its performance is inferior, the analysis of the deviation (and introspection of the testperson on why a particular step was taken) can give valuable hints for further heuristics.

In the second type of experiment the system is set to prove a theorem. In the case of success, an analysis of the protocol, where in the listing of all steps the steps necessary for the proof are marked, provides a remarkably good source of ideas for improvement, particularly if the reason why a certain step was chosen, is also printed in the protocol listing. During the last two years several hundreds of such listings were analyzed.

Initially we experimented with about twenty different *heuristic features*, where each feature attaches a certain value to every link  $k$  in  $G_i$ .  $G_i$  is the present graph,  $G_{i+1}$  is the resulting graph after resolution upon link  $k$  and  $Res$  is the resolvent resulting from this step:

- (i) Sum of literals in  $G_{i+1}$
- (ii) Sum of clauses in  $G_{i+1}$
- (iii) Sum of links in  $G_{i+1}$
- (iv) Average length of clauses in  $G_{i+1}$
- (v) Average sum of links on literals in  $G_{i+1}$
- (vi) Sum (resp. average sum) of constant symbols in  $G_{i+1}$
- (vii) Number of distinct predicate symbols in  $G_{i+1}$
- (viii) Number of distinct variables in  $G_{i+1}$
- (ix) Sum of literals of  $Res$
- (x) Sum of links of  $Res$
- (xi) Sum of constant symbols in  $Res$
- (xii) Sum of distinct variables in  $Res$
- (xiii) Number of distinct predicate symbols in  $Res$
- (xiv) Term complexity of  $Res$
- (xv) Minimum of links on literals in  $Res$
- (xvi) Complexity of the most general unifier  $\sigma_k$  attached to link  $k$
- (xvii) Age of  $Res$
- (xviii) Degree of isolation of  $Res$
- (xix) Degree of isolation of the parents of  $Res$ .

The problem is that although each heuristic feature has a certain worth, the cost of its computation can by far outweigh its potential contribution. Also it may not be independent of the other heuristic features; for example features (xi) and (xii) both measure the "degree of groundness of  $Res$ ", but in a different way. Similarly the values for  $Res$  and for  $G_{i+1}$  are not independent for certain features (e.g. xiii and vii). Also there are the two problems of finding an appropriate *metric* for each feature and to decide upon their *relative worth* in case of *conflict* with other features.

The information contained in the heuristic features is entered in different ways: certain facts (e.g. decreasing size of the graph) have *absolute priority* and override all other information (see also the merge feature of TT in [DA78]). Most of the information of the other features is expressed as a real number in  $[0,1]$ , where we experimented with several (linear, nonlinear) metrics. This in-

formation is then entered in a weighted polynomial and the resulting real number (the priority value) expresses the *relative worth* of the particular link and is attached to each link. In case of no overriding information the Control-Module selects the link with the highest priority value. Still other information is entered using the "window-technique": among the links whose priority value is within a certain interval (the 'window'), the Control-Module chooses the one which minimizes some other feature.

In addition to this information based on the syntax of the graph (or the resolvent), we are planning to attach semantic and pragmatic knowledge in form of small CSSA-programs [RA79] to each link.

The system has been designed such that heuristic features can easily be added and deleted, however after two years of experimentation the system has stabilized with the following solution (stabilized in the sense that neither the addition of heuristics from the above list nor the use of different metrics will significantly change the performance of the system on an appropriately large set of tests):

### 1. Complexity of the Graph

- 1.1 FCLSUM = ( $\Sigma$  of clauses of  $G_{i+1}$ ) - ( $\Sigma$  of clauses of  $G_i$ )
- 1.2 FLINKSUM = ( $\Sigma$  of links of  $G_{i+1}$ ) - ( $\Sigma$  of links of  $G_i$ )
- 1.3 FCANCEL = # {P|P is predicate symbol occurring in  $G_{i+1}$ }
- 1.4 FTERMINATE: (see below)

### 2. Complexity of the Resolvent

- 2.1 FAGE = Age of Res
- 2.2 FLITSUM = Sum of literals in Res
- 2.3 FTERM = Term complexity of Res
- 2.4 FRESISO = Degree of isolation of Res

### 3. Complexity of the Parents of Res

- 3.1 FPARISO = Degree of isolation of the parents

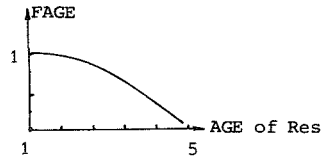
These features influence the actual derivation in the following way: all steps that lead to a reduction in the size of the graph have *absolute priority* and are immediately executed. That is, every link which leads to a graph with fewer clauses or fewer links or both is put into a special class, which is executed before any further evaluation takes place. The decision whether or not a link leads to a reduction is based on information from the Deletion-Module (see below) and is optionally taken for *every* link or for the *active* links only (see the Strategy-Module below). Note that the reduction in the size of the graph may lead to further deletions, hence a potential snowball effect of deletions is carried out *immediately*, which accounts for the first main source of the strength of the system.

All other features have a *relative priority* and are classified as situation dependent and situation independent respectively, since the cost of their initial computation and later updating differs fundamentally [SS80].

A successful usage of the relative priorities depends on an appropriate *metric* for each feature, which expresses its estimated worth. A discussion of why the following metrics where chosen is outside of the scope of this paper and may be found in [SS80]; but the important point to notice is, that each metric displays a particular *characteristic*, which expresses the heuristic worth relative to its argument.

$$\underline{\text{PAGE}} := 1 - \left( \frac{\text{Age}}{D_{\max}} \right)^{3/2}$$

Characteristic:  
(for  $D_{\max} = 5$ )



Age:  $\max\{\text{Age Parent1}, \text{Age Parent2}\} + 1$

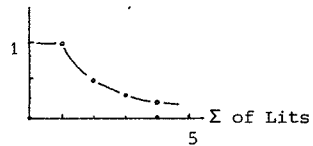
$D_{\max}$ : user defined maximally admitted depth of derivation

This feature is mainly used to avoid "infinite holes" in the search space.

$$\underline{\text{FLITSUM}} := \frac{1}{\sum \text{ of Literals of Res}}$$

FLITSUM

Characteristic:



This feature adds a strong "unit-preference-behaviour" to the system.

$$\underline{\text{FTERM}} := \begin{cases} 1 & \text{if no nested terms in Res or no terms in Res} \\ \frac{1}{2} \left( 1 - \frac{1}{\left( \frac{1}{n} \sum (s_i)^{3/2} \right)^{3/2}} \right) & \text{otherwise} \end{cases}$$

where:  $s_i$ : maximal nesting of  $i^{\text{th}}$  term in Res

$s_{\max}$ : user defined maximally admitted depth of nesting

$n$ : number of terms in Res

$$\underline{\text{FCANCEL}} := \text{LIT}(G_i) - \text{LIT}(G_{i+1})$$

where  $\text{LIT}(G_i) = \#\{P | P \text{ occurs as predicate symbol in } G_i\}$

Note that the value is either 0 or 1 and this information is useful for a simulation of Colmerauer's cancellation strategy [K075].

FLINKSUM: has either *absolute priority* if it decreases or else is entered into the selection process of the CO-Module with the window-technique

FRESISO: has either *absolute priority* if Res is pure, else

$$:= \begin{cases} 0 & \text{each literal of Res has at least 3 links} \\ 1/3 & \text{there is a literal in Res with 2 links} \\ 1 & \text{there is a literal in Res with 1 link} \end{cases}$$

FPARISO: similar to FRESISO

Note that FPARISO and FRESISO are useful, since they provide the Control-Module with the information that after one (or less useful, after two) further steps another deletion process starts, which potentially leads again to a snow-ball effect of deletions.

The main constraint for these heuristics is that after each resolution step the values of the heuristic features have to be updated and it is essential that the cost of this updating is much less than the cost of performing every possible resolution. For that reason not every value of the arguments for the

metrics is computed exactly but estimated by some heuristic estimation function.

And finally there is the problem that the heuristics have the very limited horizon of one step ahead but that the computation of a further  $n$ -level look ahead for  $n > 2$ , is so prohibitively expensive that it outweighs the advantage. For that reason we implemented in addition a different  $n$ -level look ahead technique, which checks at tolerable cost if there is a proof within a pre-defined complexity bound. This terminator heuristic *F*TERMINATE is akin to the no-loop-requirement of [SI76 p. 832] and the  $n$ -level-look-ahead heuristic proposed by [KO75 p. 593] and is the second main, source for the success of the current system. The essential idea is an elaboration of the following observation:

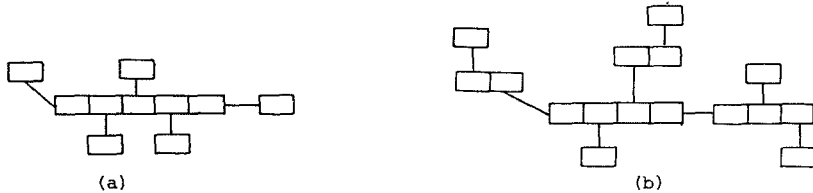


Figure 3

Each box in figure 3 represents a literal, a string of boxes is a clause and complementary boxes (literals) are connected by a link. If all unifiers attached to the links in figure 3a are compatible it represents a one-level-terminator, since it immediately allows for the derivation of the empty clause. Similarly figure 3b represents a two-level-terminator if the unifiers are compatible (see [SS80] for a detailed presentation).

### *Searching for a Proof*

We now return to a description of the search for a proof: The Control-Module locally controls the selection of the appropriate link for the next resolution step. This selection is based on information from the *Heuristic-Module* as presented above and from the *Deletion-Module*, which heavily exploits the crucial property of this proof procedure that distinguishes it from other proof procedures based on graphs [SH76], [SI76]: the fact that the deletion of links and clauses can lead to a snowball effect of further deletions. Because of this effect it is worth every effort to find and compute as many criteria for potential deletions as possible. At present a clause is marked for deletion if:

- (i) it is *pure*
- (ii) it is a *tautology*
- (iii) it is *subsumed* by some other clause [EI80].

Furthermore it is planned to have a deletion process based on models as well as a deletion process based on particular configurations of links (for example a circular connection), however because of the potential snowball effect the resulting completeness problems are not at all easy.

Mostly only a small fraction of all links in the graph is declared active. This is done by the *Strategy-Module*, which simulates standard derivation strategies: e.g. if only the links emanating from a clause in the initial set of support are declared 'active' and subsequently only the links of each resolvent in turn are declared active and the previous ones passive (by appropriate 'on-off-switches'), the resulting derivation is *linear*. The *Strategy-Module* allows to choose among some of the standard strategies and setting refinements [LO78]. This has turned out to be advantageous, since it substantially reduces the num-

ber of 'active' links and hence the expense for the computation of the heuristic selection functions and the most successful runs were obtained with this 'mixed approach'. The decision, which strategie to choose, is taken by the Monitor based on some general heuristics.

However certain features with relative priority as well as the features having absolute priority are optionally computed for both active and passive links. The passive links may be declared active, if their worth exceeds a certain threshold.

It should not be necessary to say that the complex interplay of the various modules, which 'suggest' which step to take next, prevents of course the overall deduction from being linear or 'standard' and the respective completeness results do not necessarily hold.

The potential explosion of links is the bottleneck of Kowalski's proof procedure: the following 'challenge' proposed by P. Andrews, Carnegie Mellon at the 1979 Deduction Workshop, provides a point of demonstration:

$$(\exists x Qx = \forall y Qy) = (\exists x \forall y Qx = Qy).$$

The initial graph of this formula consists of almost 10 000 links and several hundred new links are added to the graph for *each* resolution step. If all these links were declared 'active', the computation of the selection functions would become intolerably expensive.

In our system the example is split, reduced and subsequently deleted to a graph never exceeding 50 links and easily proved within a few steps. Even for more 'natural' examples, the number of deletion steps is about one third and sometimes over one half of the total number of steps.

#### *Some Technical Data about the Project*

*Name of Project:* Incorporation of Mathematical Knowledge into an ATP-System, investigated for the Case of Automata Theory.  
*Funding Agency:* German Research Organisation (DFG) Bonn, De 238/1, De 238/2.  
*Time Period:* 1976 to 1982 (six years)  
*Machine:* SIEMENS 7.760  
*Minimally Required Storage Space:* 6.000 K (virtual memory)  
*Languages:* SIEMENS-INTERLISP [EP75]/CSSA [Ra79]  
*Present size of system:* > 500 K of source code  
*Effort:* ≥ 10 manyears for its implementation

With more than 500 K of actual code at present and approximately 1.000 K under design for the next two years, the system is the largest software development undertaken in the history of automated theorem proving and it may be indicative for the changing pattern of research in this field.

### 3. PERFORMANCE STATISTICS

To gain a feeling for the improvement achieved by the system, figure 3 gives a sample of some test runs. In order to avoid one of the pitfalls of statistical data, which is to show the improvement achieved on certain examples and not showing the deterioration on others, the system is to be tested on all of the main examples quoted in the ATP literature: [WM76], [MOW76], [RRYKU72]. Of all examples tested thus far, the examples of figure 3 are representative (and worstcase, i.e. all other examples were even more favourable for our system). The examples of figure 3 are taken from the extensive, comparative study undertaken at University of Maryland [WM76], where eight different proof procedures were tested and statistically evaluated on a total of 152 examples.

The table is to be understood as follows: the first column gives the name of the set of axioms in [WM76], e.g. LS-35 in line 9. The next three columns quote the findings of [WM76], where the figure in brackets gives the value for the worst proof procedure among the eight tested procedures and the other figure gives the value for the proof procedure that performed best. The final three columns give the corresponding values for the Markgraf Karl Procedure. For example, in order to prove the axiom set LS-35 (line 9) the best proof procedure of [WM76] had to generate 335 clauses in order to find the proof, which consisted of 14 clauses, and the worst proof procedure had to generate 1.521 clauses in order to find that proof. In contrast our system *generated only 9 clauses* and as these figures are typical and hold uniformly for *all* cases, they are the statistical expression and justification for the first two claims put forward in the abstract.

Example	Maryland Refutation Procedure			Markgraf Karl Refutation Procedure		
	NOC-P	NOC-G	G-P	NOC-P	NOC-G	G-P
Ances	19 (18)	62 (943)	0,306 (0,019)	7	7	1
Ew 33	19 (21)	63 (2585)	0,302 (0,008)	8	16	0,5
Prim	21 (20)	89 (221)	0,236 (0,09)	12	27	0,444
Wos 3	7 (7)	17 (154)	0,412 (0,045)	3	3	1
Wos 7	13 (12)	241 (244)	0,054 (0,049)	9	10	0,9
Wos 8	12 (12)	210 (360)	0,057 (0,033)	6	9	0,667
LS-17	20 (14)	98 (1273)	0,204 (0,011)	4	7	0,571
LS-21	12 (12)	252 (684)	0,048 (0,018)	7	12	0,583
LS-35	14 (14)	335 (1521)	0,042 (0,009)	8	9	0,889
LS-65	17 (17)	48 (880)	0,354 (0,019)	11	58	0,189
LS-115	13 (13)	20 (227)	0,65 (0,057)	7	11	0,636
LS-121	31	536	0,058	12	30	0,4

NOC-P = Number of Clauses in the Proof

NOC-G = Number of Clauses generated

G-P = G-Penetrance

$$G-P = \frac{NOC-P}{NOC-G}$$

Figure 4

#### 4. KINSHIP TO OTHER DEDUCTION SYSTEMS

The advent of PLANNER [HE72] marked an important point in the history of automatic theorem proving research [AH72], and although none of the techniques proposed there are actually present in our system it is none the less the product of the shift of the research paradigm, of which PLANNER was an early hallmark.

The work most influential, which more permeates our system than is possible to credit in detail, is that of W. Bledsoe, University of Texas [BT75],[BB75],[BB72],[BL71],[BL77]. In contrast to [BT75] however, we tried to separate as much as possible the logic within which the proofs are carried out from the heuristics which are helpful in finding the proof.

The strongest resolution based system at present is [MOW76], and we have tested their examples in our system. Comparison with their reported results, shows that if our system finds a proof it is superior to the same degree as reported in figure 4.

However, there are still several more difficult examples reported in [MOW76] which we can not prove at present. The strength of the system [MOW76] derives mainly from a successful technique to handle equality axioms and almost

all the examples quoted in [MOW76] rely on this technique. For that reason, as long as our paramodulation module is not fully equipped with proper heuristics there is no fair comparison (the test cases were obtained with the full set of equality axioms and no special treatment for the equality predicate).

Finally among the very large systems which presently dominate theorem proving research is the system developed by R. Boyer and J.S. Moore at SRI [BM78]. Their system relies on powerful induction techniques and although some of the easier examples quoted in [BM78] could be proved by our system at present, a justifiable comparison is only possible once our induction modules are completed.

A theorem prover based on heuristic evaluation was also reported by Slagle and Farrell [SF71] however it appears that such heuristics are not too successful for an ordinary resolution based prover.

## 5. CONCLUSION

At present the system performs substantially better than most other automatic theorem proving systems, however on certain classes of examples (induction, equality) the comparison is unfavourable for our system (section 4). But there is little doubt that these shortcomings reflect the present state of development; once the other modules (T-unification, paramodulated connection graphs, a far more refined monitoring, induction, improved heuristics etc) are operational, traditional theorem provers will no longer be competitive.

This statement is less comforting than it appears: the comparison is based on measures of the search space and it *totally neglects* the (enormous) resources needed in order to achieve the behaviour described. Within this frame of reference it would be easy to design the "perfect" proof procedure: the supervisor and the look-ahead heuristics would find the proof and then guide the system without any unnecessary steps through the search space.

Doubtlessly, the TP systems of the future will have to be evaluated in totally different terms, which take into account the *total* (time and space) resources needed in order to find the proof of a given theorem.

But then, is the complex system A, which 'wastes' enormous resources even on relatively easy theorems but is capable of proving difficult theorems, worse than the smart system B, which efficiently strives for a proof but is unable to contemplate anything above the current average of the TP community? But the fact that system A proves a theorem of which system B is incapable is no measure of performance either, unless there is an objective measure of 'difficulty' (system A may e.g. be tuned to that particular example). If now the difficulty of a theorem is expressed in terms of the resources needed in order to prove it the circulus virtuosus is closed and it becomes apparent that the 'objective' comparison of systems will be marred by the same kind of problems that have marked the attempts to "objectify" and "quantify" human intelligence: they measure certain aspects but ignore others.

In summary, although there are good fundamental arguments supporting the hypothesis that the future of TP research is with the finely knowledge engineered systems as proposed here, there is at present no evidence that a traditional TP with its capacity to quickly generate many ten thousands of clauses is not just as capable. The situation is reminiscent of today's chess playing programs, where the programs based on intellectually more interesting principles are outperformed by the brute force systems relying on advances in hardware technology.



*Acknowledgement:* The presentation of this paper was substantially changed in view of the constructive criticism of the referees, which we gratefully acknowledge. In particular we would like to thank J.L. Darlington for a careful reading of an earlier manuscript and many detailed suggestions.

We would also like to express our thanks to the faculty of computer science and in particular to the head of our department, Prof. Deussen, for the very generous support and the resources provided, without which an undertaking of this kind would have been impossible. Also we would like to acknowledge the financial help from the Deutsche Forschungsgemeinschaft and the software support received from the INTERLISP-group with SIEMENS, München.

## REFERENCES

- [AH72] B. Anderson, P. Hayes, An Arraignment of Theorem Proving or the Logicians Folly, Comp. Logic memo 54, University of Edin
- [BB75] M. Ballantyne, W. Bledsoe, Autom. Proofs and Theorems in Analysis using nonstandard Techniques, ATP-23, 1975, University of Texas,
- [BBH72] W. Bledsoe, B. Boyer, Hemmeman, Computer Proofs of Limit Theorems, J. Art. Intellig. vol 3, 1972
- [BL71] W. Bledsoe, Splitting and Reduction Heuristics in ATP, J. Art. Int. vol 2, 1971
- [BL77] W. Bledsoe, A maximal method for set variables in ATP, ATP-33, University of Texas, Austin, 1977
- [BM78] R.S. Boyer, J.S. Moore, A Computational Logic, SRI International, Comp. Sci. Lab., 1978
- [BT75] W. Bledsoe, M. Tyson, The UT Interactive Prover, University of Texas, Austin, ATP-7, 1975
- [DA78] J.L. Darlington, Connection Graphs and Fact Nets, Internal Report, GMD Bonn, I.S.T., 1978
- [DA78] J.L. Darlington, Connected Fact Nets and Automatic Programming, GMD Bonn, I.S.T., 1978
- [DA79] J.L. Darlington, A Net Based Theorem Proving Procedure for Program Verification and Synthesis, 4th Workshop on Art. Intelligence, Bad Honnef, 1979
- [DE71] P. Deussen, Halbgruppen und Automaten, Springer 1971
- [DS77] P. Deussen, J. Siekmann, Neuantrag zum Forschungsvorhaben 'Untersuchung zur Einbeziehung mathematischen Wissens beim Automatischen Beweisen am Beispiel der Automatentheorie', DFG-Forschungsprojekt, Aktenzeichen De 238/1, 1977
- [DS79] P. Deussen, J. Siekmann, Zusatzantrag zum Forschungsvorhaben 'Untersuchung zur Einbeziehung mathematischen Wissens beim Autom. Beweisen am Beispiel der Automatentheorie', DFG-Projekt De 238/1, 1979
- [EI80] N. Eisinger, Subsumed Connection Graphs, Dipl.-thesis, Universität Karlsruhe, 1980 (in German)
- [EP75] B. Epp, INTERLISP Programmierhandbuch, Institut für Deutsche Sprache, Mannheim
- [HE72] C. Hewitt, Description and Theoretical Analysis of PLANNER, AI-TR-258, MIT
- [HU77] G. Huet, Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, 18th IEEE Symp. on Found. of Comp. Sci., 1977
- [KO75] R. Kowalski, A Proof Procedure based on Connection Graphs, JACM, vol 22, no 4, 1975
- [LA77] D.S. Lankford, A.M. Ballantyne, Complete Sets of Permutative Reductions, ATP-35, ATP-37, ATP-39, University of Texas at Austin, 1977
- [LO78] D. Loveland, Automated Theorem Proving, North Holland, 1978

- [MOW76] J. McCharen, R. Overbeck, L. Wos, Problems and Experiments for and with Automated Theorem Proving Programs, IEEE Transac. on Computers, vol-C-25, no 6, 1976
- [RA79] G. Fischer, P. Raulefs, CSSA Primer, Universität Bonn, Programming manual
- [RRYKV72] Reboh, Raphael, Yates, Kling, Velarde, Study of Automatic Theorem Proving Programs, 1972, Stanford Research Institute, SRI-TN-75, Stanford
- [SF71] J.R. Slagle, C.D. Farrell, Experiments in Automatic Learning for a Multipurpose Heuristic Program, CACM vol 14, 1971
- [SG65] A.I. Schulze-Gauss, Resolution considered harmless, Acta Theoria vol 2, 1965
- [SH76] R.F. Shostak, Refutation Graphs, J. of Art. Intelligence, vol 7, no 1, 1976
- [SI76] S. Sickel, Interconnectivity Graphs, IEEE Trans. on Computers, vol C-25, no 6, 1976
- [SI79] J. Siekmann, Forschungsvorhaben zur Behandlung des Gleichheitsprädikates beim Automatischen Beweisen, Universität Karlsruhe, 1979
- [SS80] J. Siekmann, G. Smolka; Selection Heuristics, Deletion Strategies and N-Level-Terminator Situations in Connection Graphs, Universität Karlsruhe, SEKI-Report, 1980
- [SW79] J. Siekmann, G. Wrightson, Paramodulated Connection Graphs, Acta Informatica, 13, 1980
- [SZ69] P. Szabó, The collected papers of G. Gentzen, North Holland, 1969
- [WA79] Ch. Walther, Forschungsvorhaben zur Automatisierung von Induktionsbeweisen, Universität Karlsruhe, 1979
- [WM76] G. Wilson, J. Minker, Resolution, Refinements and Search Strategies; A Comparative Study, IEEE Transac. on Comp., vol C-25, no 6, 1976

DECISION RULES FOR BINARY  
CHOICES: A PROCESS TRACING STUDY

Eduard J. Fidler  
Faculty of Commerce and Business Administration  
The University of British Columbia  
Vancouver, B.C., CANADA V6T 1Y8

ABSTRACT

This paper proposes a descriptive choice model for decisions involving two alternatives and gives experimental results for twenty-eight decision makers. For each decision maker a computer simulation model, based on verbal accounts of subjects' thought processes, was developed. An analysis of these models indicates substantial support for the proposed model. Additionally, the computer models were found to predict correctly more than 99% of all reliable decisions. An analysis of the incorrect predictions indicates that both the selection of decision rules, as well as their parameters are partly governed by a probabilistic process.

During the past several years, the understanding of the thought process underlying decision making behavior has received more and more attention in the literature. Instead of only focusing on an analysis of the relationships between decision attributes and choice outcomes by means of regression analysis [cf. Slovic and Lichtenstein, 1971 for an extensive review of that literature], students of choice behavior are increasingly investigating predecisional behavior, like the acquisition, evaluation, and integration of information [e.g., Svenson, 1979]. Unfortunately, relatively little research is available that has examined in detail the cognitive processes of decision makers. Furthermore, few attempts have been made to integrate empirical findings into a theoretical framework. Hence, the major goal of this study is to examine in detail the choice strategies of decision makers, and to develop a theoretical model for binary choices. More specifically, the plan of this study is to provide first a short summary of recent work on decision-making processes. Then, a binary choice model is presented and the results of two experiments are described and discussed in the light of the findings of other work on choice behavior.

BINARY CHOICE RESEARCH

There have been several decision making studies which have examined binary choices [e.g., Payne, 1976; Russo and Doshier, 1975; and Slovic, 1975]. Among those studies, Russo and Doshier [1975] provide the probably most detailed treatment of decision making strategies for choices among two alternatives. As a result of the analysis of eye-movements and verbal reports, Russo and Doshier identified two major simplifying heuristics that characterized subjects' choice behavior. These decision rules, which were established for tasks involving three decision attributes, were called dimensional reduction (DR) and majority of confirming dimensions (MCD) heuristic.

As the name suggests, the major characteristic of the DR heuristic is the reduction of the number of dimensions being considered during the choice process. More specifically, Russo and Doshier suggest that at the outset

subjects eliminate the dimension with the smallest dimensional difference. Thereafter, the choice will be determined by selecting the alternative which has the largest dimensional difference on the two remaining dimensions. Obviously, this choice model is most appropriate when the number of attributes is three. For a larger number of attributes, say six, four (i.e., six minus two) would have to be eliminated in order to allow the choice to be determined by ordinal comparisons of two dimensions. However, the elimination of two-thirds of the available information seems to be an oversimplification of subjects' decision rules. Therefore, the generality of this heuristic appears to be restricted to tasks with a small number of dimensions (e.g., three or four).

The second choice strategy, the MCD heuristic, simplifies the choice process by (a) ignoring the magnitude of dimensional differences and (b) considering only which of the two alternatives is better on each dimension. Then these simple evaluations of the decision attributes are integrated by selecting the alternative which is perceived to be better on more dimensions. Like the DR heuristic, the MCD strategy seems to have some major limitations. First, the implicit assumption that all dimensions are having the same impact on the choice outcomes (by being equally important) for all subjects appears somewhat unrealistic, judging from the results of several studies [e.g., Slovic and Lichtenstein, 1971] which suggest substantial individual differences in cue utilizations. The second major drawback of the MCD model is that it does not allow the prediction of a choice when an equal number of dimensions favor each alternative. Slovic [1975] partially addressed the latter issue, by examining binary choices among alternatives characterized by two decision attributes. His conclusions were that subjects tend to select the alternative which is better on the more important dimension. This short discussion shows that, for becoming a more realistic description of the choice process, several modifications of Russo and Doshier's [1975] original model seem to be necessary. As a first step in this direction a decision-process model, called "generalized attribute dominance (GAD) model" will be presented below.

#### A GENERALIZED ATTRIBUTE DOMINANCE MODEL

The present model, which provides an integration and extension of the choice models proposed by Russo and Doshier [1975] and Slovic [1975], consists of three stages (see also Fig. 1).

##### 1. Determination of the Decision Relevant Attributes ( $A_R$ )

It is presumed that at the beginning of the choice process the decision maker (DM) will divide the choice dimensions into important and unimportant ones. As a result of this classification subjects are assumed to ignore the unimportant attributes during the remainder of the choice process.

##### 2. Dimensional Evaluation of Attributes

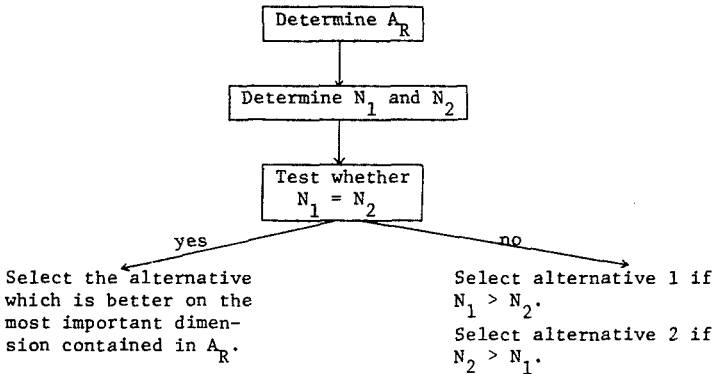
Having determined the decision relevant attributes, the DM will identify for each important attribute which alternative is better or whether both alternatives are equivalent. The characterization of two alternatives as being equivalent on a particular attribute also eliminates this attribute from the decision relevant attributes. Next, subjects will integrate the results of the dimensional evaluations by calculating for each alternative the number of dimensions (N) on which the respective alternative is better.

In particular,  $N_1$  and  $N_2$  denote the number of dimensions on which alternatives one and two are superior.

### 3. Comparison of $N_1$ and $N_2$

The final phase of the choice model involves the selection of the preferred alternative by comparing  $N_1$  and  $N_2$ . This comparison process can be considered as compensatory (i.e., involving trade-offs between attributes) in nature. Depending on the results of these comparisons, i.e., whether or not one alternative is perceived to be better on the majority of the decision relevant attributes, two decision mechanisms are proposed. In situations where  $N_1$  and  $N_2$  have different values the model predicts that the decision maker selects the alternative with the higher  $N$ . On the other hand, when both alternatives are identified as being superior on the same number of dimensions (i.e.,  $N_1 = N_2$ ), a tie-breaker rule has to be activated. According to this rule, the DM is expected to select the alternative which is better on the most important attribute contained in  $A_R$ .

Figure 1  
Generalized Attribute Dominance Model



The above model generalizes Russo and Doshier's [1975] MCD model in several important ways. First, it allows for differential weighting of dimensions across subjects. Second, it permits the evaluation of alternatives to be equivalent on any dimension. Finally, the model is capable of generating a choice prediction when the MCD-heuristic indicates a stand-off between the two alternatives. The tie-breaker rule "select the alternative which is better on the most important dimension", as suggested by the GAD-model, is very similar to a lexicographic choice model [cf. Tversky, 1972] and the choice procedure proposed by Slovic [1975]. Specifically, the lexicographic model presumes an ordering of the decision attributes according to their salience and prescribes the choice of the alternative which is superior on the most important dimension. If no alternative is perceived to be better on this dimension, the alternative which is superior on the second most important attribute is selected. This procedure is repeated until one alternative is identified as superior on a particular dimension. Since the dimensions on which both alternatives are perceived to be

equivalent are already excluded from the decision relevant attributes, no iterative choice procedure is required for the GAD model.

In order to obtain empirical data which permit direct testing of the GAD-model, it appears to be necessary to obtain access to subjects' thought process. Data on only the choice outcomes are unlikely to be capable of answering most of the above research questions. Therefore, it was decided to employ in this study verbal accounts of the thought process as the major source of data to be analyzed.

## METHOD

### Subjects and Task

Twenty-eight business seniors volunteered to participate in two very similar decision making experiments in response to several class announcements. In both experiments subjects were presented with information about pairs of students that were admitted into a quantitatively oriented graduate business program. The task was to decide which of the two students would be expected to achieve a higher grade point average (GPA) in that graduate program. The cues being presented for each of the hypothetical graduate students included some or all of the following pieces of information: (1) GMAT-Q: The percentile score on a quantitative aptitude test; (2) GMAT-V: The percentile score on a verbal aptitude test, (3) Undergraduate Institution: The name of the undergraduate school, and a rating of the overall quality of undergraduate education provided in that school; (4) GPA-Total: The GPA the student has achieved during his undergraduate education on a scale from 1 to 4; (5) GPA-Quantitative: The GPA for mathematics and statistics courses the student has achieved during his undergraduate education on a scale from 1 to 4; (6) Undergraduate Major: The major field of study during undergraduate education, and a rating of the major in terms of suitability for entering a quantitatively oriented business school.

Hypothetical decisions were generated based on the knowledge of the characteristics of applicants being admitted to a particular business school. Each decision was listed on a different page in the same form as the decision shown in Table 1.

Table 1  
Example of a Decision Problem

	<u>GMAT Verbal</u>	<u>Under- graduate Institution</u>	<u>GPA Total</u>	<u>GMAT Quant.</u>	<u>Under- graduate Major</u>	<u>GPA Quant.</u>
Alternative 1	68		3.2	86	Chemistry (3)	3.1
Alternative 2		Amherst (5)	2,8	74	Mathematics (5)	2.7

Since another purpose of the experiments was to examine choice behavior in the presence of missing information [cf, Fidler, 1979b], some pieces of information were frequently unknown (e.g., for the decision in Table 1 the school in alternative one and the GMAT-V in alternative two were not known).

### Experimental Procedure

In both experiments each subject was told that s/he would be participating in a study of human decision making. Thirteen subjects (six females and seven males) volunteered for experiment 1 and made their decisions in two parts which were one week apart. In the first part, 44 decisions were made (including six practice decisions), while in the second part 32 decisions - a subset of the part 1 decision - had to be made. The fifteen subjects (five females and ten males) participating in the second experiment had to make forty different decisions, plus six practice decisions at the beginning of the experiment. All decisions were made in the same session while thinking aloud.

Think aloud instructions were given before the first decision to be verbalized in both experiments. In these instructions subjects were told to verbalize every thought and every detail of their thinking process, including what information they were looking at, what thoughts they were having about any piece of information, how they were evaluating the different pieces of information, and the reasoning which led to their decisions.

### RESULTS

After transcribing each subject's verbalizations, an initial decision process model, consistent with the assumptions of the GAD-model, was derived for each subject. First, the decision relevant attributes were determined. Then, the conditions under which two scores of a particular dimension are considered to be equivalent or different were identified. Finally, the choice rules were formalized.

In the next phase of the model building process the inference and decision rules were translated into a FORTRAN computer program and the predictions of the computer model were compared with the actual choices. Then, the verbal reports for incorrect predictions were examined in detail, in order to detect potential misspecifications of the inference and decision rules. This detailed analysis of verbal reports for incorrectly predicted decisions frequently resulted in modifications of the decision rules and in improvements of the predictive quality of the process models.

### General Characteristics of the Decision Process Models

A characterization of the decision rules incorporated in the decision process models is shown in Table 2. These results substantially support the GAD-model proposed in this paper. Particularly, the choice behavior of more than 85% of the subjects was best represented by a decision process model that is largely consistent with the GAD-model. Five of these subjects also applied a configural decision rule in addition to the decision rules implied by the GAD-model. The configural decision rules did not involve trade-off relationships between the decision relevant attributes. Rather the mere existence of a particular configuration of the stimulus material on one or two attributes was sufficient for a choice response, regardless of the information available for the remaining attributes. Two particular configural decision rules that were identified are: "When the GMAT-Q of one alternative has a score of 90 or higher, while the GMAT-Q of the other alternative is (i) 85 or lower, or (ii) missing, the alternative with the GMAT-Q of 90 or higher will be chosen"; and "When one alternative has a better GMAT-Q and at the same time a one rated major, the alternative with the better GMAT-Q will be chosen".

Table 2  
Classification of the Decision-Process-Models

<u>Model Type</u>	<u>Number of Subjects</u>
Pure GAD-model	19
Configural decision rule plus GAD-model	5
Lexicographic choice model	3
Holistic choice model	<u>1</u>
	28

The GAD-models varied substantially across subjects. In particular, subjects differed in the relative importances they assigned to the decision attributes. In terms of the GAD-model parameters, individual differences were found to exist for (1) the decision relevant attributes and (2) the salience rankorder of the decision attributes. As illustrations of these individual differences the choice models for two subjects are presented:

Subject 25: decision relevant attributes: school, GPA-T, and GMAT-Q;  
 attributes in order of decreasing importance: school,  
 GMAT-Q, GPA-T.

Subject 2 : decision relevant attributes: all dimensions;  
 attributes in order of decreasing importance: GPA-T, GPA-Q,  
 major, GMAT-Q, GMAT-V, school.

Only two pairs of subjects had the same set of decision relevant attributes with an identical salience rankorder. Thus, even though the underlying cognitive choice processes seem to be very similar, the parameters of the choice models differ substantially between decision makers.

The process models of four subjects do not correspond to the decision strategies outlined for the GAD-model. Three of these subjects were best modeled by a lexicographic and one subject by a holistic choice model. A holistic decision rule evaluates each alternative as a whole and compares the outcomes of the holistic evaluations (e.g., utilities) for determining the preferred alternative. The particular holistic evaluation strategy applied by Subject 17 involved (1) the multiplication of each alternative's school, major, and GPA-T for determining each alternative's "utility"; and (2) the choice of the alternative with the highest utility score. Lexicographic decision rules do not permit any trade-offs among decision attributes and, therefore, are inconsistent with the major characteristic of the GAD-model, i.e., a primarily compensatory nature of the choice process. The lexicographic choice models varied substantially between subjects. Specifically, the sequence of the attributes in decreasing order of importance was as follows:

Subject 13: major, school, GMAT-Q.

Subject 18: GPA-T, GPA-Q, GMAT-Q, school.

Subject 20: GPA-T, GPA-Q, coin toss.

Subject 20, in addition, was the only DM who, when in doubt about which alternative to choose, determined the "preferred" alternative from the outcome of a coin toss.



All process models, except that for Subject 17, can be characterized as comparative in nature (as opposed to holistic) because they derive their predictions from comparisons of individual dimensions. This representation of subjects' choice behavior is also supported by an analysis of the choice reasons expressed in the verbal reports. Specifically, 88.4% of the choice reasons expressed for common dimensions were comparative evaluations of attributes. The remaining evaluations were non-comparative in nature and characterized individual attribute information as being good, high, average, low, etc.

#### Consistency of Decision Rules

In order to explore the consistency of the choice outcomes, all decisions that were made four or two times during the course of Experiment 1 were analyzed. For each subject the number of inconsistent decisions was determined as follows: for decisions made twice the number of inconsistencies was 0 if the same alternative was preferred both times and 1 otherwise; for decisions made four times the number of inconsistencies was 0 if the same alternative was preferred each time, 1 if one choice outcome was inconsistent with the majority of choices, and 2 if both alternatives were preferred equally often. An analysis of the number of inconsistent decisions indicates that across the 13 subjects participating in that experiment, on the average, 8.2 out of 62 decisions were inconsistent. The number of inconsistent decisions ranged between 3 and 15.

The individual decision process models correctly predicted, on the average, 87.6% of the 70 decisions made by all subjects during the course of Experiment 1. This corresponds to more than 99% of the potentially predictable decisions, when considering that inconsistent decisions cannot be predicted. For Experiment 2, on the average, 90.7% of all 40 decisions were predicted correctly by the process models. Assuming that the consistency of the choices is very similar for the subjects in both experiments, this result suggests that also for Experiment 2 the process models provide excellent predictions. Hence, the decision process models seem to be an extremely good predictor of the choice outcomes.

An analysis of the verbal protocols provided some interesting insights into the processes that are responsible for inconsistencies in choice behavior. The findings of this analysis will be illustrated for two decision makers, Subject 2 and Subject 16. For Subject 2 nine of the 70 decisions were incorrectly predicted by the process model. This was exactly the number of inconsistent decisions. Four of the inconsistent decisions were caused by a switch to a configural decision rule of the form "if the GMAT-Q is 90 or higher for one alternative and below 86 or missing for the other alternative, choose the alternative with the GMAT-Q of 90 or higher"; for three decisions a different attribute than that predicted by the model was used for breaking a tie between the two alternatives; one inconsistency was caused by the application of an otherwise unused inference rule for missing information; and the cause of one inconsistency could not be determined because of the lack of a verbal report for that decision. For Subject 16, two predictions of the decision process model did not coincide with the actual choices. While for all except these two decisions that subject appeared to be using a GAD-model strategy with the school, major, and GPA-T as the decision relevant attributes, for the two inconsistent decisions he simply chose the alternative with the better school without considering the other two decision attributes.

In general two types of changes in decision rules were observed: (1) changes in the type of decision rule being applied, and (2) changes in the parameters of the decision rule. The above results for Subject 2 provide a good illustration of changes in the type of decision rules that were observed for several subjects. Subject 2 used occasionally the previously described configural decision rule when one of the two alternatives had a GMAT-Q of 90 or higher. In most decision situations, however, he applied a decision rule which also took into consideration information on the other attributes. Therefore, the decision process model does not contain a configural decision rule. Several other subjects also applied infrequently a configural rule which is not shown in the decision process models. Similarly, subjects, for which the decision process model indicates configural decision rules, are not consistently applying the same choice procedure. Thus, they occasionally ignore the specific cue configurations and determine their choices based on other decision strategies. Another example of changes in the decision rule is obtained from the subjects applying predominantly a lexicographic choice model. These subjects not only appear to be using a probabilistic version of a lexicographic decision rule (similar to Tversky's [1972] elimination by aspects model) but, in addition, use sometimes a compensatory choice strategy (i.e., one that considers trade-offs among attributes) instead of the lexicographic choice model. Besides applying different types of decision strategies for similar decisions, subjects also appear to change probabilistically the parameters of their decision rules. For example, most subjects do not apply the lexicographic decision rules in the same sequence. Furthermore, several decision makers occasionally changed the set of decision relevant attributes for their GAD-models. More specifically, subjects sometimes considered a marginally important decision attribute (e.g., GMAT-V), and other times not.

A detailed analysis of the inconsistent decision outcomes suggests that these irregularities of the choice rules do not seem to stem from any systematic influences, like the order of presentation of the decision alternatives, the verbalization of the decisions, learning effects, etc. [cf. Fidler, 1979]. Instead, the inconsistencies of the choice outcomes appear to be caused by random changes in the decision rules applied to identical choice problems.

#### CONCLUSION

The findings of the present study provide substantial support for the choice model proposed in this paper. However, the current results also show that human choice behavior cannot be explained completely by one single choice model. In particular, this research found four major choice mechanisms: (1) the GAD-model proposed in the present paper; (2) decision rules dependent on particular stimulus configurations; (3) the lexicographic choice model; and (4) holistic decision rules.

The present results also suggest a conceptual model which views decision making behavior as a two stage probabilistic process. In the first stage the decision maker selects a particular type of decision rule, while in the second stage the parameters of the decision rule are determined. This conceptualization of choice behavior as being partly governed by a probabilistic process can be considered as an extension of Tversky's [1972] elimination by aspects model. Specifically, the current model permits the

selection of different classes of decision rules in addition to that of the parameters of one particular choice model.

However, more research is needed which (1) examines the conditions under which subjects tend to apply a particular choice model and (2) investigates the factors influencing the selection of the choice parameters. For example, the parameters of the decision rules may differ across subjects because of experience differentials for the specific choice situations. Particularly, perceptions of attribute salience may be strongly affected by experience. Further research is also necessary for testing the model for different decision tasks, in order to explore more thoroughly the limits of the present findings.

#### REFERENCES

- Fidler, E. J. The reliability and validity of concurrent and retrospective verbal reports for studying decision making behavior. Unpublished manuscript, Faculty of Commerce, University of British Columbia, 1979.
- Fidler, E. J. Choice processes under conditions of missing information. Unpublished manuscript, Faculty of Commerce, University of British Columbia, 1979.
- Payne, J. W. Task complexity and contingent processing in decision making: An information search and protocol analysis. Organizational Behavior and Human Performance, 1976, 16, 366-387.
- Russo, J. E. and Doshier, B.A. Dimensional evaluation: A heuristic for binary choice. Unpublished manuscript, University of California, San Diego, 1975.
- Slovic, P. Choice between equally-valued alternatives. Journal of Experimental Psychology: Human Perception and Performance, 1975, 1, 280-287.
- Slovic, P., and Lichtenstein, S. Comparison of Bayesian and regression approaches to the study of information processing in judgment. Organizational Behavior and Human Performance, 1971, 6, 649-744.
- Svenson, O. Process descriptions of decision making. Organizational Behavior and Human Performance, 1979, 23, 86-112.
- Tversky, A. Elimination by aspects: A theory of choice. Psychological Review, 1972, 79, 281-299.

## PROGRAM UNDERSTANDING BY REDUCTION SETS

Patrick GREUSSAY

Departement d'Informatique  
 Université Paris VIII - Vincennes  
 75571 PARIS Cedex 12

&  
 C.N.R.S. LA 248 L.I.T.P.  
 2 place Jussieu  
 75005 Paris

## Abstract :

While building or understanding large LISP systems, many small auxiliary functions are often subject to errors or misunderstanding, in the case of very involved recursions. RAINBOW is a specialized program understanding system able to reduce automatically such sets of recursive functions to a form where the goal of these sets are clearly displayed. RAINBOW can display interactively the goal-forms into two sets of new external 2-dimensional notations: recursive and linear. Program understanding is obtained by the translation of the original set of LISP functions into the open recursive notation, then by elementary symbolic evaluation yielding closed linear forms of the original functions. Those linear forms are exactly the goals wanted. RAINBOW operates efficiently on a definite class of LISP functions, and uses an extendable set of reduction rules, which constitute the symbolic interpreter. RAINBOW can be used interactively if a user want to verify that a set of functions perform its intended goal, or can be incorporated easily as a specialized component of a larger program understanding system. This paper shows how RAINBOW operates on sets of recursive functions building combinatorial objects.

## KEY-WORDS:

automatic program understanding, program debugging, LISP, RAINBOW system, multiple representations, program transformation, symbolic interpretation.

## 1. INTRODUCTION

RAINBOW is a specialized interactive program understanding system. It asks from its user a set of recursive functions definitions, then extracts and displays graphically its *goal*, in terms of properties of lists viewed as sequences.

To display the goal of a definition set, we have introduced two classes of 2-dimensional external notations which are implemented within RAINBOW. The *open* notation is a compact notation for recursive programs or data structures, the *closed* notation expresses intrinsic properties of linear sequences in term of generic properties of their elements.

Program understanding is obtained by the translation of the original set of functions into the open notation, then by elementary symbolic evaluation yielding closed forms of the original functions. The closed forms are exactly the goals wanted.

Presently, RAINBOW can reason about classes of data-structures as lists considered as sequences, extensions to recursive LISP functions operating on other classes of data-structures are considered.

RAINBOW can be used either as a front-end of a LISP system, or as a specialized part for low-level understanding of sequences, in a larger program understanding system. An analogy with low-level machine vision is here in order: scene analysis has to rely upon intrinsic properties of pictures, as incidence, gradient, illumination or texture. We believe that a large program understanding system must also rely upon intrinsic properties of the data involved in the programs tentatively analyzed.

A programming apprentice system (RICH 1979, WERTZ 1979, SHROBE 1979), if used interactively, must have the capability of focusing in a visually understandable way, to lower levels of plans for simple modules, which are the most error-prone in very large systems. RAINBOW allows the user to check immediately, the goal of a set of functions definitions. The user does not have to provide any assertion to verify about his definitions set, because in a sense RAINBOW is precisely reducing this set to its assertion.

RAINBOW is an implemented system written in LISP (CHAILLOUX 1978) running on DEC KI-10, and PDP 11/40. The external notations can be visualized on any kind of display or hard-copy terminal.

## 2. OVERVIEW OF THE RAINBOW SYSTEM

### *The symbolic interpreter:*

The symbolic interpreter is essentially a production system having an initial fixed set of reduction rules for the handling of LISP sequences. When a user submits a new function definition to RAINBOW, the reduced closed linear form obtained as a goal is incorporated by the system into the set of rules. The interpreter is able to expand every inner function call with the replacement part of the corresponding rule along with the renaming of variables when necessary ( $\alpha$ -conversion). As in (BOYER 1977), The interpreter operates iteratively until no more rule can apply to the reduced form.

The definitions entered can be also called within the LISP interpreter, and every step of the reduction can be interactively reversed, providing an history of the reduction. Also at user-level, RAINBOW can handle symbolic function calls. The process of resolution into linear forms uses a set of reduction rules for the translation of recursive representations into linear representations. The reduction set of rules for a LISP function is expressed into two new classes of graphical notations for recursive programs and data, and for linear sequences.

### *Recursive and linear external notations:*

#### 1) linear

It is used to express generic results or *goals* of the analysed functions: it expresses characteristics of sequences.

This notation is a 2-dimensional specialization of the one used in (TEITELMAN 1967, GOOSSENS 1979).

$$\begin{array}{c} n \\ | \\ [i \ E_i] \\ | \\ 1 \end{array} =df \ (E_1 \ E_2 \ \dots \ E_n)$$

The letter *i* is an "index variable" ranging from lower to upper indices, here from 1 to *n*. When one of the indices is 0, the notation denotes the empty sequence. In the context of RAINBOW, the range of the index variable is the LISP expression immediately following it.

#### EXAMPLES:

$$\begin{array}{ccc} \begin{array}{c} n \\ | \\ (CAR \ [i \ E_i]) \\ | \\ 1 \end{array} \rightsquigarrow E_1 & & \begin{array}{c} n \\ | \\ (CDR \ [i \ E_i]) \\ | \\ 1 \end{array} \rightsquigarrow \begin{array}{c} n \\ | \\ [i \ E_i] \\ | \\ 2 \end{array} \\ \begin{array}{c} n \\ | \\ (CONS \ E_1 \ [i \ E_i]) \\ | \\ 2 \end{array} \rightsquigarrow \begin{array}{c} n \\ | \\ [i \ E_i] \\ | \\ 1 \end{array} & & \begin{array}{c} n \\ | \\ (REVERSE \ [i \ E_i]) \\ | \\ 1 \end{array} \rightsquigarrow \begin{array}{c} 1 \\ | \\ [i \ E_i] \\ | \\ n \end{array} \end{array}$$

A more complex example is:

$$\begin{array}{c} n \quad l-1 \\ | \quad | \\ [i \ [j \ L_j] \ (CONS \ A \ L_l) \ k \ L_k] \\ | \quad | \\ 1 \quad 1 \end{array} \rightsquigarrow \begin{array}{c} n \\ | \\ [l+1 \end{array}$$

$$[[ (CONS \ A \ L_1) \ \dots \ L_n] \ [L_1 \ (CONS \ A \ L_2) \ \dots \ L_n] \ \dots \ [L_1 \ \dots \ (CONS \ A \ L_n)]]$$

## 2) recursive

It is used to express the recursive computation performed by the function. It has the general form:

$$\left[ \begin{array}{c} \text{---} \\ | \\ \alpha_i \text{ --- } \beta_i \text{ } \partial \\ | \\ \text{---} \end{array} \right]_1^n = \text{df} \quad \alpha_1 \dots \alpha_n \partial \beta_n \dots \beta_1$$

where  $\alpha_i$ ,  $\beta_i$ , and  $\partial$  are parts of LISP expressions as well as linear or recursive forms. The recursive notation is essentially an indexed context-free grammar rule. It expresses  $n$  levels of nesting of function calls terminating with the expression  $\partial$ . When the RAINBOW system uses the recursive notation,  $n$  is the length of the list which is the value of the recursion variable. The crossing of lines in the notation denotes a self-reference to the entire expression with the index variable progressing one step towards the highest index.

As an example RAINBOW translates interactively, the following function definition

```
(DE append (X Y)
  (IF (NULL X) Y
      (CONS (CAR X)
            (append (CDR X) Y))))
```

into the recursive form

$$\left[ \begin{array}{c} \text{---} \\ | \\ (\text{CONS } X_i \text{ ---}) Y \\ | \\ \text{---} \end{array} \right]_1^n \quad \text{which schematizes the nested expression} \quad (\text{CONS } X_1 (\text{CONS } X_2 \dots (\text{CONS } X_n Y) \dots ))$$

where  $X_i$  translates  $(\text{CAR } (\text{CDR}^{i-1} X))$

$$\left[ \begin{array}{c} \text{---} \\ | \\ X_j \\ | \\ \text{---} \end{array} \right]_1^n \text{ translates } X \text{ and } \left[ \begin{array}{c} \text{---} \\ | \\ X_j \\ | \\ \text{---} \end{array} \right]_{i+1}^n \text{ translates } (\text{CDR } X)$$

It happens that the goal of the append function is itself expressed into the reduction rule RC2:

$$\left[ \begin{array}{c} \text{---} \\ | \\ (\text{CONS } X_i \text{ ---}) \left[ \begin{array}{c} \text{---} \\ | \\ Y_j \\ | \\ \text{---} \end{array} \right] \\ | \\ \text{---} \end{array} \right]_1^n \quad \Rightarrow \quad \left[ \begin{array}{c} \text{---} \\ | \\ \left[ \begin{array}{c} \text{---} \\ | \\ X_i \\ | \\ \text{---} \end{array} \right] Y_j \\ | \\ \text{---} \end{array} \right]_1^n$$

## 3. USING RAINBOW: A SIMPLE REDUCTION

RAINBOW can be used interactively for the automatic documentation of programs as soon as they are typed, as advocated by (WINOGRAD 1979). In the following example, RAINBOW is reducing to its goal a set of two simple recursive functions. The user types the following definition, that RAINBOW translates immediately as:

```
(DE f (X Y)
  (IF (NULL X) NIL
      (APPEND (g (CAR X) Y)
              (f (CDR X) Y))))
```

$$\left[ \begin{array}{c} \text{---} \\ | \\ (\text{APPEND } (g X_i Y) \text{ ---}) \text{ NIL} \\ | \\ \text{---} \end{array} \right]_1^n$$

then the following definition

```
(DE g (A Y)
  (IF (NULL Y) NIL
      (CONS (LIST A (CAR Y))
            (g A (CDR Y)))))
```

is translated into

$$\left[ \begin{array}{c} \text{---} \\ | \\ (\text{CONS } [A Y_i] \text{ ---}) \text{ NIL} \\ | \\ \text{---} \end{array} \right]_1^n$$

then the user types to RAINBOW the symbolic call

```
(f (*list* A) (*list* B))
```

which is reduced into the linear form

$$\begin{array}{cc} n & m \\ | & | \\ [i & j] \\ | & | \\ 1 & 1 \end{array} [A_i \ B_j]]$$

which yields the goal of the function  $f$ :

*f* builds the **cartesian product** of its two argument-lists.

The reduction set of rules which has been used in the previous example is:

- the definition of the function  $g$  itself.

- the rule RC1

*and* - the rule RA1

$$\left[ \begin{array}{c} \text{CONS } E_i \\ \text{NIL} \end{array} \right] \rightsquigarrow [E_i]$$

So the RAINBOW system yields in succession :

```
? (f (*list* A) (*list* B))
```

Lines beginning with "?" are typed directly by the user

```

n-----m-----
| (APPEND (G Ai [ Bj]) -) NIL |
|-----|

```

OK

? ap all  
APPLYING... G GIVING...

```

( APPEND (CONS [Ai Bj] -) NIL -) NIL

```

APPLYING... RC1 GIVING...

```

      (APPEND [j [A, Bj]] -) NIL

```

APPLYING... RA1 GIVING...

$$\begin{bmatrix} n & m \\ [i] & [A, B] \end{bmatrix}$$

The final linear form

## 4. UNDERSTANDING COMPLEX RECURSIONS

Sometimes recursion can be quite involved, as in the following square matrix generator:

where

$$\begin{array}{l}
 \text{(DE vecmat (E))} \\
 \text{(IF (NULL E) NIL)} \\
 \text{(CONS E} \\
 \quad \text{(DCONS (CAR E)} \\
 \quad \quad \text{(vecmat (CDR E))))))
 \end{array}
 \quad
 \begin{array}{c}
 \text{where} \\
 \begin{array}{ccc}
 \begin{array}{c} n \\ \vdots \\ 1 \end{array} & \begin{array}{c} \text{(DCONS } \alpha \text{ } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{ E}_i \text{)} \\ \hline 1 \end{array} & \Rightarrow \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \text{(CONS } \alpha \text{ E}_i \text{)} \\ \hline 1 \end{array}
 \end{array}
 \end{array}$$

The goal of VECMAT extracted by RAINBOW is:

$$\begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \text{(VECMAT } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{ E}_i \text{)} \\ \hline 1 \end{array} \Rightarrow \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \text{(VECMAT } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{ E}_i \text{)} \\ \hline 1 \end{array} \\ \hline 1 \end{array}$$

The body of VECMAT is slightly generalized to obtain the main reduction rule:

$$\begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \text{(CONS } \alpha_i \text{ (DCONS E}_i \text{ } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{)) NIL} \\ \hline 1 \end{array} \Rightarrow \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \begin{array}{c} i-1 \\ \vdots \\ 1 \end{array} \begin{array}{c} \text{(CONS } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{ (CONS E}_j \text{ } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{)) } \alpha_i \text{ } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{ NIL} \\ \hline 1 \end{array}$$

which by the rule RC1 is reduced itself to:

$$\Rightarrow \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} i-1 \\ \vdots \\ 1 \end{array} \begin{array}{c} \text{(CONS E}_j \text{ } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{)) } \alpha_i \text{ } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{ (rule RDC4)} \\ \hline 1 \end{array}$$

Now if we sets  $\alpha_i$  to  $\begin{bmatrix} k \\ E_k \end{bmatrix}$ , RAINBOW obtains the linear form for VECMAT by:

$$\begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} i-1 \\ \vdots \\ 1 \end{array} \begin{array}{c} \text{(CONS E}_j \text{ } \begin{array}{c} n \\ \vdots \\ 1 \end{array} \text{)) } \begin{array}{c} k \\ E_k \end{array} \\ \hline 1 \end{array} \Rightarrow \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} i-1 \\ \vdots \\ 1 \end{array} \begin{array}{c} \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} E_j \end{array} \begin{array}{c} k \\ E_k \end{array} \\ \hline 1 \end{array} \Rightarrow \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} E_j \end{array} \\ \hline 1 \end{array}$$

The final result being obtained by the reduction rule RIND2:

$$\begin{array}{c} i-1 \\ \vdots \\ 1 \end{array} \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \begin{array}{c} X_i \end{array} \begin{array}{c} X_j \end{array} \\ \hline 1 \end{array} \Rightarrow \begin{array}{c} n \\ \vdots \\ 1 \end{array} \begin{array}{c} \begin{array}{c} X_i \end{array} \\ \hline 1 \end{array}$$

The same reduction process can be used to obtain the goal of the function *partinsert*, main component of a recursive partition generator. Though very usual this style of recursion is rather difficult to visualize, and is very error-prone. The following example displays a session with RAINBOW.



*The user types the definition*

```
? (DE partinsert (X E)
  (IF (NULL E) NIL
    (CONS (CONS (CONS X (CAR E)) (CDR E))
      (DCONS (CAR E) (partinsert X (CDR E))))))
```

*that RAINBOW translates into*

$$(\text{LAMBDA } (X E) \left[ \begin{array}{c} \text{CONS} \left[ \left( \text{CONS } X E_i \right) \right. \\ \left. \left[ \begin{array}{c} E_j \\ \vdots \\ E_{i+1} \end{array} \right] \right] \left( \text{DCONS } E_i \right) \dots \end{array} \right] \text{NIL}$$

*Then the user provides the symbolic call*

```
? cl (partinsert A (*list* L))
```

$$\left[ \begin{array}{c} \text{CONS} \left[ \left( \text{CONS } A L_i \right) \right. \\ \left. \left[ \begin{array}{c} L_j \\ \vdots \\ L_{i+1} \end{array} \right] \right] \left( \text{DCONS } L_i \right) \dots \end{array} \right] \text{NIL}$$

*Then the user asks for all reduction rules to be applied*

```
? ap all
  APPLYING... RDC4 GIVING...
```

$$\left[ \begin{array}{c} \text{CONS} \left[ \left( \text{CONS } L_j \right) \right. \\ \left. \left[ \begin{array}{c} \vdots \\ \vdots \end{array} \right] \right] \left[ \left( \text{CONS } A L_i \right) \right. \\ \left. \left[ \begin{array}{c} L_k \\ \vdots \\ L_{i+1} \end{array} \right] \right] \dots \end{array} \right] \text{NIL}$$

APPLYING... RC1 GIVING...

$$\left[ \begin{array}{c} \text{CONS } L_j \right. \\ \left. \left[ \begin{array}{c} \vdots \\ \vdots \end{array} \right] \right] \left[ \left( \text{CONS } A L_i \right) \right. \\ \left. \left[ \begin{array}{c} L_k \\ \vdots \\ L_{i+1} \end{array} \right] \right] \end{array}$$

APPLYING... RC2 GIVING...

$$\left[ \begin{array}{c} \left[ \begin{array}{c} L_j \\ \vdots \\ L_{i+1} \end{array} \right] \text{CONS } A L_i \right. \\ \left. \left[ \begin{array}{c} L_k \\ \vdots \\ L_{i+1} \end{array} \right] \right] \end{array}$$

OK

*The final reduced form is the goal of partinsert*

## 5. CONCLUSION

The 2-dimensional display for formula within RAINBOW is claimed to be intuitively understandable by using intrinsic properties of sequences: the use of indices relates the length and order of the sequence to the general form of the generic elements.

Along with the powerful reductions from recursive to linear forms, we believe that the external notations presented here reflect our intuitive understanding of lists and their properties.

In a fast checking situation, this graphical style of verification, displaying the generic structure of data, gives the user excellent control over the goal obtained from a functions sets.

The extendability provided by the incorporation of every new function definition as a new reduction rule in the spirit of (BOYER 1977) gives a useful tool to system design and programming methodology: as advocated by (GERHART 1975), the resulting schema and transformations that are saved will have to be ultimately organized into "handbooks of knowledge" about programming.

Presently, RAINBOW is able to reason about classes of data structures as sequences, we are considering its extension to arrays using the rules given in (REYNOLDS 1979). Program understanding with RAINBOW can be viewed as a kind of simplification, and its extension to several classes of data-structures may involve combination of decision procedures for several theories described in (NELSON 1978).

Internally, RAINBOW is mainly driven, out of the fixed initial set, by user-provided rules obtained by reduction of previous definitions to their goals. Thus, the power of RAINBOW is strictly limited by the class of expressions that the external notations are able to denote. Most of the rules in the fixed initial set are properties of the function CONS, extended to APPEND and REVERSE. In the present state of the rules, RAINBOW is restricted to primitive recursive functions. A single recursion variable is handled within each rule: an obvious extension to an arbitrary number of recursion variables can be incorporated, each having the same pattern of sequence as an argument. Another extension is currently planned to handle iterative schemes.

## 6. REFERENCES

- [1] BOYER R. & MOORE J. *A Lemma Driven Automatic Theorem Prover for Recursive Function Theory*, 5th I.J.C.A.I., August 1977, Cambridge, 511-519
- [2] CHAILLOUX J. *VLISP 10.3, Manuel de Reference* Universite Paris-8-Vincennes, August 1978
- [3] GERHART S. L. *Knowledge about Programs: A Model and Case Study*, Proc. International Conf. on Reliable Software, 21-21 April 1975, Los Angeles, 88-95
- [4] GOOSSENS D. *Meta-interpretation of Recursive List-processing Programs* I.J.C.A.I 1979, August 20-23, Tokyo, s7-s11
- [5] NELSON G. & OPPEN D. C. *Simplification by Cooperating Decision Procedures*, Stanford Artificial Intelligence Laboratory, Memo AIM-311, April 1978
- [6] REYNOLDS J. C. *Reasoning about Arrays*, Comm. A.C.M., May 1979, Vol 22, no 5, 290-299
- [7] RICH C. & SHROBE H. E. *Initial Report on a LISP Programmer's Apprentice* IEEE Transactions on Software Engineering, SE-4:6 (1978), 456-467
- [8] SHROBE H. E. & WATERS R. C. *A Hypothetical Monologue Illustrating the Knowledge Underlying Program Analysis*, M.I.T. Artificial Intelligence Laboratory, Memo 507, January 1979
- [9] TEITELMAN W. *Design and Implementation of FLIP, a LISP Format Directed List Processor*, AFCRL-67-0514, Bolt Beranek and Newman, July 1967
- [10] WERTZ H. *Automatic Program Debugging* I.J.C.A.I 1979, August 20-23, Tokyo, 951-953
- [11] WINOGRAD T., *Beyond Programming Languages* C.A.C.M., Vol. 22, no. 7, July 1979, 391-401

STABILITY AND CHANGE OF STRATEGIES:  
THREE SIMULATIONS OF ONE SUBJECT WITH ONE-YEAR INTERVALS

Göran Hagert

Department of Psychology, University of Stockholm,  
Box 6706, S-113 85 Stockholm, Sweden.

Abstract

The stability or change of problem solving strategies, without or with practice, is considered as an important aspect of human problem solving. It is proposed that a strategy is not situation dependent, but reflects stability in the problem solving skill of a subject. It is assumed that without practice in a task a strategy should recur in repeated solutions where the intervals between solutions are long. This hypothesis is investigated in this study. A production system, simulating in detail the behavior of one subject's solution to a spatial series task, is presented. The program is based on a think-aloud protocol produced by the subject while solving the task. The behavior of the program is compared to the subject's solution of the task one and two years later. These comparisons show that the subject uses mainly the same strategy or set of rules on the three occasions, even though the particular solutions on each occasion were rather different in terms of time to solution and answers given. Processing errors and rule modification are two factors that can explain the differences in behavior, rather than switches in the strategies the subject used. The study gives support to the hypothesis about stability of strategies.

Introduction

Problem solving strategies have been studied in different task domains from a developmental as well as a learning point of view. Both these research areas are mainly concerned with acquisitions, switches, or modifications of the rules constituting a strategy. The studies are not aimed toward testing hypotheses about the stability of the strategies they identify. The changes are the main target of the research.

A third, neglected, approach is considered in this paper. Namely, what aspects of strategies remain constant and are used again after a solution, or after repeated solutions, to a given problem. For instance, what would be expected if the subject of the Anzai and Simon study (1979) was asked to solve the Tower of Hanoi puzzle once again, a year later? Will she again begin with forward search, with a recursive sub-goal strategy or with something quite different? Will she use any of those rules she acquired through the repeated solution of the puzzle? What about S3 in the paradigmatic study by Newell and Simon (1972)? Are there any reasons to assume that he would solve DONALD + GERALD = ROBERT in a similar manner if he was to solve it now, in 1980, as he did the first time, several years ago?

It is assumed that a strategy acquired and used by a subject in one situation and on a given task must reflect something stable in his/her intellectual skill. As a working hypothesis it is proposed that strategies are stable over time, rather than fluctuating from situation to situation. In other words, the rules used to guide the search (Newell & Simon, 1972; Newell, 1979) in one problem solving situation should be used again in a later situation.

This hypothesis was investigated in a longitudinal case study. Briefly, one subject solved a spatial series task three times with one-year intervals. A production system, modeling the subject's strategy, was constructed on the basis of the first solution. The behavior of the program was compared to the other solutions, one and two years later. On the surface of the behavior there were dramatic differences. The computer model made it possible to analyze these differences and yielded one and the same set of rules generating the three solutions.

### Background of the Protocols

The type of task used in the study is called spatial arrangement tasks and is extensively discussed by Ohlsson (1980). An example of this task is given below.

A few persons are sitting in a sofa. Eva is sitting to the right of Carl. Ann is sitting leftmost. David is sitting immediately to the left of Eva. Bob is sitting between Ann and Carl. Who is sitting immediately to the left of David?

This text describes a spatial series of five persons: Ann, Bob, Carl, David, and Eva from left to right. The problem is to find a particular relation in this arrangement.

The subject was a male psychology student. He was asked to solve the above task three times with one-year intervals. He was also asked to think-aloud during the problem solving. The verbalizations were tape-recorded and transcribed into protocols. In the protocol, the verbalizations were segmented into fragments. The criteria for separating two fragments was the occurrence of a pause. The names in the task were different from occasion to occasion in order to avoid direct recognition.

### Variations in the Solution Processes

Table 1 presents time to solution, number of fragments, and answers given in each protocol. On the first occasion the subject gave first an incorrect answer (Bob) and then the correct one (Carl). He also gave an incorrect answer one year later, but a different one (Eva). Two years later, the solution did not contain any incorrect answer. As is seen, the protocols contain a varied number of fragments and the times to solution are also quite different.

Clearly, the solutions are not identical. That is, there is no stability in these traditional types of behavioral data, which means that the surface of the behavior varies from situation to situation. The simplest conclusion is that there is not a stable strategy behind this behavior, but completely different strategies. However, Table 1 does not contain any information about what strategy has been used. It only shows that the

particular overt behaviors are different. It is the purpose of the next sections to show that the pattern in Table 1 was generated by one and the same set of rules, and thus, to reject the above interpretation.

Table 1. Answers, fragments, and times found in the protocols showing the great variation in the behavior.

Occasion	Answer	Number of fragments	Time to solution
1	Bob,Carl	61	3:00
2	Eva,Carl	134	7:35
3	Carl	54	2:35

### The First Protocol

The protocol can be divided into four episodes separated by three backups. The entire protocol shows that the subject successively tried to build a series of the objects or an internal model of the problem situation. Thus, he used some variant of the Method of Series Formation (Ohlsson, 1980; Quinton & Fellows, 1975). Mainly, the differences between the episodes consisted of the construction of partial models. For instance, in the first episode he only built a model of three objects, in the second he extended that model by one object, in the third, the final model was built, but the answer was incorrect since he displaced two objects. In the last episode, he delivered the correct answer after building the complete model from the beginning. Each episode began in a similar way. The subject read the first premise and translated it into a series or model. Then he read the next premise and tried to integrate the new information into a new series, and so forth. However, he did not extend the model unless he was sure about its correctness.

The interpretation of the protocol is that a search tree was successively expanded in long-term memory. If a series was to be extended, the subject had to recognize it as well-known. That is, if a series existed in the tree, the subject became certain about its correctness and then he extended it. Note that he did not utilize the tree in backups, its only purpose is to recognize and check the current model. A detailed discussion of the protocol and its interpretation can be found in Hagert (1980).

### A Simulation Model for the First Solution

Eight rules of the strategy were identified in the protocol and each of them is represented as a production. Five of these are purely strategic rules, containing information about what reading, translation, and integration is and when to do these operations (Hagert, 1980). Two of them are rules for backups, i.e., they contain information about when to begin all over again. A final rule handles the interface between working memory and the search-tree in long-term memory (Hagert, 1980).

In addition to the eight strategic rules, a set of inferential productions are used as background knowledge, i.e., they constitute a hypothesis about the subjects knowledge of spatial concepts. This knowledge is embedded in the three main operations which the subject used. These are: (i) Translate a Proposition (TP) into a model, (ii) Integrate a Proposition (IP) into the model, and (iii) Answer the Question (AQ). They are also represented in productions (see Ohlsson, 1980).

The productions were implemented in the PSS language (Ohlsson, 1979). All in all, the program consists of 169 productions of which 157 are inferential productions. Thus, the eight rules that constituted the strategy are translated into 12 PSS productions (see Hagert, 1980).

The run of the program showed that it almost completely reproduced the first protocol. The wrong answer in the third episode was not simulated. In this episode the program did a direct backup, rather than trying to answer the question as the subject did. With the exception of this event, the program reproduced the contents of the protocol and the backups made by the subject. In other words, the program explains 98 percent of the fragments in the protocol. Thus, it can safely be concluded that the present model is a sufficient model of the strategy which the subject used in his first solution to the task.

#### Stability and Change of the Strategy

The program will, of course, solve the same task in exactly the same way at different occasions. However, as already noticed, the subject did not seem to do so. At least not at a first glance on Table 1. In fact, if the trace of the program is compared to the protocols given by the subject on occasions 2 and 3, the program only explains about half of the fragments. However, a close look at and analysis of the differences show two different factors that change the solution processes without affecting the strategy. That is, the eight rules discussed above are not dramatically exchanged or transformed. The factor that caused the different behaviors on the second occasion is called processing error. The other factor changed the behavior on the third occasion. It consisted in a slight change of one rule in the strategy, and is therefore called a rule modification. They are both briefly discussed below.

On the second occasion the subject did not evaluate the predicates in the premises properly. The model he constructed of, for instance: "Eva is sitting to the right of Carl", was from left to right: Eva Carl. That is, in the resulting model the objects were ordered as they appeared in the premise. When this processing error was corrected, it resulted in another error. The subject detected that he must reorder the objects in the premise to get a correct model. However, he reordered them irrespective of whether the predicate was "left of" or "right of". He entered another processing error. The protocol contains 109 fragments out of a total of 134 in which these two errors prevented him from solving the task. Now, if the program is executed with corresponding processing errors, it reproduces 90 percent of the protocol. The subject used the same set of rules, but the particular solution process was changed and affected by a factor in the situation: the processing errors. Thus, even one year later the program is a sufficient model of the strategy the subject has acquired and used in two solutions.

The behavior on the third occasion was also different from that predicted

or expected by the production system. The factor that changed the behavior was a rule modification. As can be recalled, the subject utilized a search tree to check the correctness of the internal models. However, he did not utilize it in the backups. In the third solution to the same task within two years he began to utilize the search tree more effectively. That is, instead of backing up to the initial state and starting all over again, he backed up to a more recent model in the search tree and continued to extend that model. Thus, one of the two backup rules in the strategy has been modified. The change in the production system corresponds to a slight modification in the action side of this production. This modification implies that the program explains 90 percent of the protocol. Thus, even two years later the subject used the same set of rules, although one of them was modified. The stability of the strategy is obvious.

To sum up, three simulations have indicated that a seemingly dramatic variation in behavioral data does not necessarily imply great differences in the strategy which generated those data. Stability and generality in problem solving behavior can be found, but on the level of strategies. Research into these aspects of strategies is important for the theory of thinking in general, and in particular, for instruction and education.

#### References

- Anzai, Y. & Simon, H. The theory of learning by doing. Psychological Review, 1979, 86, 124-140.
- Hagert, G. Cognitive processing in a spatial series task: A case study. Working Papers from the Cognitive Seminar, Department of Psychology, University of Stockholm, No. 7, 1980.
- Newell, A. & Simon, H. Human problem solving. Englewood Cliffs, New Jersey: Prentice-Hall, 1972.
- Newell, A. Reasoning, problem solving and decision processes: The problem space as a fundamental category. Department of Computer Science, Carnegie-Mellon University, Report No. 133, 1979.
- Ohlsson, S. PSS3 reference manual. Working Papers from the Cognitive Seminar, Department of Psychology, University of Stockholm, No. 4, 1979.
- Ohlsson, S. Competence and strategy in reasoning with common spatial concepts. Working Papers from the Cognitive Seminar, Department of Psychology, University of Stockholm, No. 6, 1980.
- Quinton, G. & Fellows, B. J. 'Perceptual' strategies in the solving of three-term series problems. The British Journal of Psychology, 1975, 66, 69-78.

## SPONTANEOUS SPEECH AS A FEEDBACK PROCESS

Edward Hoenkamp  
Psychological Laboratory  
Nijmegen,  
the Netherlands

Abstract

This paper explores some minimum requirements for a sentence generating program that claims psychological plausibility, in that it can exhibit the dysfluencies that characterize the spontaneous speaker. A monitor is used to compare what the speaker says, with what he intends to say. To this end a feedback loop is introduced which contains a parser. The research reported here is based on empirical evidence in the literature. The implementation is an effort to use the AI paradigm for theory building in psychology.

1. Introduction

From a practical point of view, linguistics is the study of written language. There are however phenomena that are rarely met in written language which occur very frequently in unprepared speech. Such speech comes in fits and starts. It involves pauses, "uh"s, repeats and corrections. These phenomena indicate cognitive activity commonly assumed to be present for the benefit of speaking itself. No wonder that virtually all information about speech execution is derived from data about dysfluencies (in the literature also referred to as "speech errors" or "hesitation phenomena". See [MacLay & Osgood, 1959] for a classic account, and [Clark & Clark, 1977] for an overview. Excluded from my research are slips of the tongue and stutters). Common types of dysfluencies are:

- |                  |  |
|------------------|--|
| (1) Silent pause | Strike the // return key.                    |
| (2) Filled pause | Strike ,uh, the return key.                  |
| (3) Repeats      | Strike the return / the return key.          |
| (4) False starts | Strike the control / the return key.         |
| (5) Corrections  | Strike the rubout -- I mean, the return key. |

A language producing program that can account for errors like these may be said to define a psychologically plausible model of human sentence production. In other words, given an input sequence of conceptualizations, the control structure of the program should determine the kind of errors in a non-ad-hoc fashion.

The process of sentence production involves content, form, and sound. Concepts come up in the mind of the speaker; in some way he selects to-be-verbalized conceptual structures. Call this the process of conceptualizing. To transform these structures into a linguistic form may involve splitting the structure or assembling several structures into messages. Coding the messages into linguistic units is called formulating. Eventually, the output of the formulation process is handed over for overt articulation. In a program, one obviously has to pipeline the inputs and outputs of the processes. In previously existing programs a conceptual structure was input

\* This work was supported by the Netherlands Organization for the Advancement of Pure Research (ZWO), under contract 15-30-06.



into the generator and a complete sentence was to be produced in one go. I.e. conceptualizer and formulator worked sequentially. As will be shown in the next section, it is plausible that in human speech production these processes run in parallel. This introduces a potential interference. Actual speech is rather smooth, so the system must be governed by a device that brings the communication within certain limits. This device is called the monitor.

## 2. Monitoring the formulation process

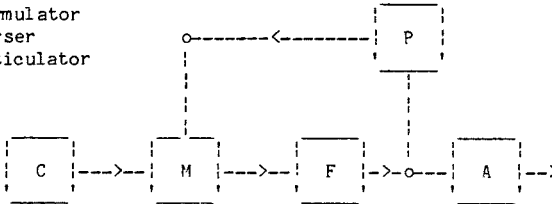
Most people can allude to the sensation of starting a sentence without quite knowing what they intend to say or how to say it. Consequently, the formulator sometimes makes grammatical commitments that may give rise to problems when a new conceptual input arrives. A monitor could be helpful at the conceptual level. Another source of interference is the order in which the conceptual structures are input in the formulator. One cannot a priori assume that words (or groups of words) are produced in this same order. The word-order is a grammatical constraint that should not limit the order in which thoughts arrive. Clearly, the monitor is not only supervising the communication, but also has to watch the end products of all the processes. Suppose the conceptualizer delivers some structure as input to the formulator (generator). The formulator will then begin producing an utterance that expresses the given meaning (close enough in the given circumstances). Half-way through the sentence the conceptualizer comes up with a new structure. The formulator has to get this new information into the unfinished sentence, preferably in a grammatical way. To see what the program does in such situations, look at the following output (the examples were taken from [DuBois, 1974]:

- (a) I want to lease -- or rather, sublease -- an apartment.
- (b) We drove 90 miles -- well, 85 -- all the way to Santa Fe.
- (c) He hit Mary -- that is, Bill did -- with a frying pan.
- (d) I really love -- I mean, despise -- getting up very early.

During the formulation of a sentence, the conceptualizer increments or amends a conceptual structure. In the program this is simulated by simply inputting a list of structures. Take for instance sentence (a). The first structure that is fed into the formulator indicates that some kind of leasing is involved, properly expressed with the verb "lease." After some time the conceptualizer produces a structure that gives a more precise sketch of the action, namely one where the verb "sublease" would be more appropriate. The monitor now recognises that what was said already, differs only in nuance from the new structure. This diagnosis is signaled by the phrase "or rather".

It should be noted that even if conceptual structures have been output, no corrections are needed as long as these structures are not overtly formulated. Therefore, it would not suffice to compare the successive structures produced by the conceptualizer. Instead, the successive conceptual inputs must first be passed through the formulator; only then the monitor compares the next conceptual input with (the meaning of) the partial sentence so far constructed. This is where a feedback from the output is required. The implementation covers the monitor, the generator, and the parser. As said before, the conceptualizer is simulated. The next figure may help to clarify the mechanism.

C = Conceptualizer  
 M = Monitor  
 F = Formulator  
 P = Parser  
 A = Articulator



Simplified diagram of the system.

### 3. Implementation

The moment one starts programming a system for natural language processing, it is necessary to choose a notation to represent the meanings underlying a language utterance. The notation used in this implementation is akin to conceptual dependency. However, care is taken to separate the functions depending on this representation from the main control structures of the system.

#### 3.1. The formulator

Two early ancestors of our formulator are transformational grammar and the ATN mechanism. Of primary concern for a psychologically interesting system, however, is that people most of the time want to convey a meaning (or content), not a form. To do this, a device is needed that transforms a meaning-representation into a linguistic form. Examples of such devices are the generator of Simmons and Slocum [1972], with semantic networks as input, and the one by Goldman [Schank, Goldman, Rieger, 1975] based on conceptual dependency notation. More recently McDonald [forthcoming] implemented a generator that can even be used for a diversity of meaning representations. There is a way in which these generators are a special case of the present formulator. Remember that the formulation (and overt utterance) of a sentence starts before a message has been completely worked out by the conceptualizer. As a result, one of the goals initially set for the generator was piecemeal (or incremental) production. More generally speaking, the aim of the theory was not to use it as part of a Q/A system but to describe a sophisticated model of human sentence production. This has some importance for the linguistics encoded in the actual implementation of the program. The structural flexibility is given precedence over a large vocabulary. This has two consequences. The program uses grammatical constructions that range over a fair amount of possibilities (e.g. virtually all interrogatives in Dutch). The vocabulary itself, on the other hand, is rather small at the time of writing this paper. It contains a few nouns representing animate and inanimate entities. Dutch verbs were partitioned into different groups of similarly behaving verbs. For each group one or two verbs were taken as representatives. In addition, enough function words were included to make sentence building possible. The psychological plausibility of the theory is reflected by the ability of the program to translate a series of cumulative meaning representations into a piecemeal production of the sentence. To do so, the program keeps track of the syntactic restrictions posed by earlier parts of the utterance, while incorporating the new semantic information. The generator was written for Dutch, but a small version for English was

used to produce the given examples.

This section does not cover all the interesting aspects of the generator. E.g. one of the properties of the control structure is that normally constituents are delivered in their entirety. This explains why in examples (2) thru (5) the correction starts at the boundary of the noun phrase, which is in line with the bulk of psycholinguistic findings. All this information, including a precise description of the generator can be found in [Kempen & Hoenkamp, 1979].

### 3.2. The parser.

The parser currently in the system, is a stripped down version of Riesbeck's ELI. Details are to be found in the Yale AI publications (e.g. [Riesbeck, 1975]). A new parser is being developed here, which is more in line with the particular model proposed. It resembles ELI in that it is strongly conceptually guided, and in that it parses a sentence word-by-word, left-to-right while building a meaning representation. The main difference is that it is less top-down (1).

Not only the parsing mechanism itself had to be scrutinized, also for the knowledge base it uses there are some important observations. Speaker and listener do not access information in the same way, and they use a different world-model. In the program therefore the strategy is adopted that the speaker parses his own utterances in a knowledge model he expects to be available to the listener. This may explain what happened in example (c) of section 2. Using the speaker's knowledge base, the formulator decided to use a pronoun for Bill. The parser, using a model of the listener's knowledge base, does not find a referent. The monitor then concludes that reference-editing is necessary, which is signaled by using the phrase "that is".

Another important point is that the speaker knows what he is talking about. This has two observable consequences. First, the listener often corrects the speaker, but there is a strong preference for self-correction. Second, when the speaker parses his own sentences problems of ambiguity or referent resolution may be absent. Evidence for this is that the listener sometimes notices an ambiguity that is not observed by the speaker.

### 3.3. The monitor

At the time of writing, the editing is done via a few simple rules that depend on the specific meaning representation used here. Hopefully a heuristic will be found to diagnose the different kinds of editing in a more general way.

The phrases used during editing ("uh", "I mean", "that is") are more or less specific for the type of editing. (Try to use "well" for "I mean" in: "Sea shells, -I mean-, she sells sea shells...") The types of editing the program can handle are the corrections of the examples (a) through (d), and are classified as: (a) nuance editing, (b) claim editing, (c) reference editing, and (d) mistake editing. The last one is the default. Also the wrong phrase can make a sentence sound less grammatical. These phrases probably bypass the formulator, (and in the program are inserted by the monitor itself). That people do likewise is suggested by the observation that people speaking a foreign language, sometimes use an editing phrase appropriate to their native language.

---

(1) I am very grateful to Larry Birnbaum (Yale-AI), for valuable comments, especially concerning implementation details.

(Some important linguistic issues, such as anaphora and ellipsis, seem to lie within the scope of the monitoring device, and need not be handled by the formulator, as the monitor knows what is said).  
The programs run as separate LISP images under the UNIX operating system which supplies the pipes for communication.

### Bibliography

- Clark, H., & Clark, E.,  
Psychology and Language. New York: Hartcourt, 1977.
- DuBois, J.,  
Syntax in mid-sentence. in: Berkeley studies in syntax and semantics  
(vol. 1).
- Kempen, G., & Hoenkamp, E.,  
A procedural grammar for sentence production. (Submitted for publication. Manuscript (Nov. 1979) available from the authors).
- MacLay, H., & Osgood, Ch.,  
Hesitation phenomena in spontaneous English speech. Word, 1959, 15,  
19-44.
- McDonald, D.,  
Ph.D. Thesis, MIT (forthcoming)
- Riesbeck, C.,  
Conceptual analysis. In: R.C. Schank (ed.), Conceptual information processing. Amsterdam, North-Holland, 1975.
- Schank, R., Goldman, N., Rieger, C.,  
Inference and Paraphrase by Computer, Journal of the ACM, 22, 3 (July 1975), 309-328.
- Simmons, R., & Slocum, J.,  
Generating English Discourse from Semantic Networks, Comm. of the ACM,  
15, 10 (oct. 1972), 891-905.

## How to Program a Society

Kenneth M. Kahn

MIT AI Laboratory (currently at the University of Stockholm)

## ABSTRACT

Conventional programming languages are inadequate for constructing reasoning systems organized as communities of autonomous individuals. They lack a means of conveniently describing the behavior of individuals, the relationships and communication conventions between individuals and subsocieties.

This paper presents the thesis that "actor" languages are ideal as an underlying base in the implementation of societies. The language "Director" is presented as an example of how one can implement individual reasoning agents as actors and their interaction as "message passing". Communication and control conventions are established between the actors to form composite actors that correspond to subsocieties. An example of a large actor system called "Ani" which was implemented in this manner is presented. Portions of Ani's reasoning in creating an animated film are used as illustrations of an actor-based societal reasoning style.

## INTRODUCTION

This paper attempts to deal with some of the issues of reasoning by a society of experts by concentrating on a particular example. The example is drawn from the operation of a program called "Ani" which was designed to create simple computer animation in response to vague high-level story descriptions [Kahn 1979a]. Since Ani is a very large and complex system we select just one small portion of its reasoning in creating an animated version of the story of Cinderella and analyze it in detail. Ani was implemented in an actor language called "Director".

Societal reasoning can best be understood in contrast to the more conventional "monolithic" reasoning. The conventional metaphor for an AI program is an individual which can access or modify facts by searching or modifying its knowledge base. The society metaphor is a community of individuals each of which can directly access and modify its own knowledge and must interact with others when that is insufficient. A monolithic reasoning system's components are very dependent upon their "caller" for direction, context, and resources. The components of a societal reasoning system are independent processes which conceptually are concurrent. The components of a monolithic reasoning system are typically subroutines and database contexts.

Monolithic systems are implemented with subroutines and databases; expert systems are constructed out of "actors" which combine in a single entity both subroutines and databases. An actor is an active computational entity consisting of a procedural component called a "script" and "acquaintances" which roughly correspond its state. Computation in an actor system consists of actors sending messages to each other.

Various computer languages have been built upon the concept of actors. Among them are Smalltalk ([Goldberg 1976] and [Kay 1977]), Act 1 [Lieberman draft], and XPRT [Steels 1979]. Director is the

actor language that was used in implementing Ani [Kahn 1976] [Kahn 1978] [Kahn 1979b]. Each actor in Director consists of a list of methods, a database, its own variables, and a "parent". The methods describe the behavior of an actor. When an actor receives a message the patterns associated with each method are checked to see if they match. If so the action associated with that method is invoked. Otherwise the message is passed along to the "parent" of the actor. This "buck passing" to parents provides simple hierarchical inheritance of behavior. The parent is also queried when an actor does not know about an item in a database or the value of a variable. In this way much knowledge is shared between actors.

#### AN EXAMPLE FROM THE BLOCKS WORLD

Let us consider a simple Director program from the world of toy blocks. Initially there are only cubes and a table. We want to be able to move blocks around and stack them up. The most sophisticated operation is to put a cube on top of another perhaps having to clear them off first. We also want to be able to make inquiries about the locations and relationships of the different objects.

We can define blocks as follows. Notice that most of the work of the program is done by recursively invoking the "clear top" method.

```
(ASK something MAKE block)
;create an actor named "block" whose parent is "something"
(ASK block DO WHEN RECEIVING (move to (top-of ?another)))
;;when I receive a message to move on top of another block
(I clear top) ;;I clear my top
(ASK `another clear top) ;;clear the top of my destination
(I MEMORIZE (underneath-me `another)))
;;I memorize that the other is below me
(ASK block DO WHEN RECEIVING (clear top))
;;when asked to clear my top of blocks
(I RECALL AN ITEM MATCHING (on-top-of-me ?something-above-me) THEN
  (SCRIPT (ASK ,something-above-me move to (top-of the-table))))
(ASK block DO WHEN RECEIVING (MEMORIZE (underneath-me ?some-other)))
;;when asked to memorize that something is underneath me
(I RECALL AN ITEM MATCHING (underneath-me ?something-below-me) THEN
  ;;If I already think something else is below me I forget it
  (SCRIPT (I FORGET ITEM (underneath-me ,something-below-me))
    ;;and tell that other thing to forget it
    (ASK ,something-below-me FORGET ITEM (on-top-of-me ,myself))))
(CONTINUE-ASKING) ;;really memorize it
;;ask the other to remember I'm above it if it doesn't already know it
(ASK `some-other MEMORIZE (on-top-of-me `myself) IF ITS NOT ALREADY))
```

#### THE ROLE OF COMPUTER GRAPHICS

Director is not only a language designed for building object-oriented reasoning and simulation systems but is also a full-fledged graphics and animation language. Actors provide a very powerful and convenient means of describing and programming dynamic graphics. Director has been used to graphically represent complex structures in an "intelligent" manner, in particular a Director program called DIAGRAMER which creates diagrams [Kahn 1979a]. Graphics is a great aid in depicting, debugging, and explaining what is going on in any complex system and so its use in AI programs in general is appropriate. (And perhaps one day its use will be considered indispensable.)

In our blocks world example we would like to be able to see these blocks as they are moved around. For reasons of space, the modifications to our little program to show the blocks as they move has been omitted. Essentially three changes were necessary: changing the blocks to be instances of Director's prototypical graphical actor, adding a method to "Block" giving them an appearance consisting of a label and a square, and modifying "Block's" "move to ..." method to also change the position of the block on the display screen.

#### A COMPARISON WITH GLOBAL DATABASES

Extending our simple program to include objects such as pyramids that cannot have any object on top of them or a robot arm that moves the objects is straight-forward. Blocks can easily be changed to permit more than one block on top of them. If the table has a limited area then it can have a method for allocating space. A history of block movements can easily be kept and used as a source of explanations. These kinds of extensions are no more difficult, and often simpler, than in the more monolithic systems that maintain all this information in a global database.

One difference between this object-oriented approach and a global database system is its greater efficiency at the price of less flexibility. Given a particular object what is immediately above or below it is readily available. Fanning out from there to answer questions like "what is two blocks above it" is not difficult. We can add a method for answering such questions as follows.

```
(ASK block DO WHEN RECEIVING (what is ?n blocks above you)
  (COND ((= n 0) myself) ;;I'm no blocks above myself
    (T (I RECALL EACH ITEM MATCHING (on-top-of-me ?someone-on-me)
      THEN ;;I ask each one above me what is one less above it
      (ASK `someone-on-me what is `(1- n) blocks above you))))))
```

This contrasts strongly with the way that transitive relationships are typically handled in an "assertional" database. The above method is an object-oriented analog of antecedent reasoning. Consequent reasoning can be handled as easily. One possible objection to this way of handling transitive relationships is that it places a much larger burden on the system builder. It also predetermines whether the piece of knowledge that "above is transitive" is to be used in an antecedent or consequent manner. Finally, the fact that "above is transitive" is not itself explicit knowledge that can be reasoned about. One way to resolve these difficulties in an object-oriented framework is to create an actor for transitive relationships and have "above" (among many others) be instances of it. The actor for transitive relationships would then know how to chain together facts. Actors like "Block" will call upon it for help when appropriate.

So far we have been concerned with questions that are directed to a particular object. However, if no object is given then it is often awkward and expensive to find the one referred to in order to ask it a question. A question like "what is on top of the big dark block" to a global database could be answered as follows,

```
(ASK the-blocks-world RECALL AN ITEM MATCHING
  (on-top-of-me {AND {big ?block} {dark ?block}} ?other-block)
  THEN other-block)
```

Since by default in Director an actor has a list of all of its instances we can take a hybrid approach and search through such lists when necessary. The above example would then be

```
(ASK block BROADCAST TO YOUR OFFSPRING
  IF (AND (big (your size)) (dark (your color))) THEN
    (RECALL AN ITEM MATCHING (on-top-of-me ?other-block) THEN other-block))
```

The major computational advantage of an object-oriented organization over a more global one is that certain questions can be answered without any search. The cost is that many other questions are more awkward or expensive to answer. It is the implementor's task to anticipate those questions that will be the most frequent and arrange so that the objects involved directly know the answer.

What is expensive or awkward on a serial computer sometimes becomes cheap and straight-forward on very parallel machines. Actor systems are conceptually parallel and as such much better equipped to take advantage of multi-processors. A global database, for example, must be locked to prevent timing errors and as a result becomes a serious computational bottleneck. With the same knowledge distributed among many actors only a small subset of them need to be locked at any one time.

#### WHAT ANI DOES

Ani is a computer system which, when presented with a high-level description of a film, attempts to create an animated film based upon that description. A user presents Ani with partial descriptions of the personality and appearance of the characters involved, of the relationships and interactions among the characters, and of the type of film desired. Ani integrates this information with more general knowledge and produces a detailed film description that in turn is run by Director.

Ani is a very large program which performs a complex and creative task. Since Ani was implemented in an actor-based language it provides evidence for the claim that actors are useful building blocks for implementing large AI systems. Inside Ani each animated character is an actor, as is every descriptor, character comparison, choice point, plan, method, scene, relationship and activity. The convenience of being able to place knowledge in an actor by adding items to a database associated with each actor and the power of being able to associate arbitrarily complex programs with the same actor were very important in implementing and modifying Ani. The use of actors eased the task of keeping the different components and bodies of knowledge of Ani as independent and modular as possible. Without this high degree of modularity, Ani would have been more difficult to design, implement, and debug.

The style with which an AI program is implemented is very important. It affects the ease of program construction and modification. The style strongly influences what sort of organization and structure an AI program has. The programming language used, in turn, strongly influences what programming styles are practical, convenient, and natural.



# PICKING A SPEED FOR A STEPMOTHER

We now concentrate on a small sample of some of Ani's reasoning while designing an animated version of the story of Cinderella. In making this film Ani must make myriad choices about what should happen in the film and how the different characters should behave. To help establish the personality and relationships of the characters of the film Ani constructs a "typical dynamics" for each character. This describes the normal or default way of moving for a character. Whether a character typically moves quickly and directly or slowly in long graceful curves, whether a character moves boldly or avoids anyone along its path contributes to giving the characters in the films their own personality. A description of Cinderella as happy will tend to make her move faster and in "bouncy" curves. This tendency to move quickly in a bouncy manner can be negated by other descriptions (e.g. that Cinderella is shy) or modified by events (e.g. when running to meet the Prince she may move faster than normal but more directly).

The example we will study in detail is how Ani determines a typical speed for Cinderella's stepmother. The choice must take into account the personality of the stepmother and her relationship with the other characters. The choice cannot be optimally made in isolation but needs to be made in coordination with other choices, especially the choices of the speeds of the other characters. Ani makes this choice by creating "choice points" which are actors who are each responsible for a particular choice. The choice point for the stepmother's speed, for example, is responsible for gathering up suggestions, trying to make sense out of the suggestions gathered, interacting with other choice points, maintaining the current state and justifications of any decisions made, and deciding whether to try to postpone work on its choice.

All that Ani is told about the stepmother is that she is physically ugly and is mean, selfish, strong, and evil. She dominates and hates Cinderella who is obedient and tolerant of her stepmother.

## HOW HER SPEED WAS CHOSEN

"Choice points" are created to represent the process of picking typical speeds for each character. The choice point for the stepmother's speed, for example, begins by asking each of the descriptors of the stepmother for suggestions for her speed. Only the description "Strong" replies and suggests a high speed. The choice point is not happy with just that because there are not enough reliable suggestions. So it asks permission to be postponed to wait for more information to become available and it is granted.

When the choice point for the stepmother's speed is reawakened, it inspects its record of previous activations. It then asks the choice points for the relative speeds of the stepmother and the other characters for suggestions. These choice points are created in response to this request and they choose values (e.g., that the stepmother be faster than Cinderella because she dominates Cinderella and differs from her), but cannot make any concrete suggestions since none of the characters have speeds yet. The choice point for the stepmother's speed asks permission to postpone to wait for the speeds of the others to be determined and it is granted.

The choice points for the other characters also ask and are granted permission to postpone. This could potentially lead to a deadlock in which the four choice points wait for each other to make a decision. One of the reasons the choice points don't just postpone themselves, but instead ask permission first, is to avoid this type of situation. A postponement manager keeps track of the situation and will not grant someone permission to postpone for the same reason twice. A common exception to this is when the choice point is waiting for other choice points to finish and at least one of these is making progress. In this case, no one is making progress so the postponement manager must refuse permission to at least one of the choice points.

Ani is built upon the principle that as few decisions as possible be determined arbitrarily. The decision as to who should be refused permission to postpone has too many consequences to be determined by something like who asks first. Instead the postponement manager asks the "focus", which indicates that conveying the personality of Cinderella is important. The choice point for Cinderella's speed is refused permission causing the speed to be based on the description of Cinderella without being constrained to be faster or slower than the others.

The choice point for the stepmother's speed finally gets suggestions from the relative choice points. It discovers conflicts with one of these suggestions and the earlier suggestion it had received from "Strong" and postpones again. Upon being resumed the choice point asks the descriptions of the film's style for suggestions and receives them from the moderate variety level, high energy level, and low flashiness. Unfortunately they do not all agree and so the choice point postpones one more time.

When it is reawakened it discovers that there are no more sources of suggestions and proceeds with what it has. First it attempts to make compromises between the conflicting suggestions and makes one that in turn generates a new conflict. Excuses are found for rejecting some of the conflicting suggestions. The choice point finally picks a high speed for the stepmother and saves away a justification for this choice.

#### A MORE DETAILED LOOK

Consider the first paragraph of the previous section. What do the sentences mean? How does a "choice point" ask "each of the descriptors of the stepmother for suggestions for her speed"? How can one ask "permission to be postponed"? In this section we present very detailed answers to these questions.

First we consider what the sentence "Choice points are created to represent the process of picking typical speeds for each character" means. The choice point for the stepmother's speed is created as follows,

```
(ASK absolute-choice-point MAKE (choice-point-of stepmother speed))
```

This creates an actor named "(Choice-Point-Of Stepmother Speed)" which is an instance of "Absolute-Choice-Point". This newly created actor just knows its task is to choose a speed for the stepmother. When it cannot handle a message it will ask "Absolute-Choice-Point",

its parent, to handle it. "Absolute-Choice-Point" can handle a few trivial messages and otherwise passes them on along to "Choice-Point" who can handle about ten different messages ranging from requests for making a choice to receiving and combining groups of suggestions. "Choice-Point" in turn passes those messages it cannot handle on up to "Something", Director's prototypical actor.

Some actor wants to know what the stepmother's speed is so it sends the new choice point a message asking it for its choice as follows.

```
(ASK (choice-point-of stepmother speed) RECALL YOUR choice)
```

A method for "recall your choice" is found in "Choice-Point". The real method is long and complex so what follows is a simplified version. The method was added to "Choice-Point" as follows.

```
(ASK choice-point DO WHEN RECEIVING (recall your choice)
  (COND ((I RECALL MY current-choice))
    ;;I answer with my current choice if I have one, otherwise
    (T ;;I recall what my reasons for previously postponing were
      (LET ((postponement-reasons (I RECALL MY postponement-reasons))
        (COND ((NULL postponement-reasons)
          ;;There are no reasons so this is my first try
          (I combine suggestions
            ;;I combine the suggestions I get by asking
            ! (I collect suggestions from
              ;;myself for suggestions from
              ;;the first of my suggestion sources
              ^ (FIRST (I RECALL MY suggestion-sources))))))
        (T ;;If I've previously postponed work on my choice
          (I continue to recall my choice
            despite ^postponement-reasons))))))
    ;;I try to continue, taking into account the difficulty. The methods
    ;;for handling this gather more information, resolve old conflicts
    ;;between suggestions, or try to postpone work until more is known
```

The running of this method produces the following series of transmissions.

```
(ASK (choice-point-of stepmother speed) RECALL YOUR current-choice)
NIL ;;NIL is returned indicating no choice has been made yet
(ASK (choice-point-of stepmother speed) RECALL YOUR postponement-reasons)
NIL ;;Returning NIL means that this is first time
(ask (choice-point-of stepmother speed) recall your suggestion-sources)
;;The following is returned after being found in "absolute-choice-point".
((absolute-suggestions ;;These first three sources are grouped together
  neighbors-absolute-suggestions ;;indicating that they should be
  opposites-absolute-suggestions) ;; explored together
  relative-suggestions global-suggestions)
;;This results in the next transmission
(ASK (choice-point-of stepmother speed)
  collect suggestions from (absolute-suggestions
    neighbors-absolute-suggestions
    opposites-absolute-suggestions))
```

The last transmission invokes a method in "choice-point" which collects suggestions of each type by sending messages which invoke methods such as the following.

```
(ASK choice-point
DO WHEN RECEIVING (collect suggestions from absolute-suggestions)
;;when I get a message asking for absolute suggestions
(I ASK MY thing ;;I ask the object I am making some choice about
;;to collect suggestions for the element in question
collect suggestions for `(I RECALL MY element)))
```

This method causes a message to be sent to the stepmother asking her to collect suggestions for her speed as follows.

```
(ASK stepmother collect suggestions for speed)
```

The method invoked by this transmission had been added to an actor named "character", the stepmother's parent, as follows.

```
(ASK character DO WHEN RECEIVING (collect suggestions for ?element)
;;when I get a message asking for suggestions for an aspect of myself
(I RECALL EACH ITEM MATCHING
;;for every item in my database matching the following pattern
(description type ? ;;any type of descriptor is fine
descriptor ?the-descriptor ;;call it "the-descriptor"
source ?) ;;any source
THEN (SCRIPT
(ASK ,the-descriptor COLLECT ITEMS MEMORIZED MATCHING
;;I ask the descriptor to search for items matching
;;a suggestion whose element is what we are looking for
(suggestion element ,element
value ? strength ? source ?))))))
```

The method above initiates the following transmissions.

```
(ASK strong COLLECT ITEMS MEMORIZED MATCHING
(suggestion element speed
value ? strength ? source ?))

((suggestion element speed ;;a high speed is recommended
value high strength medium source strong))
;;"ugly", "evil", "mean", and "selfish" are also asked but reply NIL
```

The "(Choice-Point-Of Stepmother Speed)" returns this suggestion in response to the message "(collect suggestions for absolute-suggestions)". There are still two other suggestion sources waiting to be tapped "Neighbors-Absolute-Suggestions" and "Opposites-Absolute-Suggestions". They refer to suggestions from the synonyms and antonyms of the descriptors of the stepmother. In this case they return no suggestions at all.

The suggestions gathered are then combined with any other suggestions previously gathered. In this case there is only the one from "Strong", so no conflicts are looked for, no compromises sought, or problems postponed. (Combining suggestions is a complex process that would at least double the length of this section if included.) This suggestion from "Strong" is added to the database of "(Choice-Point-Of Stepmother Speed)". The "suggestion-sources" of the "(Choice-Point-Of Stepmother Speed)" is set to those sources left, i.e. "(relative-suggestions global-suggestions)".

We are now near the end of the paragraph. "The choice point is not happy with just that because there are not enough reliable suggestions. So it asks permission to be postponed to wait for more information to become available and it is granted." An actor called "Postponement-manager" is consulted as to what to do.

```
(ASK postponement-manager
  should (choice-point-of stepmother speed) postpone with
    ((suggestion element speed ;;this suggestion
      value high
      strength medium
      source strong))
    0 conflicts) ;;and no conflicts
(postpone not-happy-enough) ;;is the Postponement-manager's answer
;;The (Choice-Point-Of Stepmother Speed) then remembers the reason.
(ask (choice-point-of stepmother speed)
  add not-happy-enough to your list of postponement-reasons regardless)
```

At this point the choice point has altered its state so that upon reawakening it will know why it postponed previously, what sources of suggestions are left, which ones have yielded suggestions, what the suggestions have been gathered so far are, and what conflicts have yet to be resolved. The choice point finally returns "(postponed not-happy-enough)" in response to the original message "choose a value". By doing this it has indicated to the actor that originally asked it for its choice that it has postponed its choice until later. This actor can try to postpone or can decide to go ahead without knowing how the choice point will decide.

Conceptually all the choice points are running in parallel and when they run into difficulties it is up to them to ask permission to go to sleep. It is typically to the choice point's advantage to postpone work when faced with difficulties because upon awakening more information is available. It is to the system's advantage to grant permission since it frees up computational resources. However, to make sure that some progress is being made it is necessary to have a postponement manager with a more global view of the situation which occasionally refuses permission to postpone thereby forcing the choice point in question to make due despite its difficulties.

Rather than the more conventional manner of putting processes to sleep by saving their state and resuming them at the same point they were interrupted, the choice points save away what state information they please and are woken by simply sending them a message requesting a choice. It is up to the choice point to go back to what it was doing when it went to sleep or to try something new. The explicit form of "memoization" of partial results and difficulties by choice points proved to be flexible and general. It is up to each actor to obtain permission to postpone and to save away what it wants for its reawakening. The major disadvantage of this explicit memoization is that any actor that uses it must have code that asks permission to postpone, that records a partial state description, and inspects such descriptions upon reawakening. The inheritance of methods and state from more generic actors greatly reduces this burden on an implementor however.

This long description of just the first attempt (out of ten) to choose a typical speed for the stepmother is still sketchy. I hope it has served its purpose of conveying what is it is like for a reasoning system to operate as a society of experts.

#### REFERENCES

- [Goldberg 1976]  
Goldberg, A., Kay A. editors, "Smalltalk-72 Instruction Manual" The Learning Research Group, Xerox Palo Alto Research Center, March 1976
- [Hewitt 1977]  
Hewitt, C., "Viewing Control Structures as Patterns of Passing Messages" "Artificial Intelligence", Vol 8. No. 3, June 1977,
- [Kahn 1976]  
Kahn, K., "An Actor-Based Computer Animation Language", ed. Treu, S., pp. 37-43 Proceedings of the SIGGRAPH/ACM Workshop on User-Oriented Design of Interactive Graphics Systems, October 1976
- [Kahn 1978]  
Kahn K. and Hewitt C., "Dynamic Graphics using Quasi Parallelism", "Computer Graphics" Vol. 12, No. 3, August 1978
- [Kahn 1979a]  
Kahn, K., "Creation of Computer Animation from Story Descriptions", MIT AI Laboratory TR 540, August 1979
- [Kahn 1979b]  
Kahn, K., "Director Guide", MIT AI Memo 482B, December 1978
- [Kay 1977]  
Kay, A., "Microelectronics and the Personal Computer", Scientific American, September 1977
- [Kornfeld 1979]  
Kornfeld W., "Using Parallel Processing for Problem Solving", MIT EECS Masters Thesis, Spring 1979
- [Lieberman draft]  
Lieberman, H., "A Preview of Act 1", submitted for publication
- [Steels 1979]  
Steels, L., "A XPRT Description System", MIT AI Working Paper 178, January 1979

A SYSTEM FOR PROGRAM SYNTHESIS AND PROGRAM OPTIMIZATION.

Yves Kodratoff G.R.22 du C.N.R.S.

Institut de programmation  
4 place Jussieu  
75230 PARIS CEDEX 05

Eric Papon Université d'Orsay  
Laboratoire de recherche en informatique  
Batiment 490  
91405 ORSAY CEDEX

**Abstract.** This paper presents an implemented system for the synthesis of tail recursive programs from input-output examples. The system can be also used to perform some new recursive to iterative transforms.

1. INTRODUCTION

It has been shown that one can synthesize non trivial programs from input-output examples [2,3,6,7,8,9,12,14,19] but little effort has been made in the direction of synthesizing optimized programs (see however [10]). We present here a system which optimizes the programs relative to a "tail-recursive" criterion: as far as possible, we obtain tail-recursive programs that are either easily transformed into iterative programs or directly compiled without stack in V-LISP [5]. It has been shown [1,4] that recursive to iterative transformations eventually improve also the computation time as exemplified by the Fibonacci function which has a  $2^n$  complexity under its recursive form while it drops down to  $n$  under its tail-recursive form. The programs we obtain are not always optimal, nor we reach a very large class of functions because we were mainly concerned by the avoidance of any combinatory explosion.

Our system divides into two very different parts. One is used to transform the input-output examples into computation traces. The second is used to transform the computation traces into programs. We will further show how this second part can be directly used as a powerful recursive to iterative program translator.

2. TRANSFORMING INPUT-OUTPUT EXAMPLES INTO COMPUTATION TRACES.

We start from a finite sequence of input-outputs  $\{X_i, Y_i\}$  where  $i$  is less than an upper bound 1, where  $X_i$  and  $Y_i$  are lists. We restrict ourselves to "ascending linear domains" [9] so that the sequence  $X_i$  is totally ordered. We transform the given sequence into a sequence of traces  $\{(P_i X), (F_i X)\}$  where  $(P_i X) = \text{True}$  if  $X = X_i$ , False if  $X$  is greater than  $X_i$ . This last sequence can be embedded into IF...THEN...ELSE... functions and becomes:

$(F_i X) = \text{IF } (P_i X) \text{ THEN } (F_i X) \text{ ELSE } \dots$

$\text{IF } (P_i X) \text{ THEN } (F_i X) \text{ ELSE undefined.}$

The limit of  $(F_i X)$  when  $i$  tends toward infinity can be easily shown [19] as equivalent to a program which can be found by a study of the recurrent properties of  $(P_i X)$  and  $(F_i X)$ .

By hypothesis on the domain we know that  $(P_{i+1} X) = (P_i(b X))$  where  $b$  is a finite composition of CAR and CDR (thereafter called a reduction function). Section 5 is devoted to the description of an algorithm which allows to find the recursion relations that link  $F_i$  and  $F_{i+1}$ .

**Definition.**

Let  $A_i$  be the  $i$ th occurrence of the atom  $A$  in the list  $x$  and  $u$  be the unique reduction function such that  $(u x) = A_i$ . Then  $u$  is said to be the functional name of  $A_i$  in  $x$ .

In order to obtain  $(F_i X)$  from  $\{X_i, Y_i\}$ , each atom of  $Y_i$  is given its functional name in  $X_i$ . This leads to  $(F_i X)$  and we state  $(F_i X) = (F_i X)$ .

**Example.**

Consider the function  $(F X)$  where  $X$  contains  $k$  atoms at its top-level.  $F$  concatenates to the first atom of  $X$ , the  $(i+1)$ th and the  $i$ th for  $i=1$  to  $k-1$ . An input-output sequence is therefore:  $\{(A) \rightarrow (A), (A B) \rightarrow (A B A), (A B C) \rightarrow (A B A C B), (A B C D) \rightarrow (A B A C B D C), \dots\}$ . We have shown [11] that  $(P_i X) = (\text{ATOM}(\text{CDRI } X))$  where  $\text{CDRI}$  is the  $i$ th power of CDR. It follows that  $(P_{i+1} X) = (P_i(\text{CDR } X))$ .

The functional name of A in  $X_2 = (A\ B)$  is  $(CAR\ X_2)$  and of B in  $X_2$  is  $(CADR\ X_2)$ , since  $Y_2 = (CONS\ A\ (CONS\ B\ (CONS\ A\ NIL)))$  we have :

$(F\ X_2) = (CONS\ (CAR\ X_2)\ (CONS\ (CADR\ X_2)\ (CONS\ (CAR\ X_2)\ NIL)))$  and  
 $(F_2\ X) = (CONS\ (CAR\ X)\ (CONS\ (CADR\ X)\ (CONS\ (CAR\ X)\ NIL)))$ .

One finds in the same way that :

$(F_1\ X) = (CONS\ (CAR\ X)\ NIL)$  ,  
 $(F_3\ X) = (CONS\ (CAR\ X)\ (CONS\ (CADR\ X)\ (CONS\ (CAR\ X)\ (CONS\ (CADDR\ X)\ (CONS\ (CADR\ X)\ NIL))))))$   
 $(F_4\ X) = (CONS\ (CAR\ X)\ (CONS\ (CADR\ X)\ (CONS\ (CAR\ X)\ (CONS\ (CADDR\ X)\ (CONS\ (CADR\ X)\ (CONS\ (CADDR\ X)\ NIL))))))$  .

This procedure has been implemented several times [14,15,19].

Remark.

It may happen that some atoms of Y do not appear in X. These atoms are interpreted as constant functions. For example, let  $(A) \rightarrow (A\ B\ C)$  then  $(F_1\ X) = (CONS\ (CAR\ X)\ (CONS\ 'B\ (CONS\ 'C\ NIL)))$ .

### 3. TRANSFORMING A PROGRAM INTO A COMPUTATION TRACE .

A recursive definition contains a recursive symbol (let us call it F) and the symbols of auxiliary functions. For instance, a classical program scheme reads :

$(F\ X) = IF\ (P\ X)\ THEN\ (H_1\ X)\ ELSE\ (H_2\ (G_1\ X)\ (F\ (E\ X)))$  where it is supposed that the auxiliary predicate P takes the value True when E has been applied n times to X and where  $H_1, H_2, G_1$  and E are the auxiliary functions .

We define the nth trace of  $(F\ X)$  by the result obtained when

1-one supposes that  $(P\ (E\ power\ i)\ X) = False$  if  $i < n-1$  and  $(P\ (E\ power\ n)\ X) = True$ .

2-the following computation is carried on :

-the computation of the auxiliary functions is suspended

-the computation of the branches that contain the recursive symbol is carried on as far as  $((E\ power\ n)\ X)$  is not reached.

For example, if X is such that  $(P\ X) = True$  then the trace of the above scheme is  $(H_1\ X)$ . If X is such that  $(P\ (E\ X)) = True$  then its trace is  $(H_2\ (G_1\ X)\ (H_2\ (G_1\ (E\ X))\ (H_1\ (E\ X))))$ .

In practice, one obtains easily the function FP associated to F and which writes down the trace of F :

$(FP\ X) = (FP_1\ X\ 'X)$   
 $(FP_1\ X\ Z) = IF\ (P\ X)\ THEN\ (CONS\ 'H_1\ (CONS\ Z\ NIL))\ ELSE$   
 $(CONS\ 'H_2\ (CONS\ (CONS\ 'G_1\ (CONS\ Z\ NIL))\ (CONS\ (FP_1\ (E\ X)\ (CONS\ 'E\ (CONS\ Z\ NIL)))\ NIL)))$

where P and E are now actually evaluated.

### 4. BASIC LEMMAS AND THEOREMS.

The definitions of a substitution, a matching, a least general term and the algorithms that compute them can be found in [16,17,18].

We try to find recursion relations that link  $(F_i\ X)$  and  $(F_{i+1}\ X)$ . This is equivalent to say that we attempt a matching between  $F_i$  and  $F_{i+1}$  for all i and if

-for all i the matching succeeds

-for all i the obtained substitutions are the same

then we have reached our goal since the success of a matching implies substitutions of the type (variable\term), these substitutions induce trivially recursion relations valid for each i (because the substitutions are constant for each i).

Definition.

Let  $N_{xi}$  be the number of leaves of the input X such that  $(P_i\ X) = True$ , let  $N_{yi}$  the number of leaves of the corresponding output  $(F_i\ X)$ . We shall say that F is of polynomial increase if  $N_{yi}$  is a polynomial in  $N_{xi}$ . Notice that we demand that the increase behaves exactly as a polynomial (i.e. we do not synthesize the whole class of functions the increase of which is BOUNDED by a polynomial).

Lemma 1.

If F is of polynomial increase and if one can find recursion relations from the traces of F then these relations involve a finite number of substitutions : one needs a finite number of variables in order to write these relations.

Definitions.



Let  $s$  be a substitution, we shall write  $V^* = \{V_1, \dots, V_p\}$  a vector of  $p$  variables modified by  $s$ , namely :

$s = (V_1 \setminus (l_1 V^*), \dots, (V_p \setminus (l_p V^*)))$ .

We write  $N_i$  the number of variables that actually appear into  $(l_i V^*)$ . For instance, if  $(V_1 \setminus (\text{CONS } V_1 (\text{CONS } V_2 \text{ NIL})))$  then  $N_1=2$ .

Lemma 2.

Let  $F$  of polynomial increase. Suppose that we find a substitution  $s = (V_1 \setminus (l_1 V^*), \dots, (V_p \setminus (l_p V^*)))$ , where the terms of the substitution have been ordered by  $N_1 < N_2 < \dots < N_p$  ( it follows that the traces of  $F$  have the recursion relation  $(F_{i+1}) = (F_i(l_1 V^*) \dots (l_p V^*))$ .

then  $N_1=0$  or 1 and  $N(i+1)=N_i$  or  $N(i+1)=N_i + 1$ .

Theorem. [19]

Recall the definition of FL in section 2. Let  $F$  defined as Sup FL. We suppose that there exist 2 substitutions :

$s_1 = (Y \setminus (b Y)), s_2 = (V_1 \setminus (l_1 V^*), \dots, (V_p \setminus (l_p V^*)))$  such that

$(F_{i+k} Y) = (F_i (b Y))$

$(F_{i+k} V^*) = (F_i (l_1 V^*) \dots (l_p V^*))$

then  $F$  equals :

$(\text{DE } F(Y V^*) (\text{COND}$

$((P_1 Y) (F_1 V^*)$

$\dots$

$((P_k Y) (F_k V^*)$

$(T(F(b Y) (l_1 V^*) \dots (l_p V^*)) ) )$ .

This theorem simply states that a program is obtained from the recursion relations by adding a counter that expresses the properties of the domain.

This theorem can be easily extended to the primitive recursive embedding [12] where

$(F_{i+k} V^*) = (H_i V^* (F_i (l_1 V^*) \dots (l_p V^*)))$

It can also be extended to composition [13] where

$(F_i V^*) = (H_i V^* (G_i V^*))$  for all  $i$ .

As far as recursion relations ( valid for each  $i$ ) are found , then the corresponding function is known. It remains to be solved the difficult problem of finding these recursion relations. We name BMWk the following algorithm which gives a partial answer to this problem when one wishes to avoid a combinatory search.

## 5. THE ELEMENTARY BMWk ALGORITHM.

5.1. General principle. The general principle of BMWk is strikingly simple : try to match  $(F_i X)$  and  $(F_{i+1} X)$ , for  $1 \leq i \leq l-1$ , if it succeeds with constant substitutions then the problem is solved. If the match is a failure or the substitutions do not remain constant then transform the sequence  $(F_i X)$  into the sequence  $(G_i V_1^*)$ ,  $1 \leq i \leq l-1$ , where  $(G_i V_1^*)$  is the least generalization of  $(F_i X)$  and  $(F_{i+1} X)$  and  $V_1^*$  the vector which contains the variables issued from this generalization. Notice that we dispose now of  $l-1$  examples only.

One attempts to match  $(G_i V_1^*)$  and  $(G_{i+1} V_1^*)$ ,  $1 \leq i \leq l-2$ , and , again, it is either a success and we stop here or a failure and we generalize each pair  $(G_i V_1^*), (G_{i+1} V_1^*)$  to their least generalization  $(H_i V_2^*)$ . This process is carried on as far as we do not reach a "lethal success" (further defined in 5.2.1) or a complete generalization (all the terms are generalized to a variable ).

Checking that all substitutions are the same is a problem by itself. This is generally not the case so that we get sequences of the type  $(V_1 \setminus (l_1 V^*))$  where  $l_1$  varies with each  $i$ . We treat it as a new sequence of traces which we try to reduce to a function  $(H V^*)$ . When this is possible the substitutions take the form  $(V_1 \setminus (H V^*))$  which is now constant. In other words, non-constant substitutions give raise to new sub-problems. Recognizing these sub-problems nature and their management will be described in the following after a few Definitions.

We distinguish several types of variables

1-the domain variable, called  $Y$ , and the predicates associated to the domain are such that  $(P_{i+1} Y) = (P_i(b Y))$ .

2-the variables initiated by the initial variable X and coming from generalizations on this X variable are named Xi. For instance one can have (F X)=(FP X X X) and we define FP by (DE FP(X1 X2 Y)...).

3-the variables issued from a composition are called XXi. For instance, if (F X)=(G X(H X)) we then define G by (DE G(X XX)...).

4-the variables issued from a generalization of constants are called Zi. For instance, if (F X)=(G X '1 NIL) we then define G by (DE G(X Z1 Z2)...). We therefore have 3 types of vectors that contain X, XX or Z variables. These vectors will be named X\*,XX\* or Z\*.

Recall that we match Fi and Fi+1,  $1 \leq i \leq l-1$ , and that we obtain  $l-1$  substitutions  $s_1, s_2, \dots, s_{l-1}$ , i.e. one substitution for each matching.

There are then 3 possible cases.

-first case.  $s_1=s_2=\dots=s_{l-1}$ =Failure.

The matching has failed and we try to generalize again.

-second case. si is a success for each i. The synthesis process begins and is described in 5.2.

-third case.  $s_1=\dots=s_k$ =Failure for  $k < l-1$  but  $s_{k+1}, \dots, s_{l-1}$  are successes. The  $k-1$  first traces are considered as particular cases and the function F is written as : (DE F(X)(COND((P1 X)(F1 X))...((Pk X)(Fk X)) (T (G X)) ))

where (G X) is synthesized from the kth to lth examples.

5.2. The matchings succeed.

5.2.1. The lethal successes.

5.2.1.1. In the substitution si appears a pair (X\((c X)) and there exists no substitution e such that  $b=e.c$  where b is the substitution which characterizes the domain. If in the data type we are working on, c is ,like b, the inverse of a constructor, then the substitution c "destroys" quicker than b and leads to undefined computations after some recursive calls. Example. Let (A B C D) $\rightarrow$ (A),(A B C D E) $\rightarrow$ (C),(A B C D E F) $\rightarrow$ (E),... We obtain  $b=CDR$  while the function traces are (F1 X)=(CONS(CAR X)NIL), (F2 X)=(CONS(CADDR X)NIL), (F3 X)=(CONS(CADDDDR X)NIL),... These traces lead to (X(CDDR X)), i.e.  $c=CDDR$  and there is no e such that  $b=e.c$ . One verifies that these examples stop at (A B C D E F G).

5.2.1.2. The substitution si contains a pair (Xk\((c Xj)) with  $j \neq k$ . Either this happens a finite number of times and we are sent to 5.6 or this never stops and we would need an infinity of variables.

5.2.1.3. The number of substituted variables must be constant in each si.

5.2.1.4. Suppose that we start from 1 examples and that we need  $l-2$  successive generalizations, we should be left with 2 examples.

5.2.1.5. Define an order of complexity on the terms by the following rules. Let T1 and T2 be CONS trees, we say that T1 is less complex than T2  $\iff$  T1 contains less CONS nodes than T2  $\iff$  if they contain the same number of nodes then T2 contains more CAR and CDR than T1.

Suppose that a generalization leads to replace a subterm T1 of (Fi X) by a variable XX1 and that XX1 is found to receive the substitution (XX1\T2). This expresses the fact that (T1\T2). If T1 is more complex than T2, it is clear that this operation cannot be too often repeated : if T2 is replaced by T3 of less complexity and so on then the terms would vanish. This cannot lead to constant substitutions.

5.2.2 The building of s such that  $F_{i+1}=s.F_i$ . One picks up the constant substitutions that appear in  $s_1, \dots, s_{n-1}$  and they constitute the first part of s.

The other substitutions will lead to sub-problems. For instance, if one has

in  $s_1$  : (XX1\((H1 X\* XX\* Z\* ))

... in  $s_{n-1}$  : (XX1\((Hn-1 X\* XX\* Z\* )) then we have a new problem with the predicates P2, ..., Pn-1 and the traces H1, ..., Hn-1. If one is able to solve this new problem, the pair (XX1\((H X\* XX\* Z\* Y)) will be added to the substitution s (notice that the domain variable Y has been added in order to obey the theorem of section 4).

5.2.3 An example. (with a trivial generalization)

We start from the (Pi X) and (Fi X) described in the example of section 2. The matching of (Fi x) and (Fi+1 X) fails because one gets substitutions like (NIL\G X)) which lead to a failure (recall that a matching succeeds only if all

substitutions have a variable in the left place, and NIL is a function of 0-arity). We therefore take the least generalized of (F1 X) and (F2 X) which is (G1 X Z)=(CONS(CAR X) Z) (and of course recall that (G1 X NIL)=(F1 X)). The least generalized of F2 and F3 is (G2 X Z)=(CONS(CADR X)(CONS(CAR X) Z))) and the least generalized of F3 and F4 is (G3 X Z)=(CONS(CAR X)(CONS(CADR X)(CONS(CAR X)(CONS(CADR X)(CONS(CADR X) Z)))))). The matching between (Gi X Z) and (Gi+1 X Z) succeeds with

```
s1= (X\X),(Z\ (CONS(CADR X) Z)))
s2= (X\X),(Z\ (CONS(CADR X)(CONS(CADR X) Z)))
We therefore state : s=(X\X),(Z\ (H X Y Z)) ,where H is given by :
IF (P2 X) THEN (CONS(CADR X)(CONS(CAR X) Z))
ELSE
IF (P3 X) THEN (CONS(CADR X)(CONS(CADR X) Z))
ELSE UNDEFINED
It follows that sH1= (X\ (CDR X)),(Z\Z)
sH2= (X\ (CDR X)),(Z\Z)
```

and the reader can verify that sH3 is the same : sH = (X\ (CDR X)),(Z\Z) so that H is defined by

```
(DE H(X Y Z)(COND
  ((ATOM(CDDR Y))(CONS(CADR X)(CONS(CAR X) Z)))
  (T(H(CDR X) (CDR Y) Z) )))
```

and F by

```
(DE F(X) (G X X NIL)) (DE G(X Y Z)(COND
  ((ATOM(CDR Y))(CONS(CAR X) Z))
  (T (G X (CDR Y) (H X Y Z) ) ) ) )
```

Notice that Y is always initiated by X since the initial problem is IF (Pi X) THEN (Fi X) , that the stopping condition of G is IF (P1 Y) THEN (G1 X Z), that the stopping condition of H is IF (P2 Y) THEN (H1 X Z), that G and H recur as indicated by s and sH, that Z is initiated by NIL since one has (Gi X NIL)=(Fi X) for each i. 5.3 The matching fails: non-trivial generalizations.

5.3.1 We apply first the methodology described in 5.1 : this process must stop after a finite time : we can introduce an only finite number of variables. The main problem consists in the link to be found between the variables introduced at each generalization (when only one of them exists, like in 5.3.1 this link is trivially found).

5.3.2. An example.

Consider the (ad'hoc) sequence of input-outputs :

```
{ (A)->(A A A) ; (A B)->((A)(B A) B) ;
(A B C)->(((A))((C B A)B)C) ; (A B C D)->((((A))(((D C B A)C B)C)D) ;
(A B C D E)->((((A))(((E D C B A)D C B)D C)D)E}}
```

The traces are :

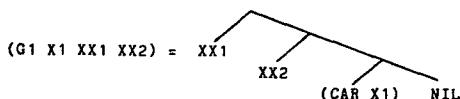
```
(F1 X)=(CONS(CAR X)(CONS(CAR X)(CONS(CAR X) NIL))) ,
(F2 X)=(CONS (CONS (CAR X) NIL)
  (CONS (CONS(CADR X)(CONS(CAR X)NIL)) (CONS(CADR X) NIL)))
```

and we leave to the reader the computation of F3 and F4.

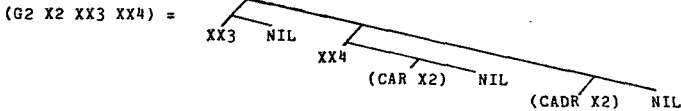
The matchings fail and we get the least generalizations:

(in order to help the reader some CONS trees will now be written as actual trees a node of which is always a CONS)

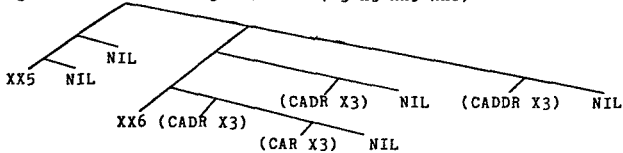
-generalization of F1 and F2 :



-generalization of F2 and F3 :



-generalization of F3 and F4 :  $(G3 \ X3 \ XX5 \ XX6) =$



We have therefore :

$$s1 = (XX1 \setminus XX3 \text{ NIL}), (XX2 \setminus XX4 \text{ (CAR X2) NIL}), (X1 \setminus (CDR X2))$$

$$s2 = (XX3 \setminus XX5 \text{ NIL}), (XX4 \setminus XX6 \text{ (CADR X3) (CAR X3) NIL}), (X2 \setminus (CDR X3))$$

5.3.3 We are now able to describe our methodology with the help of the above example. Recall now the definition of  $N_i$  in lemma 2 of section 4 : the substitutions in  $s_i$  are ordered by increasing  $N_i$ . In the example  $s_1$  and  $s_2$  become :

$$s1 = (XX1 \setminus XX3 \text{ NIL}), (X1 \setminus (CDR X2)), (XX2 \setminus XX4 \text{ (CAR X2) NIL})$$

$$s2 = (XX3 \setminus XX5 \text{ NIL}), (X2 \setminus (CDR X3)), (XX4 \setminus XX6 \text{ (CADR X3) (CAR X3) NIL})$$

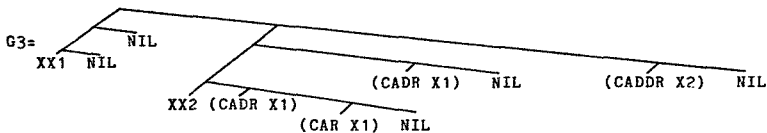
By lemma 2 of section 4, when a substitution contains one variable in its left part and one other variable in its right part, these two variables must bear the same name. This allows to give the right name to the substitution with  $N_i=1$ . In the example, we know by  $s_1$  that  $XX1$  and  $XX3$  must bear the same name, say  $X1$ , then we know that in  $s_2$   $XX3$  and  $XX5$  must both be called  $X1$ . The same is true for  $X1, X2$  and  $X3$ . Now, again by lemma 2, we know the existence of a substitution with  $N_i=2$  and this substitution contains at most one of the preceding variables. We give its name to this variable and this leaves only one unknown variable which is as before readily renamed. This process is iterated as long as unknown variables are left. In the example the substitutions to  $XX2$  and  $XX4$  contain  $X2$  and  $X3$ , known for being equal to  $X1$ . We therefore write :

$$(XX2 \setminus XX4 \text{ (CAR X1) NIL}) \quad (XX4 \setminus XX6 \text{ (CADR X1) (CAR X1) NIL})$$

and  $XX2, XX4$  and  $XX6$  are identified to only one variable, say  $XX2$ . The generalizations are accordingly re-written :

$$G1 = XX1$$

$$G2 =$$



#### 5.4 Variables initialization.

We have now the initial sequence  $F_i$  and the associated sequence  $G_i$  of generalizations: we have to instantiate the variables of  $G_i$  from the variables of  $F_i$  and from constants. We match  $G_i$  and  $F_i$   $1 \leq i \leq k-1$  if  $k$  generalizations have been necessary and obtain substitutions  $z_1, z_2, \dots, z_{(1-k-1)}$ . Recall that from the sequence  $G_i$  we obtained also  $s_1, s_2, \dots, s_{(1-k-2)}$  substitutions. In  $z_i$  we have pairs like  $(XX1 \setminus (M_i X^* XX^* Z^*))$  and in  $s_i$  we have pairs like  $(XX1 \setminus (L_i X^* XX^* Z^*))$ . If any is more complex than  $L_i$  we know that we have reached the lethal success 5.2.1.5 and we proceed to 5.5. If not, we act as in 5.3 and build a unique substitution  $z$  which will be valid for all  $G_i$ :

$z = (X^* \setminus (M_1 X^*)), (X^* \setminus (M_2 X^* XX^* Z^*)), (Z^* \setminus (M_3 X^* XX^* Z^*))$   
 and  $F$  is written as a composed function:  $(DE F(X^* XX^* Z^* Y))(G(M_1 X^*))(M_2 X^* XX^* Z^*)(M_3 X^* XX^* Z^* Y))$  where  $G$  has  $G_1, \dots, G_{1-k-2}$  as traces and the same predicates as  $F$ .

Example.

We use again the above example and match  $G_i$  and  $F_i$  in order to obtain:

$z_1 = (XX1 \setminus (CAR X)), (XX2 \setminus (CAR X)), (X1 \setminus X)$

$z_2 = (XX1 \setminus (CAR X)), (XX2 \setminus (CADDR X)), (X1 \setminus X)$

$z_3 = (XX1 \setminus (CAR X)), (XX2 \setminus (CADDR X)), (X1 \setminus X)$

it follows that  $z = (XX1 \setminus (CAR X)), (XX2 \setminus (H X Y)), (X1 \setminus X)$  where  $(H X Y)$  has the predicates  $(P_1 Y), (P_2 Y), (P_3 Y)$  and the traces  $(CAR X), (CADDR X), (CADDR X)$  so that  $(H X Y)$  is defined by:

```
(DE H(X Y)(COND
  ((ATOM(CDR Y)) (CAR X))
  (T (H (CDR X) (CDR Y)))))
```

and the function  $(F X)$  synthesized from the examples is:

```
(DE F(X) (G X (CAR X) (H X X) X))
(DE G(X1 XX1 XX2 Y)(COND
  ((ATOM(CDR Y))(CONS XX1 (CONS XX2 (CONS (CAR X1) NIL))))
  (T(G(CDR X1) (CONS XX1 NIL) (I X1 XX2 Y) (CDR Y)))))
```

The reader can easily find from  $G_1, G_2, G_3$  in 5.3.3 that  $(I X1 XX2 Y)$  is given by:

```
(DE I(X XX Y)(J X XX Y NIL))
(DE J(X XX Y Z)(COND
  ((ATOM(CDDR Y)) (CONS XX (CONS (CAR X) Z)))
  (T (J (CDR X) XX1 (CDR Y) (CONS (CAR X) Z)))))
```

#### 5.5 Composition and the lethal successes.

The general methodology is again very simple. We find the functional name of the place at which occurs the lethal success. The tree  $F_i$  is then cut into two parts: the sons of the first CONS above the lethal success and their context. We therefore obtain  $(F_i X) = (G_i X (H_i X))$  where  $(G_i X Z)$  is the context and  $(H_i X)$  the sons of the CONS above the lethal success. We then obtain  $F$  as the composition of two functions  $G$  and  $H$  obtained from the corresponding traces. This is better understood on an example.

Consider the sequence of examples:

```
{ (A) -> (A(A)) ; (A B) -> (B A (B)) ; (A B C) -> (C B A(C)) ;
  (A B C D) -> (D C B A (D)) }.
```

The traces of  $F$  are:  $(F_1 X) = (CONS(CAR X)(CONS(CAR X) NIL) NIL)$

$(F_2 X) = (CONS(CADR X)(CONS(CAR X)(CONS(CONS(CADR X) NIL) NIL)))$

$(F_3 X) = (CONS(CADDR X)$

$(CONS(CADR X)(CONS(CAR X)(CONS(CONS(CADDR X) NIL) NIL))))$

$(F_4 X) = (CONS(CADDR X)$

$(CONS(CADDR X)$

$(CONS(CADR X)(CONS(CAR X)(CONS(CONS(CAR X) NIL) NIL))))$

The generalization of F1 and F2 leads to the substitutions : (XX\ (CONS (CAR X) NIL)) for F1 and (XX\ (CADR X)) for F2 which is equivalent to ((CONS (CAR X) NIL)\ (CADR X)). The left term is more complex than the right one, this contradicts rule 5.2.1.5.

Our system will therefore attempt to replace (CONS (CAR X) NIL) by a new variable and rewrite (F1 X) as (G1 X (H1 X)) with (G1 X XX) = (CONS (CAR X) XX) and (H1 X) = (CONS (CONS (CADR X) NIL) NIL). The traces of G1 and H1 are now treated as two separate problems. In fact the system treats this problem exactly as a human programmer : the result is the composition of a reverse function and a function that CONSES the last atom of the list to NIL.

Notice however that the system never "sees" this composition, it only happens here that the place of the lethal success coincides with the place where a human sees a composition.

From an optimization point of view it might be sometimes better to compose even when there is no such a lethal success. A system which systematically attempts to find a composition has been described by JOUANNAUD and GUIHO [7] but it is restricted to traces that have only atoms at their top level and cannot therefore be used as an optimizer.

5.6 Recursion steps greater than one.

When the preceding attempts fail, one can try to match F1 with F1+k for k=2,3,...

We have described elsewhere [12] how the polynomial behaviour of the traces might show what is the value of k to be chosen (the traces no longer increase like a polynomial but like a family of polynomials defined on a partition modulo k of the integers).

We give now an example where the form of the sequence of substitutions s1 shows how to choose the value of k.

Consider the sequence of examples given by :

```
{ (A) -> (A A) ; (A B) -> (A (A)) ; (A B C) -> ((A)(A)) ;
  (A B C D) -> ((A)((A))) ; (A B C D E) -> (((A))((A))) }.
```

After a first generalization, one gets the substitutions :

```
s1 = (XX1 \ (CONS XX1 NIL)), (XX2 \ XX2)
s2 = (XX1 \ XX1), (XX2 \ (CONS XX2 NIL))
s3 = (XX1 \ (CONS XX1 NIL)), (XX2 \ XX2) ...
```

We do not obtain constant substitutions but, if we match F1 and F1+2 we indeed have constant substitutions and one obtains the function :

```
(DE F(X)(F1 (CAR X) X))
(DE F1(XX Y)(COND
  ((ATOM(CDR Y)) (CONS XX (CONS XX NIL)))
  ((ATOM(CDDR Y)) (CONS(CONS XX NIL)(CONS XX NIL)))
  (T(F1 (CONS XX NIL) (CDDR Y)))))
```

This is not actually implemented in the present system : the user has to give separately the two sequences.

## 6. RECURSIVE TO TAIL-RECURSIVE TRANSFORMS.

We apply the BMWK algorithm to the traces obtained as described in section 3.A success shows that a tail recursive expression can be equivalent to the given recursive expression. As a matter of fact nothing proves the equivalence of the two programs and one has to device by hand an induction proof of their equivalence [13].

The differences between the synthesis method and the transformation method are simply :

- 1-The domain is not explicitly given. One has to check carefully the substitutions on the x type variables do not introduce infinite computation loops.
- 2-A lethal success introduces a composition but one cannot come back to the father CONS since we no longer have the CONS function. We have seen that a good rule is to come back to the first father which is a function of arity greater than 1. It might be possible also to give to the system the name of the wished function.

### 6.1 First example.

We start from the function : (DE F(X)(FP X X X X))  
(DE FP(X1 X2 X3 Y)(COND

```
((P Y)(H X1 (G X2 (L X3))))
(T (H X1 (G X2 (FP (M X1)(N X2)(O X3)(Q Y))))))
```

where P is a predicate and H,G,L,M,N,O,Q are supposed to be known functions with no special properties.

By the method of section 3 we get the traces :

```
IF (P X) THEN (H X(G X(L X))) ELSE
IF (P(Q(X))) THEN (H X(G X(H(M X)(G(N X)(L (O X)))))) ELSE
IF (P(Q(Q X))) THEN (H X(G X(H(M X)(G(N X)(H(M(M X))
(G(N(N X))(L(O(O X))))))))...
```

The traces lead to a lethal success which cuts the problem into two parts :

```
Problem 1 : F1=(H X (G X Z))
F2=(H X (G X (H (M X)(G (N X) Z))))
F3=(H X (G X (H (M X)(G (N X) (H (M(M X))(G (N(N X) Z))))))
```

```
Problem 2 : FP1 = (L X) FP2 = (L (O X)) FP3 = (L (O (O X)))
```

Problem 2 is readily solved by the substitution  $(X \setminus (O X))$ . Problem 1 leads to the substitutions:

```
s1 = (X \ X), (Z \ (H (M X) (G (N X) Z)))
s2 = (X \ X), (Z \ (H (M(M X)) (G (N(N X) Z))))
```

The substitution on Z introduces a new subproblem to problem 1 which is easily solved by a generalization of X into two variables X1 and X2, and the substitutions  $(X1 \setminus (M X1))$ ,  $(X2 \setminus (N X2))$ ,  $(Z \setminus Z)$ .

It follows that we obtain a new program F :

```
(DE F(X) (FP X X (FPP X X)))
(DE FPP(X Y)(COND
  ((P Y) (L X))
  (T (FPP (O X) (Q Y)))))
(DE FP(X Y Z)(COND
  ((P Y) (H X (G X Z)))
  (T (FP X (Q Y) (FPPP X Y Z)))))
(DE FPPP(X Y Z)(FPPPP X X Y Z))
(DE FPPPP(X1 X2 Y Z)(COND
  ((P(Q Y)) (H (M X1) (G (N X2) Z)))
  (T (FPPPP (M X1) (N X2) (Q Y) Z)))))
```

This example brings two comments.

First by following the computation, if H and G are strict and if the existence of an X such that  $(P X)=\text{true}$  implies the existence of an X such that  $(P(Q X))=\text{true}$  then the two  $(F X)$  are strongly equivalent.

Second it illustrates well the time-place dilemma since the original F needs a stack but has a linear complexity while the second F does not need a stack but has a complexity in X power 2.

6.2 We give an other example where the resulting program is really better than the original since it needs no stack and the complexity drops down from exponential to linear.

If we give the traces of this "Fibonacci-cross-recursive" program :

```
(DE FIBAN(N) (FIBA N N))
(DE FIBA(N1 N2)(COND
  ((ATOM N1) (FIBA1 N2))
  ((ATOM (CDR N1)) (FIBA2 N2))
  (T (FIBA (CDDR N1) N2))))
(DE FIBA1(N)(COND
  ((ATOM N) 'B1)
  ((ATOM (CDR N)) 'B2)
  (T (H (FIBA1 (CDDR N)) (FIBA2 (CDR N)))))
(DE FIBA2(N)(COND
  ((ATOM N) 'B1)
  ((ATOM (CDR N)) 'B2)
  (T (H (FIBA2 (CDDR N)) (FIBA1 (CDDDR N)))))
```

where H is any known function, then the system finds FIBAN equivalent to :

```
(DE FIBAN(N)(FIBA N 'B1 'B2)) (DE FIBA N X Y) (COND
  ((ATOM N) X)
```

```
((ATOM(CDR N)) Y)
(T (FIBA (CDDR N) (H X Y) (H Y X)))
```

## 7. CONCLUSION .

We present an implemented system (in the REDUCE S-LISP of an UNIVAC-1110) which synthesizes with no combinatory explosion a tail recursive program from an input-output sequence of examples. It is an improvement to SUMMERS' methodology [19] and to the methodologies we have already described [12,14].

The system can be used as a strong recursive to iterative program translator. It is quite different from the BURSTALL-DARLINGTON system [4] and completes it more than competes with it. On the one hand our system does not use the function properties contrary to BURSTALL and DARLINGTON system, when these properties are really needed, our system will fail to improve the program. On the contrary, it is able to improve more "difficult" expressions the properties of which are not known. On the other hand, some of the eureka needed by BURSTALL AND DARLINGTON system are automatically given by the generalizations of the traces .

## REFERENCES.

- [1] J.ARSAC La construction de programmes structurés, Paris, Dunod (1977).
- [2] A.W. BIERMAN The inference of regular LISP programs from examples, I.E.E.E. on Systems, Man and Cybernetics (1978) Vol. SMC-8, pp 585-600.
- [3] A.W. BIERMAN, D.R. SMITH The hierarchical synthesis of LISP scanning programs, Information Processing 77 (1977) pp 41-45.
- [4] R.M. BURSTALL, J. DARLINGTON A transformation system for developing recursive programs, J.ACM (1977), 24, 44-67.
- [5] P. GREUSSAY Thesis Publication LITP 78-2.
- [6] S. HARDY Synthesis of LISP functions from examples, Advance papers 4th IJCAI (1975), pp 268-273.
- [7] J.P. JOUANNAUD, G. GUIHO Inference of functions with an interactive system, Machine Intelligence 9, D. MICHE ed., (1979), pp 227-250.
- [8] J.P. JOUANNAUD, Y. KODRATOFF Quelques méthodes analytiques de synthèse automatique de programmes à partir d'exemples, Journées IRIA-SESORI Synthèse, manipulation et transformation de programmes, (1978), pp 215-239.
- [9] J.P. JOUANNAUD, Y. KODRATOFF Characterization of a class of functions synthesized from examples by a SUMMERS like method using a "BMW" matching technique, Proc IJCAI-79, pp 440-445.
- [10] E. KANT A knowledge based approach to using efficiency estimation in program synthesis, Proc. IJCAI-79, pp 457-462.
- [11] Y. KODRATOFF Choix d'un programme LISP correspondant à des exemples Congrès AFCET-IRIA Reconnaissance des formes (1978), pp 212-219.
- [12] Y. KODRATOFF A class of functions synthesized from a finite number of examples and a LISP program scheme, Int. J. of Comp. and Inf. Sci. Vol 8, no 6 (1979), pp 489-521.
- [13] Y. KODRATOFF Un algorithme pour l'obtention de formes terminales récursives à partir de traces de calcul, Actes des journées francophones : Production assistée de logiciel, Genève, pp 36-63, Janvier 1980.
- [14] Y. KODRATOFF, J. FARGUES A sane algorithm for the synthesis of LISP functions from example problems, Proc. AISB meeting, Hamburg, (1978), 169-175.
- [15] E. PAPON Unpublished "stage de DEA" Orsay (1978).
- [16] G.D. PLOTKIN A note on inductive generalization, Machine intelligence 5, (1969), pp 153-163.
- [17] J.C. REYNOLDS Transformation systems and the algebraic structure of atomic formulas, Machine intelligence 5, (1969), pp 135-151.
- [18] J.A. ROBINSON A machine oriented logic based on the resolution principle, J.ACM (1965), 12, 23-41.
- [19] P.D. SUMMERS A methodology for program construction from examples, J.ACM (1977), 24, 161-175.



MEANS-ENDS ANALYSIS AND THE  
SOLUTION OF MECHANICS PROBLEMS

by  
George F. Luger  
Associate Professor, Computer Science  
University of New Mexico  
Albuquerque, New Mexico 87313, USA

-ABSTRACT-

Several years ago David Marples (1976) proposed an algorithm for derivation of simultaneous equations in the solution of Mechanics problems. Although the original intent of this algorithm was to assist his undergraduate students at Cambridge in solving applied mathematics problems, it has also proven itself a powerful tool in the MECHO automatic problem solving system (Bundy et al, 1978, 1979). This paper will briefly discuss the Marples' algorithm and demonstrate its use with two mechanics problems. Parts of traces of four humans solving the same problems will be given. Adjustments in the MECHO program are made to show how close the Marples' algorithm can fit the data of the human subjects. Brief concluding comments are made on modelling human behavior with a rule-based language.

I. INTRODUCTION

David Marples (1976) proposed an algorithm for production of simultaneous equations in the solution of mechanics problems. This algorithm was originally introduced in his tutorial sessions at Cambridge and was intended to help the students produce a sufficient number of independent simultaneous equations to solve mechanics problems. The technique is general and goal driven. It has been adopted as part of the MECHO automatic problem solver.

In Section II, the algorithm will be explained and compared with The General Problem Solver (Newell & Simon, 1963, 1972). Two problems, a pulley problem and a distance/rate/time problem will be introduced and the MECHO solution of each of these problems presented.

In Section III, parts of protocols of four subjects will be presented, and in Section IV, logically justified adjustments will be made to the Marples' algorithm in an attempt to produce a trace similar to the human protocol. Section V will present some concluding comments.

II. THE MARPLES' ALGORITHM

The Marples' algorithm is goal driven. It takes the goal or unknown to be solved for in a problem and searches back through the givens of the problem in an attempt to find an equation solving for the unknown in terms of the givens. When this is impossible, it creates intermediate unknowns, or subgoals, that will solve for the unknown and then attempts to solve for the intermediate unknowns in terms of the givens of the problem.

## LUGER-2

To see how this might be done in the solving of applied mathematics problems, consider the usual role of equations in solving a problem: Equation  $V=U+A*T$  for constant accelerations of an object over a time period

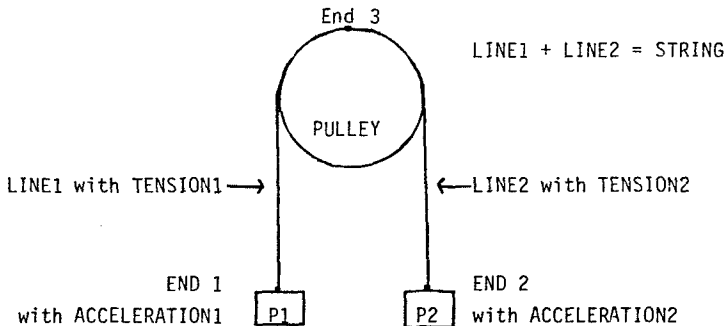
1.  $U$  is the initial velocity of the time period
2.  $V$  is the final velocity of the time period
3.  $A$  is the constant acceleration
4.  $T$  is the length of the time period

If one of  $V$ ,  $U$ ,  $A$ , or  $T$  is the unknown of the problem, the Marples' algorithm tries to assert the equation  $V=U+A*T$  by finding values in the "givens" of the problem for the other three variables. If it can only find values for one or two of the three it will then assert the equation, list the one or two values it has plus the original unknown of the problem as "given" and begin a new search for another (independent) equation with the variable it couldn't find as the unknown. And so the process continues until all the new unknowns can be determined from the givens of the problem.

The first part of the implementation of Marples' algorithm is a "focusing" algorithm that attempts to direct the Marples' algorithm to a set of equations relevant to reducing the "givens-goal difference". For example, if the final velocity  $V$  was the unknown in the problem situation above then a search would start to find out what  $V$  was: namely, the "final velocity" of a "particle or object" during a "time". When  $V$  was thus identified as the final velocity of an object during a time period, equations relevant to this situation would be identified first and a queue of these equations prepared and tested for possible given-goal reduction. Thus the Marples' algorithm would not search through all possible equations that had a  $V$  unknown but only those relevant to the particular unknown situation.

Consider now as examples of the running Marples' algorithm two problems from different areas of mechanics, a pulley and a distance/rate/time problem.

A man of 12 stone and a weight of 10 stone are connected by a light rope passing over a pulley. Find the acceleration of this man. (Palmer and Snell, 1956).



**FIGURE 1** A representation of the entities created by the pulley schemata.  $TENSION1 = TENSION2$  and  $ACCEL1 = ACCEL2$  by schema inferencing.

First, two objects, the man and a weight, are identified as connected to the rope hanging over the pulley, each object is assigned a mass and an acceleration in a direction (c.f. Figure 1). The acceleration of the man, say  $A_1$ , is identified as the sought unknown. The focussing algorithm first identifies the unknown  $A_1$  as the acceleration of the man during a time period.  $A_1$  is "bound" to the situation and a queue of possible equations examined that relate  $A_1$  to the givens. This list is examined and all equations rejected because they cannot solve for  $A_1$  without introducing new unknowns. The list is then reexamined and new unknowns allowed. The "resolution of forces" equation  $F=M*A$  (c.f. appendix) is the first in the queue and it is asserted. Originally  $A$  was unknown,  $M$  was found to be the mass of the man (known) but  $F$ , the sum of forces acting at the contact point of man and rope cannot be determined since the tension  $T_1$  in the rope is not known. Thus  $12*g+T_1=12*A_1$  is asserted  $A_1$ ,  $12$  and  $g$  are known and  $T_1$  is the new unknown. The focussing algorithm then identifies  $T_1$  as the tension in the string during the time period, a new queue of equations are proposed and the "resolution of forces" equation for the contact point of the rope and weight solves for  $T_1$  with no new unknowns.  $-10*g-T_1=10A_1$  with  $12*g+T_1=12A_1$  are seen as sufficient to solve the pulley problem.

The tower problem is slightly more complex in that four simultaneous equations are needed to solve the problem:

A particle is dropped from the top of a tower. If it takes  $t$  seconds to travel the last  $h$  feet to the ground, find the height of the tower.

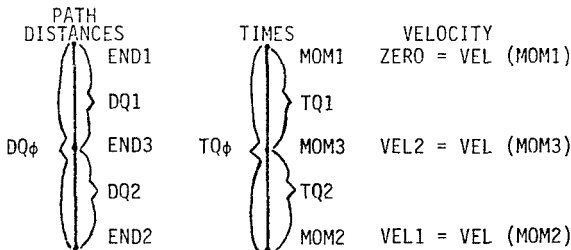


FIGURE 2 The representation of the tower problem created by the schemata.

The situation is shown in Figure 2. The total height of the tower is  $H$  and  $H=H_1+h$  where  $h$  is known. Velocity  $V_0$  is the initial velocity (0),  $V_1$  is the final velocity and  $V_2$  the velocity at the end of time  $T_1$  and the beginning of  $t$ .

$H$ , the unknown, is identified as the total length of the path of the falling object, and the Marples' algorithm examines possible equations for finding  $H$ . The equations are given in the appendix, and the progress of the Marples' algorithm through the problem is given below. When each equation is asserted a new list of givens and unknowns is prepared and the Marples' algorithm and focussing is called again (in this problem four times).

1. given ( $g, t, h$ )  
unknown ( $H$ ) - The total height

2. assert  $H=0*T+\frac{1}{2}*g*T^2$  (No. 6, appendix)  
givens (g,t,h,H); unknown (T) - The total time
3. assert  $T=T1+t$  (No. 8, appendix)  
givens (g,t,h,H,T); unknown (T1) - The time of top period
4. assert  $V2=0*g*T1$  (No. 5, appendix)  
givens (g,t,h,H,T,T1); unknowns (V2) - The velocity at midpoint
5. assert  $h=V2*t+\frac{1}{2}*g*t^2$  (No. 6, appendix)  
givens (g,t,h,H,T,T1,V2); unknowns ( )

The four equations above are seen as independent and sufficient to solve the problem.

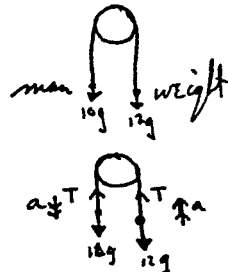
To conclude this section, the means-ends analysis implied in the Marples' algorithm may be compared to that of The General Problem Solver (Newell & Simon, 1963). The equations used by the Marples' algorithm are much like the "table of differences" used by GPS to reduce given-goal differences. The focussing technique prepares possible equations for the goal reduction just as GPS considers different given-goal combinations from the "table of differences". What is unique about MECHO, is that this is one of the first applications of means-ends analysis to solving problems in applied mathematics.

### III. FOUR PROTOCOLS OF HUMAN SUBJECTS

In this section parts of four protocols of subjects solving the pulley and tower problems are presented. The subjects were post-graduate students at the University of Edinburgh and all had had some mechanics or applied mathematics in their undergraduate education.

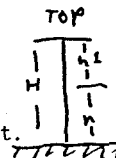
#### Protocol A (Pulley problem)

1. Mechanics problem ...
2. We'll treat the man and weight both as particles ... point masses ...
3. So man and weight ... Man has a force of 12 stone vertically downwards ...
4. Unknown at the moment ...
5. and assuming this frictionless pulley, T is the same on both sides ...
6. We'll give the rope an acceleration ... vertically down on the man's side ...
7. and vertically up on the weight's side ...
8. So resolving vertically down for the man:  $12g - T = 12a$
9. The vertically downwards acceleration
10. And the vertically upwards for the weight  $T - 10g = 10a$
11. so the thing requires the acceleration of the man which is  $\frac{a}{...}$  ...
12. So we just eliminate T from these two equations ... etc.

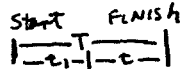


#### Protocol B (Tower problem)

1. There is a tower, suppose it has height H
2. H is made up of h, the given height
3. and h1, the unknown portion of the tower
4. We also know time t
5. Call the total time of falling T, T is made up of t plus t1 where t1 is the time for the top part.

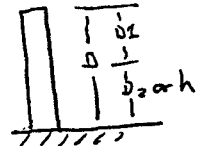


6. I need H and I'm given t and h and g the acceleration of the object
7. I know that  $H = h_1 + h$
8. Now to get  $h_1$ , a distance
9.  $h_1$  equals one half acceleration times  $t_1$  squared  $h_1 = \frac{1}{2} g t_1^2$
10. and I have an equation with  $t_1$  already  $T = t_1 + t$
11. but I still need to find T, the total time ...
12. Now, for the whole period,  $H = \frac{1}{2} g \times t^2$
13. Reviewing, I have one, two, three, four equations (indicates each)
14. and H, T,  $h_1$  and  $t_1$  are unknowns that should do it ...



#### Protocol C (Tower problem)

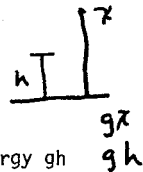
1.  $T_1$  is the time the ball crosses  $D_1$
2. and  $T_2$  the time for  $D_2$ , but  $T_2$  is  $t$  ...
3. Now I want D, knowing g is the acceleration of the ball  $D = \frac{1}{2} g T^2$
4. and I know T:  $T = T_1 + t$
5. So now I need  $T_1$  ... in the top time period
6. The final velocity ... Call it  $V_M$  ...
7. is acceleration times time  $V_M = g \times T_1$
8. and now to get  $V_M$  ...
9. The total distance D ... That won't help ...
10. The distance h and velocities ...
11. if  $V_F$  is final velocity  $V_F = V_M + g \times t$
12. and I can easily get the final velocity  $V_F$ :
13. Because I've already got T ...



time  $D \rightarrow T$   
 $D_1 \rightarrow T_1$   
 $D_2 \rightarrow T_2$   
 $or t$

#### Protocol D (Tower problem)

1. Do this by energy ...
2. It initially starts with potential energy  $mg$
3. where m is the mass of the body m is a constant
4. we can forget about m, call it 1
5.  $gx$  is the initial potential energy
6. when it reaches height h, it will have potential energy  $gh$
7. and it will have kinetic energy  $gx$
8. so it will have  $gx - hx = \frac{1}{2} v^2$
9. where v is its velocity at height h
10. we don't know what height ... what speed it is once it hits the ground
11. we want to use one of the other constant acceleration ones  $v^2 = u^2 + 2 S$
12. we want to ... an equation relating height, velocity and time, that is
13. under constant acceleration  $ah$  which is  $S = vt + \frac{1}{2} at^2$



#### IV. FOCUSING

In Section II, the Marples' algorithm for MECHO's goal driven search was presented and compared to GPS means-ends analysis (Newell & Simon, 1963). MECHO's set of possible equations served as a table of connections for goal reduction and the focussing mechanism prepared a queue of possible equations to make the connections.

In the Pulley problem above the focussing produced immediate results. In fact MECHO, like the human solver performed very little search in coming to the Resolution of Forces formula. Two applications of this formula and the problem is solved. The Tower problem was much more interesting. In fact, there are several different sets of simultaneous equations (and of course permutations within each set) that will solve the problem. Each solver (B,C,D) produced a different set of equations and each of these sets differed from MECHO's trace of the previous section.

In this section the focussing technique and table of connections is altered in an attempt to produce traces similar to the human protocols of the tower problem. First note what will not be changed. The semantic knowledge for the Tower problem will not be changed. That is, facts such as the unknown H the height of the tower and t and h the givens will be used without change by MECHO throughout this section; similarly V2 will remain the velocity at the "midpoint". Furthermore, the first part of focussing that binds situation variables for sought unknowns will remain unchanged. That is, H will be bound to the LENGTH of the PATH during the TIME. What will change is the queue of possible equation instantiations for situations and alterations will be made in the equations within the table of connections. These latter will be seen shortly.

The 10 clauses below are used to form the queue of formulae to be attempted in any problem situation. Resolve, relative velocity, etc. refer to the equation names of the Appendix.

1. relates(resolve; FORCE, ACCELERATION, MASS).
2. relates(relative velocity; VELOCITY).
3. relates(relative acceleration; ACCELERATION).
4. relates(constant acceleration-1; ACCELERATION, VELOCITY, DURATION).
5. relates(constant acceleration-2; ACCELERATION, LENGTH, VELOCITY, DURATION).
6. relates(constant acceleration-3; VELOCITY, LENGTH, DURATION).
7. relates(average velocity; VELOCITY, LENGTH, DURATION).
8. relates(constant velocity; VELOCITY, LENGTH, DURATION).
9. relates(lengthsum; LENGTH).
10. relates(timesum; DURATION).

After H is recognized as the LENGTH of the path during the episode by the first step in focussing, the second step prepares the queue by scanning the 10 clauses above to find which formulae will help solve for LENGTH. This proposes a queue of constant acceleration-2, constant acceleration-3, average velocity, and length sum. These equations are tried in that order. Each, of course, fails because new unknowns are introduced. On the second pass, when new unknowns are allowed, constant acceleration-2 is accepted. This process continues until the problem is solved.

We hypothesize that the human subject has a stack of equations that relate to specific situations. These equations are employed when attempting to reduce the given-goal differences in a specific problem. In fact, the stack might be quite similar to that produced by the 10 clauses above. This clause queue like the table of connections need not contain the full equations, only their names. These names may serve to reference the actual equation formulae which are stored with their full sets of conditions for instantiation.

This conjectured implementation of the GPS model may be tested by making simple alterations in the 10 clauses above to see if the different queue of equations to be formulated can produce a different set of simultaneous equations. In particular, we attempt to produce traces similar to the human protocols B,C, and D.

Subject B used the length sum equation with top priority when solving for LENGTH, and timesum when a DURATION was sought. Thus, if clause 9 and 10 are placed before the constant acceleration clauses the order of "relates" above becomes 1, 2, 3, 9, 10, 4, 5, 6, 7, 8. When this rearrangement of clauses was run in MECHO the following trace occurred:

- B'
1. Attempting to solve for H in terms of g,h,t.
  2.  $H=H1+h$  solves for H but introduces H1.  
\*\* H1 is a LENGTH, lengthsum cannot be used again, so constant acceleration-2 is used\*\*
  3.  $H1=0 \times T1 + \frac{1}{2} \times g \times T1^2$  solves for H1 but introduces T1
  4.  $T=T1+t$  solves for T1 but introduces T  
\*\* constant acceleration-2 is now the first on the queue for T since timesum may not be used again, and MECHO always tries to solve without further introduction of unknowns\*\*
  5.  $H=0 \times T + \frac{1}{2} \times g \times T^2$
  6. Equations 2-5 solve the Tower problem.

If the ZERO term (initial vel. x time) is removed from 3 and 5 these equations are exactly those produced by subject B above.

In an attempt to produce a trace similar to protocol C, the "relates" clause (9) for lengthsum is returned to its original position, (i.e., 1, 2, 3, 10, 4, 5, 6, 7, 8, 9). The subject of protocol C does not use the constant acceleration-2 equation to its full potential, that is, he only uses the equation when the initial velocity is zero. Marples (1976) comments on this use of equations by engineering students when he notes they often apply an equation without knowing its full power. In this instance, the subject uses constant acceleration-2 for relating acceleration, time, and distance and not in its full use of relating initial velocity, acceleration, time, and distance. If it is conjectured that this happens with subject C, constant acceleration-2 equation is changed to this limited used by rewriting 5 above to "relates (constant acceleration-2; ACCELERATION,LENGTH,DURATION)" and removing "U\*T" from the isformula clause of constant acceleration-2 c.f. appendix

MECHO is now run with these changes and the following trace results:

- C'
1. trying to solve for H in terms of g,t,h.
  2.  $H=\frac{1}{2} \times g \times T^2$  solves for H but introduces T.
  3.  $T=T1+t$  solves for T but introduces T1.
  4.  $VEL2 = ZERO + g \times T1$  solves for T1 but introduces VEL2.  
\*\* Recall that the final velocity at the bottom is VEL1 and the velocity at the "midpoint" VEL2. The solver using constant acceleration-2 properly would now be done. Our subject grinds on\*\*
  5.  $VEL1 = VEL2 + g \times t$  solves for VEL2 but introduces VEL1.
  6.  $VEL1 = ZERO + g \times T$  solves for VEL1.
  7. Equations 2-6 solve the Tower problem.

This trace is remarkably similar to protocol C.

The subject of protocol D decided to use energy equations as was explicitly stated. This was not expected by the investigator taking protocols or designing MECHO to solve distance/rate/time problems. But in principle

there was no reason why energy equations could not be used. They were, in fact, already in the system and used to solve de Kleer's problems (Bundy, 1978). A new entry for the "relates" table was constructed: No. 11. relates (conserve energy; VELOCITY,LENGTH) and this was given priority over all other LENGTH relation clauses. Further, constant acceleration-3 is equivalent to conserve energy and was removed. Finally, subject D favored constant acceleration-2 over constant acceleration-1 formula for solving VELOCITY problems, so this order was changed. The "relates" list was 1, 2, 3, 4, 10, 11, 6, 8, 9, 5.

MECHO was run in this situation; its trace:

- D'
1. Trying to solve for H in terms of g,t,h.
  2.  $\frac{1}{2} * VEL1^2 - \frac{1}{2} * ZERO^2 = g * H$  solves for H but introduces VEL1
  - \*\* The energy equation is attempted again. This time to solve for VEL1\*\*
  3.  $\frac{1}{2} * VEL1^2 - \frac{1}{2} * VEL2^2 = g * t$  solves for VEL1 but introduces VEL2
  4.  $h = VEL2 * t + \frac{1}{2} * g * t^2$  solves for VEL2
  5. 2-4 solve the Tower problem.

It can be seen that this trace is very close to the protocol of subject D above. Further comments will be made in the next section.

#### V. SUMMARY AND CONCLUSIONS

The goal of this paper has been to describe the action of the Marples' algorithm in MECHO's solution of pulley and distance/rate/time problems. After creation of a knowledge base, the Marples' algorithm was invoked and using means-ends analysis, in many ways similar to GPS, produced sets of simultaneous equations sufficient for solving the problem.

Experienced human subjects solving the same problems were presented with strategies for producing sets of simultaneous equations sufficient to solve a problem in many ways similar to those of the Marples' algorithm. Finally, with slight changes the Marples' algorithm could produce sets of equations almost identical to those produced by the human subjects.

In the pulley problem the protocol indicates the subject goes immediately to the resolution of forces equation. This is used to create the intermediate unknown of the tension in the string. Forces are again resolved at the other end of the string to solve for tension and two simultaneous equations are produced sufficient to solve the problem. The Marples' algorithm in MECHO proceeds in exactly the same fashion with the important difference that the resolution-of-forces equation, the first equation considered, is rejected because it introduces a new unknown (tension). All other equations in the queue trying to find acceleration for a particle in a time period are examined and rejected before the return to the resolution-of-forces equation and introduction of the tension as a new unknown. The difference between the experienced human and MECHO is obvious and interesting. The human realizes immediately a new unknown must be introduced, accepts it, and goes on, while MECHO tries as long as possible to avoid this course of action. Other than this, the protocol of the human and MECHO's trace are remarkably similar.

The similarities between trace and protocol are even stronger with the tower problem where many different sets of simultaneous equations sufficient for solving the tower problem may be produced. The formation of any set of these equations depends entirely on the use of the focussing technique, that is, the queue that is generated for possible equations. The similarity of



protocol C with the trace of C' of Section IV is striking. Indeed, when it was hypothesized that if the human subject had been able to use initial velocity values in constant acceleration-2 equations when the initial velocity was not zero and had a slightly altered focus, then MECHO's trace would match exactly the protocol of subject C. In section IV, these alterations were made and the resulting protocols compared. The results indicate the robustness of the Marples' algorithm and focussing to generate different sets of simultaneous equations to fit closely the traces of the human subjects.

One of the principal advantages of writing the MECHO problem solver in PROLOG (Warren, 1978) here represented by predicate logic assertions, is that PROLOG actually computes using the predicate logic statements themselves. In fact, PROLOG was designed as a predicate logic theorem prover. Thus facts, inferences, and default values are entered into the program as the potential for "meaningful bits of behavior". This allows such actions as removing a rule from the program, substituting another rule, or simply changing the order of the rules and then checking the results of these changes on the running computer program. Thus a PROLOG "fact", "inference rule", or "default assignment" may be paired with the corresponding competency in the human subject and the effect of its presence or absence in the human subject may be simulated by the running program. This can be seen when the "conservation of energy" equation was added for solution of the Tower problem (IV). The new rule added resulted in the exhibition of a new competency, and conversely, the absence of the rule marked the absence of the related ability.

A modular set of rules also allows general purpose algorithms, such as the Marples' algorithm, to be implemented and the effects of the presence of the algorithm to be seen by running the program. In a very similar fashion Larkin (1978) and Simon and Simon (1977) can run sets of production rules in an attempt to simulate the difference of skills in the expert or novice problem solver.

The presence of production or behavior rules also provides a model for the interpretation of missing or ambiguous behavior of the human subject. In D, for example, "Do this by energy" indicates an energy equation will be called to find the value for H. D5 states "gx is the initial potential energy" ... and D 6-7 "when it reaches height h it will have potential energy gh ..." and finally in D 8-9 "so it will have  $gx - gh = \frac{1}{2}V^2$  where V is the velocity at height h ..." - what does this all mean? The protocol D2 to D10 is at best confusing. Reading MECHO's trace on the same problem goes a long way towards making sense of these statements. D'2 gives a full description of gx: " $g * H = \frac{1}{2} VEL1^2$ ", where  $g * H$  is gx and VEL1 the final velocity. Similarly for gh: D'2 has " $g * h + \frac{1}{2} VEL1^2 - \frac{1}{2} VEL2^2$ ", when VEL2 is the velocity at the top of h. Now simply subtract D'3 from D'2 and line D9 of the protocol is precisely understood. The subject of protocol D was a particularly bright individual that liked to skip steps and simplify as he went along. Although MECHO is not able to imitate this behavior completely - especially the subject's proclivity for short slightly ambiguous statements - its rule system does give a precise and complete performance and often provides data sufficient to disambiguate the human subject's behavior.

There are, of course, many things that MECHO, as presently designed, does not do that human subjects do quite easily. We have already seen how sub-

ject D simplified as he went along and omitted "unnecessary" bits of equations. Similarly, other subjects - this could be easily added to MECHO- left out terms of equations that were zero. Also MECHO is exhaustively thorough in its search while human subjects are not, and this MECHO will never use five equations where four are sufficient. However, as noted in Section IV, MECHO can offer an explanation of precisely why a subject needed five equations when four would have been sufficient.

### References

1. Bundy, A. Will it Reach the Top? Prediction in the Mechanics World. Artificial Intelligence (1978), 10, 111-22.
2. Bundy, A., Luger, G., Mellish, C. & Palmer, M. Knowledge about Knowledge: Making decisions in mechanics problem solving. Proceedings of AISB Conference - 1978.
3. Ernst, G. & Newell, A. GPS: A Case Study in Generality and Problem Solving. New York: Academic Press, 1969.
4. Marples, D. Argument and Technique in the Solution of Problems in Mechanics and Electricity, Cambridge (UK) CUED/C, EDUC/TRI, 1974.
5. Newell, A. and Simon, H. GPS: A Program that Simulates Human Thought. In Computers and Thought, Feigenbaum & Feldman (ED.), New York: McGraw Hill, 1963.
6. Newell, A. & Simon, H. Human Problem Solving, Englewood-Cliffs, N.J.: Prentice-Hall, 1972.
7. Larkin, J.H. Skill Acquisition for Solving Physics Problems. C.I.P. Report #409, Dept. of Psychology, Pittsburgh: Carnegie-Mellon University, 1978.
8. Palmer, A. & Snell, K. Mechanics. London: University of London Press, 1956.
9. Polya, G. How to solve it. Princeton: Princeton University Press, 1945.
10. Simon, H. and Simon, D. Individual Differences in Solving Algebra Problems, CIS #342, Dept. of Psychology, Pittsburgh: Carnegie-Mellon University, 1977.
11. Warren, D. Implementing PROLOG - compiling predicate logic programs. DAI Tech. Report. University of Edinburgh, 1978.

I would like to thank Alan Bundy, Lawrence Byrd of the MECHO group and Richard Young of APU - Cambridge for encouragement in writing this paper and criticism of earlier drafts and the British SRC for supporting this research.

Appendix: PROLOG clauses for possible equation formation in Pulley and Tower problems

Note that words with first letter capitalized, e.g., Object or Direction, as well as single capitol letters, e.g. M or T1, represent variables.

1. isformula ( $F = M * A$ , resolve (Object, Direction, Time)):- mass (Object, A, Direction, Time), acceleration (Object, A, Direction, Time), sumforces (Object, Direction, Time, F).
2. isformula ( $V13 = V123$ , relative velocity (Obj1, Obj2, Obj3, Time)):- relative velocity (Obj1, Obj2, V12, Dir12, Time), relative velocity (Obj1, Obj3, V13, Dir13, Time), vectoradd (V12, Dir12, V23, Dir23, V123, Dir13).
3. isformula ( $A123 = A123$ , relative acceleration (Obj1, Obj2, Obj3, Time)):- relative acceleration (Obj1, Obj2, A12, Dir12, Time), relative acceleration (Obj2, Obj3, A23, Dir23, Time), relative acceleration (Obj1, Obj3, A13, Dir13, Time), vectoradd (A12, Dir12, A23, Dir23, A123, Dir13).
4. isformula ( $S = V * T$ , constant velocity (Object, Time)):- constant velocity (Object, Time), velocity (Object, V, Direction, Time), duration (Time, T), distance (Object, S, Time).
5. isformula ( $V = U + (A * T)$ , constant acceleration1 (Object, Time)):- constant acceleration (Object, Time), duration (Time, T), initial velocity (Object, U, Direction, Time), final velocity (Object, V, Direction, Time).
6. isformula ( $S = U * T + (A * (T:2)/2)$ , constant acceleration2 (Object, Time)):- constant acceleration (Object, Time), acceleration (Object, A, Direction, Time), duration (Time, T), initial velocity (Object, U, Direction, Time), distance (Object, S, Time).
7. isformula ( $((V:2) + (U:2) = 2 * A * S$ , constant acceleration3 (object, Time)):- constant acceleration (Object, A, Direction, Time), distance (Object, S, Time), initial velocity (Object, U, Direction, Time), final velocity (Object, V, Direction, Time).
8. isformula ( $T = \text{Sum}$ , timesum (Time)):- Partition (Time, Points), duration (Time, T), sumdurations (Points, Sum).
9. isformula ( $D = \text{Sum}$ , lengthsum (Path, Time)):- partition (Path, D, Time), sumlength (Points, Sum, Time).
10. isformula ( $(V:2)/2 - (U:2)/2 = g * H$ , energy (Object, Time)):- motion (Object, Path, Start, Side, Time), incline (Path, 270, Time), length (Path, H, Time), final velocity (Object, V, Direction, Time), initial velocity (Object, U, Direction, Time).

\*\* g is known by MECHO as the gravitational constant \*\*

COMPUTATIONAL AND PSYCHOPHYSICAL STUDIES TOWARDS A THEORY OF HUMAN STEREOPSIS

John E.W. Mayhew and John P. Frisby  
 Department of Psychology,  
 University of Sheffield,  
 Sheffield, England, S10 2TN

ABSTRACT

Psychophysical studies are described which pose a strong challenge to models of human stereopsis based on the processing of disparity information within independent spatial frequency tuned binocular channels. These studies support instead the proposal that the processes of human binocular combination integrally relate the extraction of disparity information with the construction of raw primal sketch assertions. This proposal implies binocular combination rules using principles of figural continuity and cross-channel correspondences to disambiguate at a global level matches found independently within spatial frequency channels at a local level. Computer implementations of stereo algorithms based on these rules are described and found to be successful in dealing with a variety of different stereo inputs. The constraints presented by objects which are exploited by these algorithms are discussed. The paper is an abbreviated version of a paper to appear in a special issue of *Artificial Intelligence* devoted to vision (Mayhew and Frisby, 1980c).

1 Introduction

Ever since the introduction by Julesz (1960) of the random dot stereogram as a research tool, it has become commonplace to consider the theoretical problems surrounding stereopsis within a conceptual framework that has two distinctive characteristics:

First, it is recognised that disparity information can be extracted using only low-level monocular 'point' descriptions as the entities which are binocularly matched. The random dot stereogram is a clear demonstration that high-level monocular descriptions, such as those dealing with surfaces and objects, are not a necessary requirement for stereopsis because for a random dot stereogram these relatively high level scene descriptions appear only after stereopsis has been achieved.

Secondly, it has become acknowledged that there is a need for mechanisms capable of selecting the correct binocular point-for-point matches from amongst the multitude which are frequently possible, most of which are false matches or 'ghosts'. That is, in the terminology of Julesz, it is necessary to posit 'global stereopsis' mechanisms to resolve ambiguity often existing between competing 'local stereopsis' matches.

The clarity of this conceptual framework has spawned a number of stereo algorithms, of varying type and capability (see Marr and Poggio (1977) for a review). Given the framework, each algorithm naturally has to face (at least) two critical design choices, namely what monocular point descriptions are to be used, and what global mechanisms are to be employed to resolve the ambiguity inherent within the population of possible local matches. Marr and Poggio (1976) have pointed out that any worthwhile algorithm must avoid the trap of ad hoc answers suitable for one type of image but not for others. They have argued, in the manner characteristic of the M.I.T. computational

approach to studying visual mechanisms, that they way to do this is to begin with a precise formulation of the goals of the visual task being considered (in this case the exact processing goals associated with stereopsis). The next step they advocate is to develop a computational theory specifying useful constraints about how objects behave in the world, constraints which enable valid processing rules to be formulated for use in an algorithm capable of achieving the goals in question. We are in great sympathy with this approach and much of what we discuss in this paper falls naturally within its scope. Nevertheless, the particular model of human stereopsis which we describe owes at least as much to the results of our psychophysical work on stereopsis as it does to computational considerations - hence the title of this paper.

## 2 The BRPS Conjecture

Over the past few years, we have conducted a series of psychophysical studies of human stereopsis which have led us to the following conjecture:

*THE BRPS CONJECTURE: The processes of human binocular combination integrally relate the extraction of disparity information with the construction of raw primal sketch assertions.*

We call this conjecture the BRPS conjecture because it amounts to the proposal that one goal of early visual processing is the computation of a Binocular Raw Primal Sketch. In this paper we discuss our psychophysical justifications for advancing the conjecture and we present various computational studies related to it. We also discuss how the conjecture relates to the models of stereopsis advanced by Marr and Poggio (1976, 1979). Before proceeding in detail to these various tasks, however, several preliminary comments about the conjecture are in order:

(a) It is worth noting that a 'raw primal sketch' is defined by Marr (1976) as a description of intensity changes in an image, using a primitive language of edge-segments, bars, blobs and terminations. The selection, grouping and summarising of these raw primitives leads to larger and more abstract tokens in a description that Marr calls the 'full primal sketch'.

(b) The most general computational implication of the conjecture is that the constraints and correspondences between spatial frequency tuned channel outputs that can be used to compute monocular descriptions of intensity changes (Marr, 1976; Marr and Hildreth, 1979) together with the grouping principle of figural continuity, can be applied to advantage at various stages during the process of binocular combination. The final result is a set of binocular intensity change descriptions to which are tied disparity assignments.

Marr's general scheme for obtaining raw primal sketch assertions from channel outputs has two major steps: (i) certain very low level descriptive elements (e.g. zero crossings) are extracted from each channel's output independently: we will call these descriptive elements 'measurement primitives' to distinguish them from raw primal sketch primitives; and (ii) measurement primitives from all the various channels are submitted to a set of cross-channel 'parsing' rules which result in the required raw primal sketch assertions. The BRPS conjecture proposes that human binocular combination takes place during both these steps, with initial binocular matches effected at the level of measurement primitives being disambiguated during the

application of binocular combination rules whose ultimate objective is the delivery of binocular raw primal sketch assertions. The need for disambiguation stems from the fact that a large population of binocular measurement primitive matches are frequently possible, with only a certain number of these being correct matches and the remainder being false matches (see earlier remarks on local vs. global stereopsis).

(c) Alternative processes of binocular combination to those specified in the conjecture can in principle be envisaged, so that the conjecture does not embrace all possible models of human stereopsis. For example, it could be that human binocular combination proceeds in a way which takes no advantage of cross-channel correspondences existing between spatial frequency channel outputs, as for example in Marr and Poggio's (1979) stereo algorithm in which disambiguation takes place independently within each channel (except for some coupling via vergence control: see Section 5.3). Also, it is possible to envisage models of stereopsis in which binocular combination proceeds after the stage at which raw primal sketch elements are constructed, models which are nevertheless 'low-level' in general terms. For example, Marr and Poggio (1976) suggested that their cooperative disparity-processing network could be fed with fully-parsed raw primal sketch elements, the network then serving to disambiguate competing element-for-element matches at a stage much later than that proposed for the initiation of disambiguation as far as the BRPS conjecture is concerned.

(d) The stereo models of Marr and Poggio just referred to are successful in dealing with a number of different types of stereo inputs, including both random-dot and natural scene stereograms. Moreover, the success of these models is based on an analysis of constraints offered by objects in the world, and on how rules for binocular combination stemming from these constraints can be implemented in a stereo algorithm which exploits the constraints to attain correct stereo fusion. The question of what constraints are available is, we recognise, a very important one and we will return frequently in this paper to the constraints on which our own conceptions of human stereopsis processing might be based.

### 3 Choice of Monocular Measurement Primitives

#### 3.1 Zero Crossings

For the various elegant computational reasons advanced by Marr and Hildreth (1979), zero crossings discovered in spatial frequency tuned convolutions seem a good starting point for the computation of a monocular raw primal sketch. Therefore, given the BRPS conjecture, it is natural to use zero crossings extracted from left and right image convolutions as at least one of the set of measurement primitives used for constructing a binocular raw primal sketch. Moreover, this conjecture implies that only left/right zero crossings of the same contrast sign and roughly similar orientation should be binocularly matched.

#### 3.2 Peaks

Despite some theoretical advantages possessed by zero crossings as measurement primitives (see Marr, Ullman and Poggio (1980) for a discussion of the possible relevance of Logan's theorem in this context), it seems to us that in some situations zero crossings are poor measurement primitives in principle for extracting disparity information. Thus the strategy of using solely zero

crossings will fail for those parts of an image where disparity variations are tied to luminance variations situated in between the parts of the image which produce zero crossings. We have devised a stereo pair illustrating such a case (Mayhew and Frisby, 1980c) in which the left and right images are sawtooth luminance profiles with the same period but with slightly differing shapes. Fusion of this stereogram produces a percept of a surface undulating in depth where the peaks and troughs in the luminance profile correspond to the peaks and troughs in the depth profile. If disparity processing was based solely on zero crossings derived from the stereo images, then the predicted perceived depth would be of a flat (or planar tilted) surface. This follows trivially from the fact that a zero crossing must reflect a local luminance average and if luminance is confounded with disparity, as in our sawtooth stereogram, then a zero crossing can convey only a local average disparity. Therefore, the fact that the human visual system sees an undulating surface in this stereogram suggests that it is not relying simply on the zero crossings but that it also utilises (at least) information carried by the peaks and troughs of the convolution profiles as well.

It is worth noting in this connection that Mayhew and Frisby (1979) and Frisby and Mayhew (1980) argued for peaks being used as measurement primitives for different reasons than those just described. They observed that zero crossings for edges exhibit spatial coincidences which are highly convenient for parsing purposes (Marr and Hildreth, 1979) but that zero crossings from lines and bars do not. For the latter, it is the peaks which show the desired spatial coincidence (see later for a fuller discussion of why spatial coincidence is valuable). This consideration provides another argument for utilising peak measurement primitives in addition to zero crossings. (The difference between peaks and zero crossings situation for bars and edges follows from the fact that the fourier components for a bar are in cosine phase whereas those for an edge are in sine phase.)

### 3.3 Orientated or Circularly Symmetric Convolutions?

Should measurement primitives be extracted from orientated or circularly symmetric convolutions? Marr and Poggio (1979) chose the orientated option although in an implementation of the key features of their stereo algorithm, Grimson and Marr (1979) substituted circular for orientated convolutions. A full theoretical debate of the computational considerations surrounding the choice between these alternatives is provided in Marr and Hildreth (1979).

Interestingly, we have arrived independently at a rejection of the orientated option (Mayhew and Frisby, 1979; Frisby and Mayhew, 1980) but mainly for psychophysical rather than for computational reasons (although as far as the latter are concerned, we noted the expensive, indeed almost wanton, use of storage implied by holding in memory many different spatial frequency and orientationally tuned convolutions, and also some of the unnecessary and rather ad hoc computational procedures required to obtain a primal sketch from orientated convolutions which inevitably smear image features along the axis of orientational tuning). Our psychophysical studies telling against orientated convolutions for stereopsis are twofold:

(a) We have discovered that if stereopsis from an orientated texture stereogram is masked by adding similarly orientated noise to one field

then rotation of the masking noise so that it would no longer perturb an orientated convolution carrying the disparity signals does not succeed in releasing the stereopsis from the effects of the mask (Mayhew and Frisby, 1978a). This is difficult to understand if local measurement primitives are extracted from orientated convolutions.

(b) We have pointed out that orientated spatial frequency tuned filters are in principle poor devices for dealing with rapidly changing disparity cues (Mayhew and Frisby, 1979). We have demonstrated their difficulties in this regard using a stereogram portraying a series of horizontal corrugations, as though the observer was looking down on a corrugated roof. Vertically-tuned filters could not in principle extract the depth from this stereogram: they would inevitably have receptive fields spreading over several 'disparity rasters' and so smear hopelessly the disparity cues contained in the stereogram. Predictably therefore, a vertically filtered version of the stereogram cannot be fused to reveal the corrugations. It might be thought that the easily-obtained stereopsis from the original could be mediated by horizontally-tuned units, but the poor quality of the stereopsis deriving from a horizontally-filtered version suggests otherwise, as does the fact that many naive subjects cannot obtain any depth whatsoever from the horizontally-filtered version whereas they can do so easily for the unfiltered original. (Note that there are also limits on the disparity gradients which can be extracted using circular convolutions but these are less restrictive than those for orientated convolutions: Mayhew et al, 1980).

We conclude from these two studies that orientated spatial frequency tuned convolutions do not seem to be used in the human visual system as a basis for establishing local disparity matches, at any rate not exclusively. Of course, this conclusion does not preclude other types of orientational selectivity embedded in the stereopsis mechanism, both at the local and global levels. Orientation is tied to zero crossings at the local level, and orientation can be made use of at the global level by a disambiguating algorithm which incorporates a principle of figural continuity (see next section).

#### 4 Binocular Combination Rule 1: Figural Continuity

If an ambiguity in left/right zero crossing matches arises, those matches which preserve figural continuity are to be preferred, given the BRPS conjecture. This is because figural continuity is an important principle utilised in obtaining raw primal sketch assertions. Thus the binocular deployment of figural continuity is one way in which we utilise the overall objective of obtaining binocular raw primal sketch assertions to constrain and direct the flow of support between local measurement primitive matches by way of achieving global disambiguation. (Note that by their very nature zero crossings must be figurally continuous for they are the points of intersection of a plane with a surface, i.e. they are the points where the DC plane intersects with the 2D convolution surface.)

##### 4.1 A Binocular Combination Algorithm Using Figural Continuity

A stereo algorithm called STEREOEDGE which exploits figural continuity has been written (Mayhew and Frisby, 1980a; Frisby and Mayhew, 1980) and it demonstrates that curvilinear grouping rules can successfully disambiguate zero crossing and peak matches in both natural and random dot stereograms. The algorithm returns edge-segment and angle assertions to which are tied disparity assignments.



Figural continuity is implemented in STEREOEDGE as the piece-wise local binocular grouping of adjacent peaks or zero crossing matches of the same contrast sign. Grouping across small gaps in peak or zero crossing continuity (e.g. those caused by quantisation fuzz) may be admitted if disparity continuity is not grossly violated, if overall orientation is preserved, and of course if no better grouping is available. Disambiguation by the colinear grouping of primitives of the same contrast sign is natural given the overall objective of arriving at binocular raw edge segment assertions. If two connected binocular edge-segments of different orientations are present, then STEREOEDGE returns an angle assertion for the appropriate location. Thus the type of figural grouping employed is similar to the process of curvilinear aggregation described by Marr (1976) in connection with the primal sketch.

#### 4.2 Matter is Cohesive: A Constraint About The World Justifying The Rule of Figural Continuity

It is easy to justify the use of binocular figural grouping principles in terms of constraints about the world upon which the processes of binocular combination must rely. As Marr and Poggio (1976) pointed out in their analysis of the stereo disparity computational problem, "matter is cohesive, it is separated into objects". It is a simple consequence of this fact that the edges of surfaces, and also surface markings such as lines and blobs, will be spatially continuous and that therefore their corresponding intensity changes in the retinal images will also be continuous. This is the ultimate justification for relying on figural continuity grouping rules to guide binocular combination.

Note, however, that our use of this constraint about the world is importantly different from that of Marr and Poggio (1976). Unlike us, they use the constraint to justify a rule of preferring matches which are disparity-continuous. This leads them to a cooperative algorithm which relies on lateral excitation between similar disparity matches. Their later algorithm, (Marr and Poggio, 1979) also has similar albeit less extensive disparity facilitation. For us on the other hand, the pursuit of raw-primal-sketch-type binocular figural descriptions is the overall objective and this does not necessarily demand the explicit use of similar-disparity information to guide the selection of left/right matches. In STEREOEDGE, for example, left/right zero crossing matches can be chosen according to the principle of figural continuity alone, without any explicit reference to the disparity continuity of these matches, and with the actual disparity value of selected matches accessed only after binocular combination has taken place.

#### 4.3 Psychophysical Evidence Supporting The Rule of Figural Continuity

Psychophysical evidence for supposing that figural continuity provides a guiding role in human stereopsis is described in the full version of this paper (Mayhew and Frisby, 1980c). In general, stereograms containing textures which admit of figural grouping are much easier to fuse even though they may have a greater ambiguity problem in terms of potential false matches.

### 5 Binocular Combination Rule 2: Correspondences Between Channel Outputs

#### 5.1. The Interpretation of Cross-Channel Correspondences for Constructing Raw Primal Sketch Assertions

Marr (1976) and Marr and Hildreth (1979) have demonstrated how various

correspondences between spatial frequency channel outputs can be utilised to compute the figural type of raw primal sketch assertions (e.g. whether an edge, line or blob is present and what contrast and width it possesses). The BRPS conjecture holds that these correspondences are used during human binocular combination, so that the patterns of between-channel correspondence which enable figural type to be asserted also help disambiguate within-channel fusions.

There are two important forms of cross-channel correspondence to which Marr (1976) and Marr and Hildreth (1979) draw attention: spatial coincidence and spectral continuity.

As far as spatial coincidence is concerned, Marr and Hildreth have shown that if zero crossings are present in two or more spatial frequency channels in the same relative locations, then this 'spatial coincidence' is clear-cut evidence that an edge is present in the image in the appropriate position. Consequently, they have argued that spatially coincident zero crossings provide excellent points at which to interpret channel outputs by way of arriving at assertions about the particular type of edge present. We have added to this the consideration that peaks show equally convenient spatial coincidence when bars are present, whereas zero crossings tend to spread out for such inputs (see Section 3.2).

Turning to the question of spectral continuity, Marr (1976) pointed out that it is both possible and desirable to use a rule of spectral continuity to guide the preliminary selection of channel measurements prior to their interpretation as a descriptive element. Thus if the distribution of channel outputs is unimodal, a single descriptive element is parsed; on the other hand, if there is a split into two groups then the assumption is made that each group comes from a different entity in the image and a separate description is obtained for each group, so that for example a sharp line can be parsed as 'sitting on top of' a blurred blob.

## 5.2 Spatial Uniqueness: A Constraint About The World Justifying The Binocular Use of Correspondences Between Channels

It is possible to justify our binocular deployment of cross-channel correspondences in terms of the constraints imposed by objects in the world. Here the relevant constraint is similar to Constraint 1 of Marr and Poggio (1976), namely "that a given point on a physical surface has a unique position in space at any one time". However, our use of this constraint is subtly but importantly different from that of Marr and Poggio. We prefer to emphasise what might loosely be termed the converse of Marr and Poggio's Constraint 1. That is, we find it more helpful to say that "a given location in space can hold only one object at any one time", a formulation we call the Constraint of Spatial Uniqueness. The straightforward implication of this formulation is that any given disparity/position location should carry a symbolic description of similar figural type in the left and right eye views, and that rivalrous matches are therefore to be rejected. Hence the use of cross-channel combination rules to guide binocular fusion finds its computational justification quite readily in terms of constraints imposed by objects.

It is perhaps worth mentioning at this point that the binocular matching rule which we extract from the Constraint of Spatial Uniqueness is very different from the rule which Marr and Poggio (1976) extract from their equivalent constraint. They elaborated a rule which states that "each item from each image may be assigned at most one disparity value".

Unfortunately, this rule flatly contradicts the usual interpretation of Panum's Limiting Case (i.e. the interpretation which assumes that a feature in one eye's image can be matched to more than one feature in the other eye's image). Marr and Poggio themselves refer to this difficulty but note that when their uniqueness assumption is violated, then the algorithm can be made to assign a match which is unique from one image but not from the other (they cite O.J. Braddick, in preparation). We, however, avoid this difficulty altogether simply by avoiding Marr and Poggio's matching rule. Instead, our formulation of the Constraint of Spatial Uniqueness and its associated rule of utilising between-channel spatial correspondences copes easily and naturally with Panum's Limiting Case. This is because it makes evident sense within our own conceptual framework to parse out separate descriptive elements existing in different disparity/position locations when the stimulus input is a single line in one eye and two side-by-side lines in the other eye.

### 5.3 Psychophysical Evidence Supporting The Binocular Combination Rule of Cross-channel Correspondences.

We have reported elsewhere certain contrast summation effects operating across spatial frequencies at stereothreshold and how these support the notion of cross-channel global disambiguation processes in human stereopsis (Mayhew and Frisby, 1978b). We now describe two further studies on this theme but based on quite different psychophysical paradigms.

#### 5.3.1. The Case of the Missing Fundamental

One line of evidence which has led us to believe that cross-channel combination rules play a role in guiding binocular fusion comes from an investigation we have conducted (Mayhew and Frisby, 1980b) into the binocular fusion of square wave gratings with a missing fundamental (figures 1a and 1b). Disparate waveforms of this kind present interesting ambiguities about which left/right zero crossing matches are to be selected. For example, consider figure 1d which shows a magnified sample of the luminance profile on the left eye's image. When convolved with a spatial frequency channel whose centre frequency is that of the 3rd harmonic of the grating, zero crossings found for this sample are as displayed in 1f (channel tuning modelled on the psychophysical data of Wilson and Giese, 1977). The interesting ambiguity within this sample is illustrated in connection with one particular zero crossing, that identified with a vertical dotted line. Note that it derives from the edge marked with a small arrow in the luminance profile of 1d. Note also that it can in principle be matched to at least two zero crossings in the left eye's convolution profile: these are shown in 1e, also with vertical dotted lines and again the parts of the luminance profile from which they derive are shown with small arrows, this time in figure 1c.

Now due to the fact that the stereopair of 1a/1b possesses a fairly large disparity (greater than half the period of the 3rd harmonic in fact), the correct zero crossing match (the leftmost one shown in figures 1e and 1f) is, as far as Marr and Poggio's (1979) model is concerned, outside the allowable range for the channel under consideration. Consequently, Marr and Poggio's model would predict that within this channel the smaller-disparity incorrect match would be the one selected (the rightmost one shown in figures 1e and 1f). Therefore, if this channel was the one critically mediating the perceived stereopsis, the predicted psychophysical result would be the percept of a grating receding relative to its surround.

(for crossed-eye fusion of 1a and 1b). This argument is borne out by a computer simulation of Marr and Poggio's algorithm which we have run on this stereogram and which does indeed choose the receding match. In marked contrast with this prediction, the depth effect which is actually seen is that of a protruding grating. That is, the human visual system selects the correct disparity match, despite the fact that it is of a size that puts it out of range for the type of channel being considered and despite the fact that an alternative within-range match is potentially available.

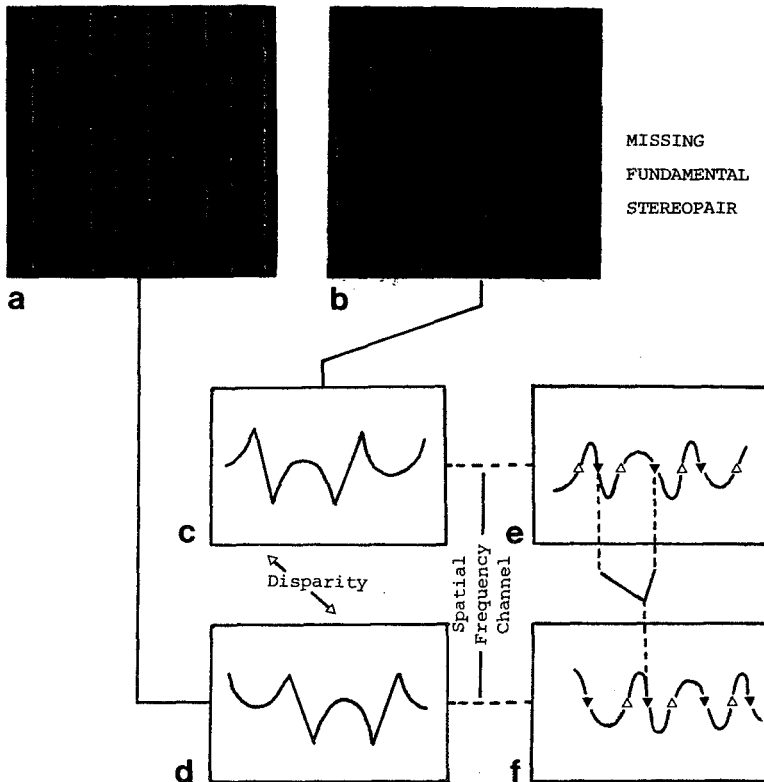


Fig. 1 The missing fundamental stereopair. See Section 5.3.1 for details. Δ and ▼ refer to positive and negative zero crossings respectively.

Thus what seems to be happening in the case of the missing fundamental stereogram is that stereopsis is based on the different types of edges in the luminance profile, rather than on zero crossing matches found and processed independently within spatial frequency tuned stereopsis channels. This conclusion is very much in keeping with the BRPS conjecture, because it requires that local matches found in all spatial frequency channels are considered jointly and in parallel, in an endeavour to find the best fitting binocular edge descriptions consistent with the total data provided by these channels.

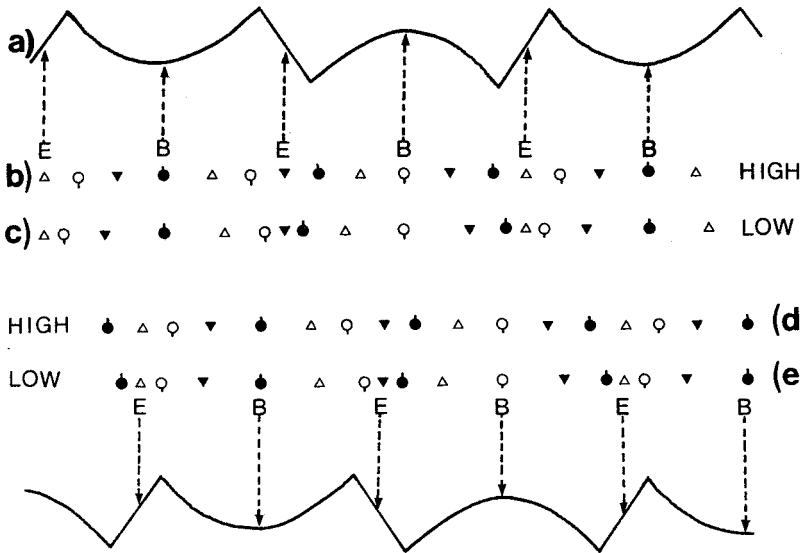


Fig. 2 Measurement primitives **extracted** from relatively high and low spatial frequency channels from left and right luminance profiles of the missing fundamental stereopair. See Section 5.3.1 for details.

This general scheme of binocular combination is illustrated in figure 2. Samples of left and right luminance profiles of the missing fundamental stereo pair are shown in 2a and 2f, and in between these are shown peaks and zero crossings discovered from these samples within two spatial frequency tuned channels (2b-e). Spatial coincidence (defined here as measurement primitives of the same type found in the same relative locations in both channels) occurs where vertical dotted lines are shown linking the channel primitives to the luminance profiles. The E and B symbols refer to edge and bar assertions which would be extractable in each case. Given the BRPS conjecture, this process would be effected binocularly, with consequent elimination of the ambiguities existing within each channel considered as a separate entity.

Certain possible objections to using the missing fundamental stereo pair to support the BRPS conjecture do, however, need to be considered:

First, it might be asked whether activity in other channels could provide a correct resolution of the ambiguity, without recourse to the type of processing envisaged by the BRPS conjecture. But in this connection, we note that channels dealing with frequencies higher than the 3rd harmonic are no better off than the channel illustrated in figure 1 (which is centred on the 3rd harmonic). Such channels would make just the same choice, considered as independent entities, with the correct match always out of range. And as far as channels tuned to lower spatial frequencies are concerned, channels which would of course have the range capable of dealing with the correct disparity in the stimulus, the fact that the stereo grating is missing its fundamental means that these channels would inevitably be only weakly stimulated (if indeed at all). More importantly, even if stimulated such low frequency channels would tend to 'see' just a simple sine wave. Sine waves are notoriously ambiguous stereo stimuli for which the lowest disparity solution is invariably selected, a fact illustrated by the lower half of figure 3 which shows the missing fundamental stereo pair filtered to reveal only its 3rd harmonic. In this half of figure 3, receding depth is seen with crossed-eye fusion - the 'incorrect' solution for the missing fundamental stereogram itself, a result which contrasts nicely with what is seen in the upper half of figure 3 in which the higher harmonics of the missing fundamental waveform are present, with the consequent result that protruding depth is once again evident.

Secondly, it might perhaps be maintained that correct stereopsis could be achieved by an independent channels model of the Marr and Poggio kind if initial left/right zero crossing matches were restricted only to those of roughly similar slope (adding this restriction to the already existing ones of similar contrast sign and similar orientation: Marr and Poggio refer to this possibility briefly in connection with stereopsis and Marr and Hildreth discuss its advantages more generally). Slopes for zero crossings within the channel illustrated in figure 1 are not in fact very different but for higher frequency channels the slopes do become progressively distinct. However, for these higher spatial frequency channels the disparity to be processed is well outside the range allowed by the model. Use of slope information is also not without other problems as far as stereopsis is concerned. For example, it is well known that stereo pairs of widely differing contrasts can readily be fused and this would seem impossible if initial local matches were restricted in a slope bound way. If it be thought that this consideration could be circumvented by some kind of 'normalisation' prior to establishing local matches, then the way

in which this might be done needs to be carefully considered. We would argue that any normalisation is most sensibly done locally in connection with arriving at binocular assertions, for example perhaps by seeking particular ratios of activity in left/right channels. Since it is the relative activity in the various channels which is used to determine the raw primal sketch element that is to be asserted, this proposal naturally leads straight back to the BRPS conjecture.

It is worth noting that 160 msec presentations of the missing fundamental stereo pair still produce correct stereopsis, so that any important involvement of the vergence mechanism seems ruled out, and also that we have done a similar analysis of this stereogram using peaks rather than zero crossings, with exactly the same conclusions.

Finally, it is worth pointing out that in the case of the missing fundamental stereo pair, where the ghosts and correct targets show equal figural continuity, we find that cross-channel correspondences are sufficient. In most images, of course, the two combination rules could be applied simultaneously to great advantage.

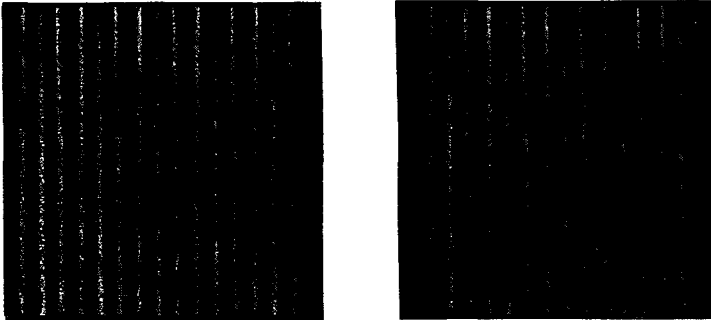


Fig. 3 Stereogram whose upper half is composed of the missing fundamental waveform described in figure 1 and whose lower half is that waveform filtered to reveal just the 3rd harmonic. Opposite depth effects are obtained in each half: see Section 5.3.1 for details.

### 5.3.2 Spatial Frequency Filtered Stereograms Portraying Corrugated Surfaces

Other psychophysical evidence implicating the use of cross-channel correspondences by the mechanisms of human stereopsis comes from a study using the horizontally-corrugated stereogram referred to earlier. The details of this study are described in the full version of this paper (Mayhew and Frisby, 1980c). In brief, the study showed that low spatial frequency information which on its own was not capable of providing a solution to the required psychophysical discrimination nevertheless facilitated disambiguation of the high spatial frequency content necessary for the discrimination. This was true even in the absence of eye movements.

In the next section we describe a computational experiment which demonstrates the potential value of cross-channel correspondences for solving both the missing fundamental and corrugated surface stereograms.

### 5.4 A Computer Simulation Demonstrating Binocular Use of Cross-Channel Correspondences

We have devised an algorithm (called FRECKLES: Mayhew and Frisby, 1980a) which uses peaks and zero crossings as measurement primitives to arrive at monocular raw primal sketch assertions, and we are presently engaged in extending this algorithm to cope with binocular inputs in a manner consistent with the BRPS conjecture and as already outlined in principle in connection with the missing fundamental stereo pair (figure 1). However, by way of evaluating the BRPS conjecture in general terms, we now describe an interim study which demonstrates that cross-channel correspondences can in principle provide a powerful basis for the computation of a local piece-wise binocular correlation.

Figure 4 illustrates a stereo algorithm which takes advantage of cross-channel correspondences at what might be termed a 'pre-parsing' level. The algorithm is based upon three monocular spatial frequency tuned channels (which is probably the sensible minimum to employ: two only were shown in figure 2 for simplicity). For each location (point) in one eye's image, a triplet of channel measurement primitives found at that location (an example is shown at the top of figure 4) is correlated with all other triplets found for locations in the other eye's image within a disparity range which defines Panum's fusional area for the algorithm (what range seems sensible is presently a matter of investigation). The correlation coefficient which we employ to discover the measure of agreement between left and right triplets is a fairly crude statistic but we suspect that almost any sensible weighting for the state of agreement between individual measures comprising each triplet will suffice. Thus for each spatial frequency, similar measurement primitives of similar contrast sign are weighted positively, and mis-matched primitives (i.e. 'rivalrous' ones of different type) are weighted negatively. Nil entries are allowed in each triplet (i.e. nil means the absence of a measurement primitive) and if a nil entry in one monocular triplet is coupled with a peak or zero crossing in a triplet from the other eye then this too reduces the correlation score. The details are, however, unimportant: suffice it to say that this type of cross-channel correlation finds the missing fundamental stereo pair trivially easy, and it can also reduce to negligible proportions the population of ghost matches for a 1D slice of a densely-textured random dot stereogram (figure 5), even over a disparity range more than twice as great as that allowed for the lowest spatial



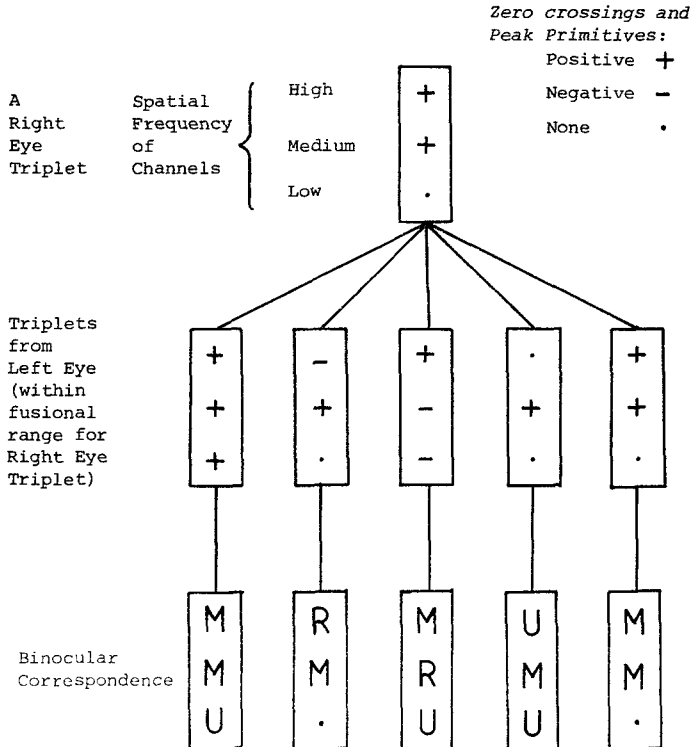


Fig. 4 A stereo algorithm taking advantage of cross-channel correspondences at a pre-parsing level. In the upper part of the figure is shown a triplet of measurement primitives found in three monocular spatial frequency channels at a particular location ('point'). Below this triplet is shown a sample of triplets found in the other field for locations within a given disparity range that defines Panum's fusional area for the algorithm. In the bottom row are shown the results of correlating the upper triplet with each of the middle triplets: this bottom row thus displays the preliminary stage of cross-channel binocular combination as far as the upper triplet is concerned. It is an easy matter to weight the kinds of measurement primitive matches found for each spatial frequency (see Section 5.4 for further comment on this) to enable selection of the correct triplet match. In the illustrated case, the triplet on the extreme right is appropriately selected as 'correct' because it possesses only correct matches (M). All others possess either rivalrous matches (R) or they contain primitives for one eye's view only (U - unocular), there being no primitive at all found in the other eye's triplet. Note that each positive or negative primitive could be either a zero crossing or a peak and that a rivalrous match would be logged if primitives of these different types were found.

frequency tuned channel in Marr and Poggio's (1979) algorithm. Moreover, we find that it can cope with a corrugated surface of the kind discussed in Section 5.3.2, with easy selection of correct matches in the peak and trough zones and degraded but nevertheless adequate performance for the regions of texture dealing with the slopes linking each peak and trough.

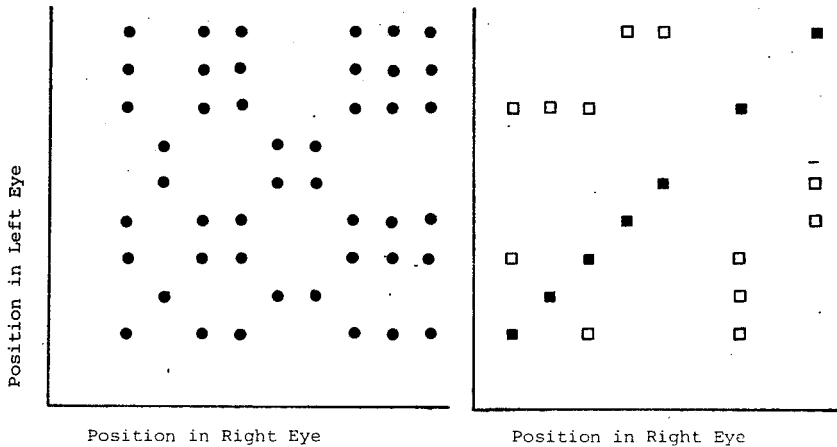


Fig. 5

A demonstration of binocular cross-channel combination for a 1D slice of a random dot stereogram. In (a) is shown the ghost structure for this slice if potential local matches are defined simply as black-for-black or white-for-white pixel matches. The correct disparity plane is zero so that the correct matches are those lying on the diagonal. As can be seen, on this basis the relatively fine texture of the stereogram hides the correct matches within an enormous pool of possible matches. In (b), the same 1D slice is input but now the ghost structure is shown when each potential local match is based on the discovery of a matching peak or zero crossing primitive of similar contrast sign in any and all spatial frequency channels. It thus defines the ghost structure for an independent channels model of stereopsis. The filled-in squares of (b) are those matches selected by the cross-channel stereo algorithm described in Section 5.4 and illustrated in figure 4.

A full evaluation of our cross-channel algorithm which compares it against other image-processing cross-correlation techniques will be the subject of a further report. Suffice it to say for the present that our preliminary studies have been very encouraging, returning much sharper autocorrelation functions than classical techniques. Most importantly for present purposes, the algorithm demonstrates that cross-channel correspondences can in principle provide an excellent basis for binocular combination, and therefore that the BRPS conjecture gains considerable support from this computational experiment. This is particularly so when it is realised that each channel on its own suffers a fairly dense ghost structure (figure 5) for the random dot stereogram used to explore the properties of the algorithm to date.

### 5.5 Summary

To summarise this section, it seems that the processes of human binocular combination optimally combine patterns of zero crossing and peak matches presented in parallel by several spatial frequency channels. Matches are not chosen independently of their cross-channel context but instead selected according to cross-channel combination rules and interpretive constraints forced by the requirement to produce a coherent binocular description of the local intensity changes in the scene. Our psychophysical data challenge any model based purely on the independent within-channel processing of zero crossings and peaks, such as that of Marr and Poggio (1979; see also Frisby and Mayhew, 1977), and our computational experiments demonstrate that cross-channel correspondences are in fact capable of aiding binocular combination in a powerful fashion. Whether cross-channel correspondences that violate the principle of spectral continuity should be used for disambiguation is currently the subject of psychophysical investigation.

### 6 Micropattern Matching vs. The BRPS Conjecture

It might be wondered whether the BRPS conjecture is equivalent to the oft-cited idea that the stereo ambiguity problem might be resolvable by matching similar micropatterns in the two eyes' views. Julesz considered just such an idea at the outset of his research programme using random dot stereograms but quickly rejected it following a simple experiment. Thus he found that if equivalent left/right micropatterns were perturbed by changing the diagonal connectivities between neighbouring points in one eye's image, then stereopsis survived despite the radically different appearance of the left and right eye images which this procedure creates (Julesz, 1960). One might add to this line of evidence Julesz's demonstration that stereopsis also survives strongly blurring one image, again despite the remarkably different left/right micropatterns that this procedure creates (see Julesz(1971) for stereo illustrations relating to both these experiments). One is forced to conclude that as far as human stereopsis is concerned, disparity extraction cannot be mediated solely by matching exactly similar micropatterns in each eye's view. We would add to this the fact that high pass filtered textures which preserve the micropattern structure of the original are more difficult stimuli to fuse than would be expected were micropatterns used as matching primitives.

The BRPS conjecture proposes a form of binocular combination that takes advantage of local figural information, but note that the stereo algorithm described in Section 5.4 and developed with the BRPS conjecture

in mind does not require identical micropatterns in the two fields for it to work satisfactorily. At least in principle (and we are currently exploring details of implementation), the cross-channel triplet matching we have proposed copes naturally and easily with many kinds of left/right perturbations which it is known human stereopsis can surmount, such as those described in the preceding paragraph. That is, this type of stereo algorithm will find a 'best fit' correlation of the two fields without insisting that the correlation be perfect. All that is required is that some spectral overlap exists between the left and right textures. Interestingly, this is just the requirement which human stereopsis seems to demand, because large spectral differences produce marked binocular rivalry (Mayhew and Frisby, 1976; Frisby and Mayhew, 1978).

#### 7 Concluding Remarks

We suggest that the local and global combination rules listed above will be a sufficient basis for obtaining global stereopsis from most stereo inputs. They will clearly fail in certain situations, however, for example those involving strictly repetitive sub-patterns of the kind used to create ambiguous random dot stereograms (Julesz, 1971) and effects such as the wallpaper illusion. But repetitive patterns pose problems for any stereo algorithm because alternative coherent and stable fusions are intrinsic to their design. The human visual system seems to deal with them simply by choosing that set of disparity matches closest to the fixation plane (Julesz and Chang, 1976) and by not allowing fusions which would be spatially incoherent. That is, at any one moment the human visual system seems to reject binocular assertions which would amount to positing elements existing in disparity/position locations that would entail them being masked in both eyes' views by other elements asserted in other occluding locations. This need not be thought of as an ad hoc restriction because it could be based on what might be called the 'opacity constraint' presented by objects in the world, i.e. if a non-transparent entity is asserted in a given location then it is sensible not to assert other entities hidden behind it which could not be seen from either eye's viewpoint. The use of this constraint would not preclude Panum's limiting case and it is consistent with the limited subset of perceptual outcomes that appear for stereo inputs of the nail illusion kind.

There are, of course, many other problems to be solved for a 'complete' model of stereopsis. For example, it is necessary to face the problems posed by the need to interpolate disparity assignments to those regions of the field of view between binocular raw primal sketch assertions. This enters the domain of processing required for the 2½D sketch (Marr and Nishihara, 1978), a representation of surfaces and their orientations in depth based upon many cues besides disparity and a processing objective beyond the scope of this paper.

Acknowledgements

We would like to thank Philip Stenton for his considerable help in running various psychophysical experiments referred to in this paper and for his patient assistance during program development. Alison Kidd made some useful comments on the manuscript, and Chris Brown's help with equipment at all stages has been invaluable. Sandra Lowry, our typist is thanked for her patient reaction to late changes in the manuscript.

The research was supported by SRC research grant GR/A/50894

References

- Frisby, J.P., Mayhew, J.E.W. (1977) "Global processes in stereopsis: some comments on Ramachandran and Nelson." Perception 6 195-206.
- Frisby, J.P., Mayhew, J.E.W. (1978) "The relationship between apparent depth and disparity in rivalrous texture stereograms." Perception 7 661-678.
- Frisby, J.P., Mayhew, J.E.W. (1980) "Spatial frequency tuned channels: implications for structure and function from psychophysical and computational studies of stereopsis." In The Psychology of Vision, Longuet-Higgins, C. and Sutherland, N.S. (Eds.), Philosophical Transactions of Royal Society, London, B. (in press).
- Grimson, W.E.L., Marr, D. (1979) "A computer implementation of a theory of human stereo vision." Image Understanding Workshop, April 1979, A.I. Lab., M.I.T., Cambridge, Massachusetts.
- Julesz, B. (1960) "Binocular depth perception of computer-generated patterns." Bell Systems Technical Journal 39 1125-1162
- Julesz, B. (1971) Foundations of Cyclopean Perception. Chicago: University of Chicago Press.
- Julesz, B., Chang, J.J. (1976) "Interaction between pools of binocular disparity detectors tuned to different disparities." Biological Cybernetics 22 107-119.
- Marr, D. (1976) "Early processing of visual information." Philosophical Transactions of the Royal Society, London, Series B, 275 483-524.
- Marr, D., Hildreth, E. (1979) "Theory of edge detection." A.I. Memo No. 518, A.I. Lab., M.I.T., Cambridge, Massachusetts.
- Marr, D., Nishihara, H.K. (1978) "Visual information processing, artificial intelligence and the sensorium of sight." Technology Review 81 2-23.
- Marr, D., Poggio, T. (1976) "A cooperative computation of stereo disparity." Science 194 283-287.
- Marr, D., Poggio, T. (1977) "A theory of human stereo vision." A.I. Memo No. 451, A.I. Lab., M.I.T., Cambridge, Massachusetts.

- Marr, D., Poggio, T. (1979) "A theory of human stereopsis." Proceedings of the Royal Society, London, Series B, 204 301-328.
- Marr, D., Ullman, S., Poggio, T. (1980) "Bandpass channels, zero crossings and early visual information processing." Journal of the Optical Society of America 69 6 914-916.
- Mayhew, J.E.W., Frisby, J.P. (1976) "Rivalrous texture stereograms." Nature 264 53-56.
- Mayhew, J.E.W., Frisby, J.P. (1978a) "Stereopsis masking in humans is not orientationally tuned." Perception 7 431-436.
- Mayhew, J.E.W., Frisby, J.P. (1978b) "Contrast summation effects and stereopsis." Perception 7 537-550.
- Mayhew, J.E.W., Frisby, J.P. (1979) "Surfaces with steep variations in depth pose difficulties for orientationally tuned disparity filters." Perception 8 691-698.
- Mayhew, J.E.W., Frisby, J.P. (1980a) "The computation of binocular edges." Perception 9 69-86.
- Mayhew, J.E.W., Frisby, J.P. (1980b) "The king is naked: another case of the missing fundamental." Paper read to the London meeting of the Experimental Psychology Society, January, 1980.
- Mayhew, J.E.W., Frisby, J.P. (1980c) "Computational and psychophysical investigations of human stereopsis." Artificial Intelligence. This is the full version of the present paper.
- Mayhew, J.E.W., Frisby, J.P., Stenton, P. (1980) "Evidence for facilitation between spatial frequency tuned stereopsis channels." Perception 9 (in press).
- Wilson, H.R., Giese, S.C. (1977) "Threshold visibility of frequency gradient patterns." Vision Research 17 1177-1190.

## SOME PROBLEMS IN EARLY NOUN PHRASE INTERPRETATION

C.S.Mellish,  
Department of Artificial Intelligence,  
University of Edinburgh,  
Edinburgh EH8 9NW, UK

Abstract

How does a piece of text provide the information necessary for generating a symbolic "meaning" and how can a computer program be organised to pick up that information? The work described here aims to investigate some of the constraints on the timing of semantic interpretation. In particular, we are interested in seeing to what extent the meaning can be built up in an incremental way as the analysis proceeds from left to right. We look at some problems of noun phrase interpretation in such a scheme and indicate some representational ideas that help to overcome them. This paper is a brief summary of a forthcoming PhD thesis [Mellish 80].

Keywords

Parsing, reference evaluation, quantification in natural language, computational linguistics.

Introduction

How would noun phrases be treated in a system for semantic interpretation that worked strictly left-to-right? Semantic analysis of a noun phrase might be expected to immediately determine the set of objects in the world that it refers to. This is, of course, an impossible requirement, for many noun phrases in themselves only give incomplete information about their referents. Even if we make use of the combined syntactic and semantic context coming before a noun phrase, it is still often impossible to determine on the spot exactly what is involved. We consider here two areas where left-to-right analysis encounters problems - definite reference evaluation and the interpretation of indefinite noun phrases. For each, we indicate how appropriate choices of representation enable the idea of left-to-right processing to be largely retained.

Basic Framework

The ideas presented in this paper are embodied in a working computer program [Mellish 79] that was developed as part of a system to solve mechanics problems stated in English [Bundy 79]. In this system, a model of the small world described by a mechanics problem is kept in the form of a database of Predicate Calculus formulae. Objects in the world are represented here by constant symbols, and possible relationships between

them by predicate symbols. Since the possible worlds are all finite and the information conveyed in a mechanics problem is normally fully specific, in practice the formulae are of a restricted sort. This fact is used in certain assumptions that our program makes, namely:

- Each noun phrase refers to a finite number of objects.
- It is not necessary to deal with the logical operators 'or' or 'not' in connection with information about the world (the world model is a simple conjunction of facts).
- It is also not necessary to deal with full quantification (over sets which cannot be conveniently enumerated) in this context.

It remains to be seen to what extent our work can be extended to handle domains where these assumptions do not hold.

#### Definite Reference Evaluation

An important paper by Ritchie [Ritchie 76] discusses some of the problems of carrying out semantic interpretation on a local basis. Ritchie makes two main points, which are closely related to the problem of reference evaluation. Firstly, whether an interpretation is semantically anomalous (eg whether an inappropriate referent has been chosen) cannot in general be decided locally, but can only be evaluated within a global system of "preference" [Wilks 75]. Secondly, important "environmental" factors (such as the relevant time period) may be unknown when a phrase is evaluated. Because of this, it may not even be possible to obtain reasonably sized sets of candidate referents locally.

While we fully agree with Ritchie's points, we feel that there are interesting problems in local reference evaluation even in examples where the candidate sets are small and semantic anomaly is fairly clear-cut. Therefore these particular issues will not be considered further here. In our simplified view, definite reference evaluation can be seen as the task of instantiating a variable with a value that satisfies a set of constraints. The constraints arise both from the explicit information given in the definite description and from preconditions ("semantic checks") associated with relationships that the referent is taking part in. Consider the referent of "it" in the sentence:

A rod is supported by a string attached at its ends. (1)

Because of what the word "it" means, the referent must be inanimate and singular. Further, because of preconditions for the relationships it is involved in, it must be something able to have ends, and its ends must be things physically capable of being attached to the string. Constraints affecting the evaluation of a reference may arise at many places in the analysis. Is it possible to arrange for their satisfaction to take place within a "left-to-right semantics" framework?

We propose a strategy that attempts to keep all its options open until the



accumulated constraints have narrowed the possible interpretations to one. This is not a simple postponement of reference evaluation, for it is possible to use information about partially-evaluated references to influence which possibilities in the parsing are considered. Nor is this a simple substitution of breadth-first for depth-first search, for the use of appropriate partial representations enables common paths to be merged until a choice is absolutely necessary. Because the method is incremental, there can be a close interaction between reference evaluation and other analytical processes, which allows false hypotheses to be rejected early.

In order to carry out incremental reference evaluation it is necessary to have representations of partially-evaluated references and to be able to perform semantic operations on these. Semantic routines must be able to handle unevaluated references in the same ways as they handle other objects in the world, and so there must be (at a superficial level) no significant difference between the ways in which these are represented. Hence we are led to the idea of having explicit "reference symbols" as well as symbols corresponding to unique objects in the world. Since at a deeper level there are basic differences between how the two kinds are to be treated, we must associate extra meta-information with a constant symbol. For a reference symbol, this information includes the current candidate set and any constraints that link its value to that of others. Hence when a new constraint involving one or more partially-evaluated references is generated, the implications for all the candidate sets can be followed up by a "filtering" algorithm [Waltz 72, Mackworth 77].

### Interpreting Indefinite Noun Phrases

When we come to an indefinite noun phrase in a left-to-right analysis, we may be unable to tell how many objects in the world are referred to, either because of the vagueness in the phrase itself or because a dominating quantifier has not yet been read.

Small blocks, each of mass  $m$ , are clamped at the ends  
and at the center of a light rod. (2)

A wooden stool 2ft 2in high consists of a square seat with  
a uniform vertical leg at each corner. (3)

It follows that an interpretation in terms of concrete referents is not possible. However, with a phrase like "some blocks", although we do not know how many objects there are, the information we have about each one is identical. It follows that most kinds of inferences (such as checking the suitability of the adjective "small") would proceed in exactly the same way for each one. This justifies considering all the elements of the set at once, by using a "typical element". We have therefore introduced typical element symbols into the world model as possible referents of phrases. These can, in fact, be used equally well when there is knowledge about the set's cardinality, as with the phrases "3000 blocks" and "2n blocks".

The way an indefinite NP refers to a set of objects may well indicate more structure than is conveyed by a simple set representation. It may say something about how the set decomposes into subsets. Consider the set of pulleys in:

A length of rope and two blocks each containing 3 pulleys are supplied.

(4)

In this example, there are six pulleys in total. Firstly, there is a "top level" decomposition into three subsets, as communicated by the number in the phrase. Secondly, each subset decomposes into two elements, corresponding to the two blocks.

Although we can do a lot of work at the level of typical elements of sets, it is unreasonable to expect that we will never have to deal with individual elements. So we must keep track of what kind of set a 'typical element' corresponds to - how it decomposes into subsets, what cardinality information is known and so on. Not all of this will be immediately available, so it will be necessary to represent partial information that can be gradually updated.

Our way of keeping track of these matters is to keep a "dependency list" associated with each constant symbol in the world model. This will be able to expand as necessary to record all the separate dimensions making up the set of objects. Each entry in a dependency list has information about the origin of the dependency, the cardinality associated with it and whether it still corresponds to a division of the set into "indistinguishable" subsets.

Assertions formulated in terms of typical elements must be useable by the inference system at various times in the semantic analysis. Each time there may be more concrete information about the dependencies than the time before. An indefinite noun phrase referent may start with no known dependencies, may then accumulate some as a quantifier is discovered, and finally inferences may be carried out in terms of a very specific element of the set. At each stage, the very earliest assertions made about the referent may be needed. The meta-information embodied in the dependency list provides the basis for interpreting the assertions in the correct amount of detail each time.

### Conclusions

In this paper we have introduced representations for partially-evaluated references and various kinds of typical elements of sets. By expressing propositions in terms of these instead of "concrete referents", we can often maintain a policy of early noun phrase interpretation without encountering overwhelming search problems.

### Acknowledgements

This research was made possible by the provision of an SRC studentship and computer resources in connection with the MECHO project at Edinburgh University (SRC grant number B/RG 94493).

### References

[Bundy 79]

Bundy, A., Byrd, L., Luger, G., Mellish, C. and Palmer, M.  
Solving Mechanics Problems using Meta-Level Inference.  
In Procs of the Sixth International Joint Conference on  
Artificial Intelligence. IJCAI, 1979.

[Mackworth 77]

Mackworth, A.  
Consistency in Networks of Relations.  
Artificial Intelligence 8:99-118, 1977.

[Mellish 79]

Mellish, C.S.  
Computer program, to be found in files /us/gris/nl/\* on the  
departmental computer, Dept of Artificial Intelligence,  
University of Edinburgh.

[Mellish 80]

Mellish, C.S.  
Coping with uncertainty: Noun phrase interpretation and  
early semantic analysis.  
PhD thesis, Dept of Artificial Intelligence, University of  
Edinburgh, 1980.  
(forthcoming).

[Ritchie 76]

Ritchie, G.D.  
Problems in Local Semantic Processing.  
In Brady, M., editor, Proceedings of the AISB Conference,  
Edinburgh. AISB, 1976.

[Waltz 72]

Waltz, D.  
Generating Semantic Descriptions from Drawings of Scenes  
with Shadows.  
Technical Report, MIT, 1972.

[Wilks 75]

Wilks, Y.A.  
A Preferential, Pattern-Seeking, Semantics for Natural  
Language Inference.  
Artificial Intelligence 6, 1975.

## USING DETERMINISM TO PREDICT GARDEN PATHS

Rob Milne

Department of Artificial Intelligence and School of Epistemics  
 University of Edinburgh  
 Edinburgh, Scotland EH8 9NW

ABSTRACT

I am interested in making a psychologically valid model of human natural language understanding, and especially in a processing model for predicting when a sentence will be a garden path. While extending the Marcus deterministic parser to include noun-noun modification, several counter examples to Marcus' garden path prediction were found. In this paper I propose that when people encounter an ambiguous situation that may lead to a garden path, they use semantics to decide rather than look ahead. I will present an extension to the garden path prediction mechanism of Marcus' parser to account for this and several experiments to test this theory.

INTRODUCTION

I am interested in making a psychologically valid model of human natural language understanding. Especially in the question: "Is it possible to predict which sentences will cause people to garden path?" A garden path sentence is a sentence which people cannot properly analyze without the need to re-analyze (backtrack on) a portion. For example:

[1] The horse raced past the barn fell.

In each sentence of this type there is a point where two possible analyses are possible (i.e. at raced). The need to backtrack is a result of selecting an analysis different from that demanded by the rest of the sentence. For each garden path sentence there is a partner sentence that does not require backtracking, e.g.

[2] The horse raced past the barn.

This partner sentence has the same two possible readings at the same point, but the analysis selected is the one demanded by the rest of the sentence. Such a pair of sentences will be called a pair of potential garden path sentences. Another pair of potential garden path sentences is:

[3] The building blocks the sun faded are red.

[4] The building blocks the sun.

For sentences [3] and [4], some people would need to backtrack on [3] and some would need to backtrack on [4]. Neither of these sentences is a garden path for every person, but each is a garden path for some people.

We would like our model to analyze without backtracking, sentences which people don't seem to need to "backtrack" on. If we build a parser that never backtracks, then it would analyze non-garden path sentences properly, but fail to analyze a garden path sentence. This model would then predict a garden path sentence as being any sentence it is unable to analyze.

For these purposes, an ATN model [Woods 70] is clearly inadequate. Even though ATN parsers can be implemented very efficiently, their extensive use of backtracking eliminates them as a possible model. Instead we take the

Marcus [1977] deterministic parser as a starting point.

### THE EXISTING PARSER

I have implemented a version of Marcus' original parser in Prolog. The entire system has been designed to be "deterministic". That is it "never undoes structure that has been built up". Marcus has shown that a large subset of natural language can be parsed without the need for backtracking, if the parser uses two simple techniques. The first of these is 3 constituent look ahead. The grammar is written such that each rule can examine 3 "buffers". Each buffer can contain any constituent representable as a single node. (i.e. a word, a NP, an embedded sentence, etc.) This technique causes "wait and see". If it is unclear how to use a certain word or constituent during the parse, the parser "waits to see" what it should be. By using the look ahead and "wait and see" techniques, it is possible to analyze properly many sentence forms without the need for backtracking or undoing structure. For a fuller discussion of determinism, see Marcus or Milne [1979].

During the implementation of the parser, I extended and modified it to handle several areas Marcus did not include. In Marcus' original parser, no facility was included for noun-noun modification. Instead each NP could have only one headnoun. This prevented the parser from analyzing many sentences such as:

[5] The cover screw is red.

In [5] the complex headnoun (cover handle) could not be built in Marcus' parser. The parser was also not able to deal with words defined as multiple parts of speech. In order to extend the parser to allow complex headnouns, a problem arose determining the end of a NP, which I shall call the "end of constituent" problem.

### END OF CONSTITUENT

In the sentence fragment:

[6] The cover screws....

The sentence could be completed as either:

[7] The cover screws are red.

[8] The cover screws easily.

In sentence [7] the headnoun is (cover screws) while in [8] it is only "cover". Detecting this and deciding to attach each possible headnoun, is an example of the "end of constituent" problem. This problem is especially important for the question of PP attachment. For, in order to attach a PP to make a larger NP, the root NP must be located. An ATN parser is able to choose one possibility and then backtrack, if it discovers this is incorrect. But a deterministic parser is not able to backtrack, and must identify the correct end without an error.

For a deterministic parser, a sentence such as [9] presents no problem finding the end of the constituent.

[9] The falling block is made of wood.

In [9], the word "falling" can be either a verb or an adjective and "block" can be either a noun or a verb. The general solution to this case is implemented implicitly by the grammar packets. While an NP is being parsed,

the parser de-activates all rules not dealing with NP elements. That is, when the parser is expecting a noun for the sentence, no rule is active that can recognize a verb, or any part of the auxiliary. The adjective rule will match on "falling", making it an adjective and will never consider "falling" as a verb. Likewise for "blocks". Whilst an NP is being built, words are attached to the Current Node Stack and removed from the first buffer. Grammar rules can only match the buffers, so eventually the AUX or a verb will arrive in the first buffer. At this time the only grammar rule to match will cause the NP to be finished. In this way ambiguous word in a NP are handled.

### THE FINAL S

This first case is trivial and presents no problem. The greatest difficulty arises when there is a series of words that can be either nouns, or verbs. The following examples illustrate this:

- [10] The soup pot cover handle screw is red.
- [11] The soup pot cover handles screw tightly.
- \*[12] The soup pot cover handles screws tightly.
- [13] The soup pot cover handle screws tightly.
- [14] The soup pot cover handle screws are red.

Each of the words (soup pot cover handle screw) can be either a noun or a verb. The end of constituent problem is to find which word is used as a verb and which words make up the complex headnoun.

In [10] each word is singular. For this case all words must be nouns and are part of the headnoun. In [11] "handles" is noun plural. In this situation each word before it must be a noun. When a noun/verb word follows it, the word (screw) must be a verb and "handles" is the last of the headnouns. It is not possible in this situation to use "handles" as a verb. Sentence [12], with two plural words, is ungrammatical. This case will not be dealt with. (Do not confuse plural "s" with possessive "'s").

Sentences [13] and [14] both have the same word string until after "screws", but in [13] "screws" is a verb while in [14] "screws" is part of the headnoun. In this situation where the final word of the series is plural, each word before it must be a noun. The plural word can be either a noun or a verb, depending on what follows. Now consider the following sentences:

- [15] The toy rocks the child vigorously.
- [16] The toy rocks the child has are red.

These two are a pair of potential garden path sentences, as described in the introduction. They demonstrate that, in the situation of a singular noun/verb word followed by a plural noun/verb word, it is always possible to finish the sentence so that it will be a garden path. Deterministic parsing, as Marcus proposed, will not be able to handle all occurrences of this situation.

### PREDICTING GARDEN PATHS

As I explained in the introduction, I am interested in predicting garden paths. We have just seen, while handling the end of constituent problem a case that leads to potential garden paths. Can our model predict when this

case will be a garden path?

First let me explain the garden path prediction of Marcus' parser. The parser consists of a active node stack where partially built items reside, and 3 buffers. Each buffer contains a word or constituent that can be represented by a single node (i.e. word, NP, PP, VP, etc.). In the best situation, an ambiguous word will be in the first buffer and the look ahead will be two items (Buffers 2 and 3). When an NP is being built, these two items of look ahead will be words and never whole NPs or larger items. This is because a NP is built using an Attention Shift [Marcus 77] and it is not possible to perform an Attention Shift whilst in another Attention Shift. At the time the word "rocks" is being analyzed in sentence [15], the parser state will be:

Active Node Stack: NP    the  
                                  toy  
Buffers:                [rocks] [the] [child]

A sentence is predicted to be a garden path if the look ahead is not sufficient to disambiguate the word correctly. The original prediction did not encompass potential garden path sentences. Instead, for the case of [15] and [16], it would arbitrarily choose one case always to be a garden path.

It should be noted that, in the case of:

[17] Have the students take the exam.

[18] Have the students taken the exam?

the look ahead will be: [have] [NP] [take/taken]

since the ambiguous word is not part of a NP. I stated earlier that for each person either [15] or [16] must be a garden path. Determinism predicts that both are a potential garden path, but cannot tell which it will be. This uncertainty suggests some possible counter examples to the garden path prediction of determinism.

In fact, the classic garden path:

[19] The prime number few.

is a counter example to determinism's prediction. When the parser is analyzing the word "number" the state will be:

Active Node Stack: NP    the  
                                  prime  
Buffers:                [number] [few] [.]

Since the entire sentence fits into the three buffers, all information to analyze the sentence is available, but people do garden path on this sentence. I predict the following to be another counter example:

[20] The jeep rocks are red.

Even though the number of words until the error is realized is very small, people are aware of some confusion whilst analyzing this sentence. Again, all the information for proper analysis is contained in the three buffers and it is predicted not to be a garden path.

These two sentences are counter examples to our prediction of garden paths, but this does not mean that all our predictions are no longer valid. Sentences of the type [1] will be properly predicted to be garden paths. I propose the following extension to our theory if garden path prediction is to handle these apparent counter examples:-

When a person encounters a situation such as a noun/verb word followed by a noun/verb that is plural, instead of using look ahead, they attempt to make a complex item name of the two words using semantic information. They do this without regard to the following words in the sentence. If their preference for this leads to an analysis different from the analysis demanded by the sentence, then they will garden path. Notice this prediction now depends on a semantic preference for complex nouns, rather than on look ahead. I will explain this preference more in the following section.

As the above examples show people like prime numbers, but don't like jeep rocks. I believe people will garden path if "prime number" is not a complex headnoun because it is a common construction, as with "map pins" and "granite rocks". People will use "rocks" as a verb in "jeep rocks" since it is very difficult to imagine the complex item (jeep rock). People also will not build complex item names of "boy screws" and "cook handles". For each of these, the 2nd word is predicted to be used as a verb. Finally for the case as in [15] and [16] (toy rocks), both constructions are equally possible, so some people would garden path on [15] and some on [16]. It is also very easy to bias this last case with context, etc. altering the predictions.

#### FURTHER WORK

The theory just presented will provide a better prediction of garden paths. Previously the garden path prediction of determinism was only, "this form may lead to a garden path". The new theory will be able to tell us more accurately which sentences are garden paths and which are not. But this theory relies on the psychological preference of people to construct complex named objects, without using look ahead. In order to use the new prediction, data needs to be gathered to show these biases and determine if our new predictions are correct. I conducted two experiments to gather data on biases and test this theory.

#### TESTING THE PREDICTIONS

An informal survey was first conducted to test the generation of complex item names, which supported the above proposal. A reaction time experiment was then performed. The purpose of the experiment was to present sentences predicted to be garden paths by our new theory, and those believed to be wrongly predicted to be garden paths by the old theory and test our predictions. The subject was directed to read the series of words presented and indicate whether it was a complete sentence or not. Reaction times were recorded from the presentation of the sentence until the response. The test examples contained a mix of obvious fragments, obvious sentences, and a mixture of control and test sentences. The test sentences included such examples as:

- [21] The jeep rocks are red.
- [22] The building blocks the sun.
- [23] The granite rocks during the earthquake.
- [24] The granite rocks were by the seashore.

Our theory predicted that sentence [23] will take longer to process than sentence [24]. This is because the reader will have a preference for



(granite rocks) and hence make an error and require garden pathing in [23], but not in [24]. The experiment confirmed this prediction, but space doesn't allow me to include the details here.

#### A POSSIBLE EXPLANATION

It is generally agreed that PP attachment, to be correct, must be done on a semantic basis. Crain and Coker [1979] have shown that the problem of "raced" in [1] is also resolved on a semantic basis, rather than using syntax. I have just proposed that the same is true for ambiguous noun/verb plural words.

In each of these situations, syntax does not provide sufficient information to prevent a garden path, and in each situation we have claimed semantics is used for the disambiguation, rather than look ahead. This suggests that, in a potential garden path situation, people do not use look ahead, but resolve the ambiguity using semantics, and this semantic basis can easily be biased by context. This proposal needs to be further tested and checked carefully, before we are sure it is true.

#### CONCLUSION

Whilst extending the Marcus parser to deal with complex headnouns, we came across a case leading to garden paths. This then presented counter examples to the prediction of garden paths made by determinism as presented by Marcus. The reason these predictions are wrong, is that when people encounter a situation that may lead to a garden path, they don't use look ahead, but instead attempt to disambiguate the situation on purely semantic grounds. Finally I performed two experiments to test this theory.

#### BIBLIOGRAPHY

- Bever, T., Garret, M., Hurtig, R. [1978] "The Interaction of Perceptual Process and Ambiguous Sentences" in *Memory and Cognition*, v1.1.n3.
- Cairns, H., Kamerman, J. [1975] "Lexical Information Processing During Semantic Comprehension", in *The Journal of Verbal Learning and Verbal Behavior*, vol. 14.
- Crain, S. and Coker, P. [1979] "Lexical Access During Sentence Processing", and "A Semantic Constraint on Parsing", papers presented at the Linguistics Society of America Annual Meeting.
- Fodor, J., Frazier, L. [1978] "The Sausage Machine: A New Two-Stage Parsing Model", in *Cognition*, vol. 6.
- Marcus, M. P. [1977] "A Theory of Syntactic Recognition for Natural Language", unpublished Ph.D. thesis, MIT.
- Milne, R. [1978] "Handling Lexical Ambiguity in a Deterministic Parsing Environment", unpublished B.Sc. thesis, MIT.
- Milne, R. [1979] "A Framework For Deterministic Parsing Using Syntax and Semantics", DAI Working Paper No. 64.
- Pereira, C.M., Pereira, F.C.N. and Warren, D.H.D. [1978] "User's Guide to DECsystem-10 PROLOG", Available from the AI Dept, Edinburgh.
- Woods, W. A. [1970] "Transition Network Grammars for Natural Language Analysis", *Communications of the ACM* 13:591.

STRATEGY GRAMMARS.

AN APPROACH TO GENERALITY IN COMPUTER SIMULATION

OF HUMAN REASONING

Stellan Ohlsson

Department of Psychology, University of Stockholm,

Box 6706, S-113 85 Stockholm, Sweden

The concept of a strategy grammar is introduced as one of several possible approaches to the problem of how to express properties which recur over many simulation programs induced from think-aloud protocols. An example of such a grammar is presented which is capable of generating several specific simulation programs which have been verified against human data. It turns out that additional programs, not corresponding to any observed subject, can be derived from the grammar. If the grammar is interpreted as a theory, such derivations correspond to predictions about which strategies people will be found to use in a particular task domain. Also, it is shown that not all programs found in human data can be derived from the particular grammar shown. Thus, a strategy grammar can categorize subjects with respect to their problem solving strategies. Other approaches to the same problem are briefly commented upon.

Introduction

Computer simulation of human thought has become a common-place event (see Simon, 1979, for a review). One of the problems for the computer simulation approach is how to abstract out common properties of several simulation programs, without reverting to the uninformative generalizations which plagued psychological theorizing in the past. The purpose of this paper is to outline one approach to this problem. To make the discussion concrete, it is presented in the context of a substantive example.

Spatial Reasoning

In the study of problem solving, the tasks used have usually been either so-called puzzles, which de-emphasize the role of prior knowledge, or technical task domains like physics and chess, in which the background knowledge is learned through a process of explicit training. The work reported here investigates a well-structured and semantically rich, but non-technical, knowledge-area, namely spatial knowledge as it is expressed in the use of concepts like "left", "between", "topmost", etc.

A number of think-aloud protocols have been collected from students who were asked to solve spatial arrangement problems (Ohlsson, 1980). Two

examples of such problems are given in Figure 1. The subjects solved the problems in their heads, without any external aids other than the card with the problem text.

#### The Block Problem

A child is putting blocks of different colors on top of each other.

A black block is between a red and a green block.  
 A yellow block is further up than the red one.  
 A green block is bottommost but one.  
 A blue block is immediately below the yellow one.  
 A white block is further down than the black one.

Which block is immediately below the blue one?

#### The Tool Problem

A craftsman has some tools in a row on his workbench.

The saw and the pair of tongs are right by each other.  
 The jackplane is immediately to the left of the knife.  
 The knife is further right than the chisel.  
 The bor is immediately to the right of the pair of tongs.  
 The jackplane is further left than the saw.

Which tool is immediately to the left of the saw?

Figure 1. Two examples of spatial arrangement problems.

In several studies, it has been found (a) that a majority of the subjects use some version of what Quinton and Fellows (1975) have called the Method of Series Formation, i. e. they try to form a mental model of the spatial arrangement talked about in the problem text, placing each object in its proper place, and then reading off the answer to the problem from that model, (b) that a minority instead employ the Method of Elimination, i. e. they try to exclude all objects except one, which is then inferred to be the answer, and (c) that occasional subjects work with yet other methods.

Several think-aloud protocols from these studies have been simulated by computer programs on the form of production systems (PDS:s). Figure 2 shows one PDS induced from a think-aloud protocol. The notation used was introduced by Newell and Simon (1972, p. 44), and is essentially the standard BNF notation. This PDS encodes a data-driven form of the Method of Series Formation. Briefly, the subject first looks for a particular pattern, defined by the test-operator FPP, and then translates the corresponding proposition into an internal model (P5). She then tries to integrate the remaining premises into that model (P3, P4). She has several different heuristics for how to access the problem text. In the beginning, she reads the premises in the order in which they are written, i. e. she reads the first premise (P10), and continues by reading the next premise (P9). After she has begun to construct a mental model, she instead looks

```

P1 <question> <model> ==> ANSW(<question>)
P2 new<FAIL GMO> ==> READ(QUESTION)
P3 new<proposition> <model> ==> FPM(<model>)(=> proposition);
    INT(proposition)
P4 new<model> <proposition> ==> INT(<proposition>)
P5 abs<model> new<proposition>.1 <proposition>.2 ==>
    FPP(<proposition>.1)(=> proposition);
    TRNS(<proposition>.1)
P6 imp<model> ==> BKUP()
P7 new<expression> (REMAINS = NONE) ==> GMO(<model>)(=> object);
    SCAN(object)(= probe);
    READ(probe)
P8 new<expression> <model> ==> SCAN(UNUSED)(=> premise);
    READ(premise)
P9 new<expression> ==> READ(NEXT-PREMISE)
P10 BEGIN ==> READ(FIRST-PREMISE)

```

---

ANSW(x)	Derives the answer to the question x.
INT(x)	Integrates proposition x into the current model.
TRNS(x)	Translates proposition x into a model.
READ(x)	Reads the x part of the problem text.
SCAN(x)	Scans the problem text for the occurrence of x.
GMO(x)	Generates the objects not yet included in x.
FPP(x)	Finds proposition related to proposition x.
FPM(x)	Finds proposition related to model x.
BKUP()	Backs up.

Figure 2. Production system modelling subject SI6 on the Block Problem, with explanation of operators.

for unused information (P8). Finally, towards the end of the process, she instead searches the problem text for information about those objects which have not yet been placed in the model (P7). If she discovers a contradiction between her current result and the givens of the problem, she

backs up (P6). When the GMO operator cannot generate any more missing objects, she reads the question (P2) and derives the answer (P1).

The PDS has been implemented in the production system language PSS, developed by the author (Ohlsson, 1979). The translation from the informal notation used here to a running PSS program was straightforward. The PSS program was run on the same problem as the subject, and solved it in the same way, except for repetitions of inferences.

```

P1  <model> <question> ==> ANSW(<question>)
P2  new(FAIL READ UNUSED) ==> READ(QUESTION)
P3  <model> new<proposition> ==> INT(<proposition>)
P4  abs<model> new<hypothesis> ntc<proposition> ==>
      TRNS(<proposition>)
P5  abs<model> new<proposition> ==> TRNS(<proposition>)
P6  new(FAIL TRNS <proposition>) ==> HYP(<proposition>)
P7  new<expression> ==> READ(NEXT-PREMISE)
P8  new<expression> ==> SCAN(UNUSED)(=> premise);
      READ(premise)
P9  BEGIN ==> READ(FIRST-PREMISE)

```

Figure 3. Production system modelling subject SII4 on the Block Problem.

Figure 3 shows a different PDS, induced from a think-aloud protocol from another subject on the same problem. On a very abstract level, the two PDS:s are similar; they both encode data-driven versions of the Method of Series Formation. However, on a more detailed level, there are several differences. This subject reacts to a failure of the TRNS operator by setting up a hypothesis (P6), which is then used in a renewed effort to apply TRNS (P5). But she has, on the other hand, simpler heuristics for searching the problem text: she either reads the next premise (P7) or scans for unused information (P8).

Several more PDS:s could have been shown, had there been space (see Ohlsson, 1980).

#### A Strategy Grammar

How should one summarize the similarities and differences between such simulation programs? One clue is given by Young (1976, p. 197), who summarized a set of programs for piagetian seriation with the help of a production kit. Such a kit divides the productions in a set of programs into functional groups, all productions within a group performing

essentially the same function, but in different ways. A complete program is assembled by putting together one or more productions from each group. A rather different approach was taken by Newell and Simon (1972, p. 838). They characterized different forms of search with the help of what we might call production schemas from which productions can be generated by replacing meta-variables with specific mechanisms.

```

<strategy> ::= <terminator> <integrator> <translator>
              <<failure-reaction>> <reader> <initiator>
<terminator> ::= <answer-finder> <goal-criterion>
<answer-finder> ::= <model> <question> ==> ANSW(<question>)
<goal-criterion> ::= new<failsignal> ==> READ(QUESTION)
<integrator> ::= <integrate-production> /
                 <integrate-production> <integrator>
<integrate-production> ::=
    <<tag>> <model> <<tag>> <proposition> ==>
    <<test>> INT(<proposition-description>)
<<test>> ::= <test-operator> (<expression>)(=> proposition);
<proposition-description> ::=
    <proposition> / premise / proposition
<test-operator> ::= FPM / FPP
<translator> ::= <translation-production> /
                 <translation-production> <translator>
<translation-production> ::=
    abs<model> new<proposition> <<proposition>>.2 ==>
    <<test>> TRNS(<proposition-description>) /
    abs<model> new<hypothesis> ntc<proposition> ==>
    TRNS(<proposition>)
<failure-reaction> ::= <correction> /
                       <correction> <failure-reaction>
<correction> ::= <backup> / <hypothesize>
<backup> ::= imp<model> ==> BKUP
<hypothesize> ::= new(FAIL TRNS <proposition>) ==> HYP(<proposition>)
<reader> ::= <read-production> /
             <read-production> <reader>
<read-production> ::=
    new<expression> <<expression>> ==>
    <<focusser>> READ(<target-description>)
<focusser> ::= <attention-operator> (=) object;
              SCAN(object)(=> premise); /
              SCAN(UNUSED)(=> premise);
<attention-operator> ::= FOB / GMO
<target-description> ::= premise / <probe>
<initiator> ::= BEGIN ==> READ(<probe>)
<failsignal> ::= (FAIL <operator> <<expression>>)
<probe> ::= FIRSTPREM / QUESTION / NEXTPREM / FIRSTPREM /
           LASTPREM / PREMISE.<n>

```

Figure 4. A strategy grammar for the data-driven form of the Method of Series Formation.

Combining these two ideas, we get the idea of a strategy grammar, i. e. a formal system which is able to generate each member of a set of observed

simulation programs. Figure 4 shows a strategy grammar for the data-driven form of the Method of Series Formation. This grammar can generate the two PDS:s shown above, as well as two other PDS:s induced from protocols delivered by other persons. It is written in BNF notation, with one extension: double brackets, "<< >>", indicate an optional symbol. Briefly, the grammar says that a strategy in this domain is a sequence of components, responsible for the termination of the solution attempt, integration of new information into the current mental model, translation of propositional information into a model format, reacting to failures, reading the problem text, and initiating the solution attempt, respectively. Each component consists of one or more production schemas, each of which can give rise to a specific production in different ways.

Such a grammar can be given different interpretations. According to a weak interpretation, the grammar is simply a convenient summary of the commonalities which recur over a set of simulation programs. It communicates those commonalities in a format which is no less precise than the format used to state the PDS:s themselves. According to a strong interpretation, the strategy grammar is a theory of human performance in this task domain. This interpretation implies that (a) each program derivable from the grammar should correspond to a psychologically real strategy, and (b) that each program observed to be used by some person should be derivable from the grammar. Thus, the grammar should generate the set of psychologically well-formed programs, as it were.

This implies that, given a new think-aloud protocol from the relevant task domain, one should be able to generate a program simulating that protocol from the grammar. From this point of view, the grammar appears as a guide for how to construct a computer simulation in a certain task domain.

```
P1  <model> <question> ==>  ANSW(<question>)
P2  new(FAIL SCAN <object>) ==>  READ(QUESTION)
P3  new<proposition> <model> ==>  INT(<proposition>)
P4  new<proposition> abs<model> ==>  TRNS(<proposition>)
P5  new<expression> ==>  FOB() (= > object);
                               SCAN(object)(= > premise);
                               READ(premise)
P6  BEGIN ==>  READ(QUESTION)
```

Figure 5. A production system derived from the strategy grammar in Figure 4, which does not correspond to any observed subject.

Pertaining to point (a), we can observe that it is possible to derive programs from the grammar in Figure 4 which are different from the programs which gave rise to it. Figure 5 shows such a "synthetic" PDS. This program solves problems through a chaining-heuristic, in which the program

always looks for more information about those objects which have already been placed in the internal model. It also differs from the observed programs in other ways.

According to the strong interpretation of the strategy grammar, the derivation of the PDS in Figure 5 should be seen as a prediction that there exists some person who employs this strategy on these problems - or at least that it should prove possible to teach this strategy to human subjects. No data concerning this prediction are available as yet.

Pertaining to point (b), Figure 6 shows an observed PDS which cannot be derived from the strategy grammar. This program also works with a version of the Method of Series Formation, but it is goal-driven rather than data-driven, which prevents its derivation from the grammar. This fact allows us to say, with a precise meaning, that this program is more different from each of the other PDS:s presented than they are from each other. I. e. the derivability of PDS:s from different grammars impose a similarity metric on a set of simulation programs, allowing us to compare them and to categorize subjects into well-defined groups on the basis of the strategies they use.

Nothing prevents us, of course, from extending the grammar in Figure 4 so as to include the PDS in Figure 6 as well.

### Discussion

One approach to generality in cognitive theories is to concern oneself with system architecture (cf. Newell, 1973; Anderson, 1976). However, hypotheses about system architecture must be complemented with specific simulation programs in order to be interfaced with observations of human performance. Thus, a level of theorizing between the single simulation program and the system architecture is needed. A second approach to generality is to define a general problem solver, and then try to see specific simulation programs as instantiations of that general scheme (cf. Simon, 1975). However, the diversity of cognitive strategies discovered in human data (cf. Quinton & Fellows, 1975) makes this approach rather implausible.

The present approach should rather be compared with another idea, which is similar in intent, but very different in execution. This is the Deductive Analysis proposed by Hagert and Tärnlund (1979), which utilizes the logic programming language PROLOG (see e. g. Lichtman, 1975). In this approach, a set of simulation programs is delimited through an abstract specification, written in first-order predicate logic. Specific simulation programs are then constructed by deduction from the abstract specification, i. e. each simulation program appears as a theorem proved by the standard tools of formal logic. Such a theorem is immediately executable in PROLOG. Deductive Analysis seems a promising approach; however, it restricts the user to a certain programming language, and the derivations are more cumbersome than those of the present approach.

### Conclusions

A level of theorizing between the single simulation program and the system architecture is needed. The idea of strategy grammars was proposed as one



```

P1 (read X) new<proposition> ==> POPP()
P2 (read X) ==> READ(X)
P3 (find UNUSED) new<proposition> ==> POPP()
P4 (find .. X ..) new(<predicate> .. X ..) ==> POPP()
P5 (find <object-sequence>) ==>
    SCAN(<object-sequence>)(= premise);
    PUSH((read premise))
P6 (infer) new<model> <proposition> ==>
    INT(<proposition>)
P7 (infer) new<outcome> ==> POPP()
P8 (infer) new<proposition> <model> ==> INT(<proposition>)
P9 (infer) new<proposition> abs<model> ==>
    TRNS(<proposition>)
P10 (answer (<predicate> ? <object>))
    new(<predicate><object>.2 <object>.1) ==>
    INS((ANSWER IS <object>.2));
    POPP()
P11 (answer <question>) new<outcome> ==> EVL(<outcome>)
P12 (answer <question>) (REMAINS = ONE) ==>
    PUSH((infer));
    PUSH((find UNUSED))
P13 (answer <question>) ==> FOB()(= <object-sequence>);
    PUSH((infer));
    PUSH((find <object-sequence>))
P14 (solve) (ANSWER IS X) ==> SAY(X);
    POPP()
P15 (solve) <question> ==> PUSH((answer <question>))
P16 BEGIN ==> PUSH((solve));
    READ(QUESTION)

```

Figure 6. Production system modelling subject SII2 on the Tool Problem.

approach to this problem. Such a grammar (a) provides a way to present, in a concise form, a set of simulation programs, (b) makes it possible to speak about the degree of similarity between simulation programs, (c) can serve as a device for categorizing subjects with respect to strategy, (d)

imposes discipline on the task of assembling a simulation program for a new protocol, (e) predicts, if interpreted as a theory, which strategies will be found in data from humans and which will not.

### References

- Anderson, J. R. Language memory and thought. Hillsdale, New Jersey: Lawrence Erlbaum, 1976.
- Hagert, G. & Tärnlund, S.-A. Deductive analysis of cognitive processing: A first example. Working Papers from the Cognitive Seminar, Department of Psychology, University of Stockholm, No. 5, 1979.
- Lichtman, B. M. Features of very high level programming with PROLOG. Dissertation, University of London, Imperial College of Science and Technology, Department of Computing and Control, 1975.
- Newell, A. Production systems: Models of control structures. In W. G. Chase (Ed.) Visual information processing. New York: Academic Press, 1973.
- Newell, A. & Simon, H. Human problem solving. Englewood Cliffs, New Jersey: Prentice-Hall, 1972.
- Ohlsson, S. PSS.3 reference manual. Working Papers from the Cognitive Seminar, Department of Psychology, University of Stockholm, No. 4, 1979.
- Ohlsson, S. Competence and strategy in reasoning with common spatial concepts. Working Papers from the Cognitive Seminar, Department of Psychology, University of Stockholm, No. 6, 1980.
- Quinton, G. & Fellows, B. J. 'Perceptual' strategies in the solving of three-term series problems. The British Journal of Psychology, 66, 69-78, 1975.
- Simon, H. A. The functional equivalence of problem solving skills. Cognitive Psychology, 7, 268-288, 1975.
- Simon, H. A. Information processing models of cognition. Annual Review of Psychology, 30, 363-396, 1979.
- Young, R. M. Seriation in children: An artificial intelligence analysis of a piagetian task. Basel und Stuttgart: Birkhauser Verlag, 1976.

# INTERMEDIATE DESCRIPTIONS IN "POPEYE"

David Owen  
Cognitive Studies Programme,  
University of Sussex,  
Brighton, BN1 9QN England.

**ABSTRACT:-** Some ideas are presented, derived from work on the POPEYE vision project, concerning the nature and use of different kinds of intermediate picture descriptions. It is suggested that there are "natural elements" in terms of which stored models should be defined and that it is of prime importance to search for those intermediate picture descriptions which are most characteristic of the expression of such elements.

## INTRODUCTION

The POPEYE project is concerned with the interpretation in the domain of letters and words of pictures of the kind shown in Fig.1 One of the preoccupations of the project has been the identification of those intermediate descriptions of the picture data which best facilitate the interpretation of the scene from which the data has been derived. An intermediate description corresponds to the identification of a picture object. For example in the POPEYE program contiguous collinear sequences of dots are explicitly represented as "line" data-structures, and pairs of collinear and overlapping lines are explicitly represented as "picture bar" data structures. There are many such objects which may be identified in the picture, corresponding to the representation of "objects" and relations between objects in the different domains involved. (The different domains have been discussed in Sloman et al. 1978).

What follows is a discussion of some emerging ideas concerning the significance of different kinds of picture object and their relation to letter models. An attempt is also made to relate these ideas to the analysis of 3-D polyhedral scenes.

## A PARTICULAR VIEW OF LETTERS

A letter is taken to be an abstract object defined by a set of relationships between a number of strokes, themselves abstract objects, and for the current purpose it is only necessary to consider those letters which comprise straight strokes. The important distinction between this kind of description of a letter and a description in some "expressive domain" was made by Clowes (1971). It is also important to distinguish between those characteristics of the representation in the expressive domain which express important properties of the abstract description, and those which are artefacts of the medium of expression.

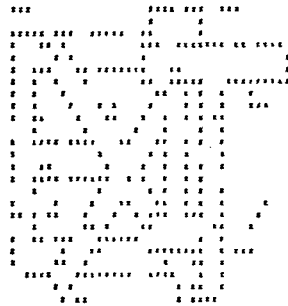


Fig. 1

A 2-D representation of a letter may be regarded as representing two kinds of entity, namely strokes and relations between strokes. Further, it is particular properties of strokes which are of significance and the relationship between two strokes may be described in terms of the values of some simple functions (E.g. difference) defined over the stroke properties. A letter

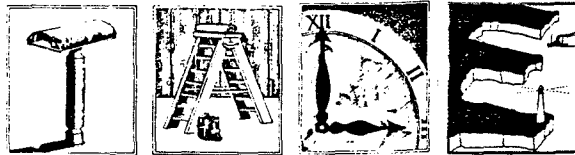
prescribes for a fixed number of strokes the relative values of the stroke properties by specifying the values of the set of functions defined over them. In such a view of letters, a stroke may be regarded as for example an n-tuple of property values including orientation, length, and position of endpoints.

The significance of using this kind of abstract representation of a letter is that it is independent of the way the values of the n-tuple and any consequent relationship with other n-tuple values are represented. Secondly the functions which describe the relations may be continuous, so that in any representation of a relationship between strokes, the accuracy with which it accords with that prescribed in a letter definition may be measured.

#### Letter Depictions:-

What is required of the depiction of a stroke is that it should express a particular n-tuple of properties so that a collection of stroke depictions expresses relative values for the properties which may accord with the definition of a letter. Any picture object from which an approximate major axis can be found will fulfil this requirement and some examples are given in Fig.2, of the different ways in which an axis may be defined.

Fig. 2  
(From:-  
Earnshaw)



The relative values of the properties expressed by a collection of stroke depictions need not conform accurately to those prescribed by a letter definition, for the letter to be recognisable and Fig.2 includes some examples in which the relative values of orientation, length and endpoint positions vary considerably from those of the "ideal" letter they depict. In some of the examples a relationship is not accurately expressed because the corresponding properties are only approximately expressed by the stroke depictions.

#### A PARTICULAR VIEW OF LETTER RECOGNITION

It may be argued that the underlying theoretical framework of a mechanism which is to interpret a picture in terms of letter depictions has two parts. The first is the recognition of instances of the expression of strokes; the second is a search among those instances for sets of strokes for which the relative values of the properties of the set members conform to within acceptable tolerances to those prescribed by a letter definition.

It is the identification of the two types of task in the underlying theoretical framework which is significant for the choice of intermediate descriptions. They separate the two areas in which the expression of two different types of entity have the potential for great variation, giving rise to the variety of ways in which letters may be recognisably represented (Fig.2.). The first entity is the n-tuple of properties which characterise a stroke, and the second the constraints between sets of strokes corresponding to a particular letter.

One of the implications of adopting such a model is for the relative importance of different kinds of picture object which may be found in the picture. Of prime importance are those which capture instances of the expression of a stroke which in the case of the POPEYE domain is Picture Bars, parallel and overlapping pairs of lines. From them values for all the properties of a stroke may be obtained, and the relative values for different strokes may then be used to address letter models.

The other picture objects which are available in POPEYE, for example line junctions, are a manifestation of the expression of a precise relationship between between two such strokes. As such they are vulnerable; small changes in the relationship they express will cause them to disappear, without a similarly large effect on the recognisability of the total letter (See also Brady 1978). Their role then, should be as heuristics for limiting the search for which implicitly expressed relationships between strokes are of significance.

In some sense strokes have a "stand alone" meaning, a junction is the precise expression of a compound meaning.

In the POPEYE program this approach has been exploited to some extent. Initial attempts were strongly influenced by the "linguistic analogy", with line junctions taken as the language primitives, and so ideas revolved around grammars over the kind of objects shown in Fig.3. However, the discovery of Picture Bars is now of fundamental importance, and the evidence in the form of line junctions is used more in a segmentation role. The letter models however are still based on a notion of a letter being composed of junctions between strokes, expressed in the form of line junctions. An alternative model system, based on the above ideas, is being developed.

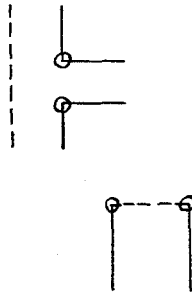


Fig. 3

#### POLYHEDRAL SCENES

A similar argument applied to 3-D polyhedral scene analysis would suggest that properties characterising a surface would be the counterpart of strokes, and that discovering instances of their expression is of prime importance rather than the manifestations of the expression of a precise relationship between them; (E.g. Fork or Arrow junctions).

Returning to the "linguistic analogy" and considering what Becker (1975) had to say gives this vague notion a little more motivation. Briefly, he argues that speech is generated by a process of "stitching together" appropriate elements from a phrasal lexicon according to grammatical rules. However, the flavour of Becker's paper is an attack on linguists as "frustrated physicists" for attempting to establish and use grammars only over the primitives of the language, in an attempt to capture the nature of legal sentences in that language. The "principle" which may be extracted from experience with POPEYE, and would appear to have some relevance to 3-D scene analysis, amounts to generalising that criticism of linguistics into the linguistic metaphor in vision, and in particular making a proposal as to the nature of the vision equivalent of the phrases of Becker's Lexicon.

Some examples of phrases which Becker gives are as follows:

THIS IS NOT TO SAY THAT

WHAT DOES THIS IMPLY FOR

WE MUST CONCLUDE THAT

Each invokes a meaning in its own right all be it incomplete. Only a few such phrases are required to generate a meaningful sentence, compared with the number of language primitives in the sentence, and this is an important part of the motivation given by Becker for his ideas. The implication is that it is unnecessarily difficult to generate sentences from primitives of the language all the time, that the art which is language acquisition is about learning new phrases and how to "stitch them together" to convey the desired meaning, and that the resulting utterance may be understood in the same way.

The problem with drawing analogies in vision is that it is not obvious what the primitives of the language are (edge features? lines? line-junctions?) and consequently what constitutes a meaningful phrase is equally unclear. In the work of Huffman(1971) and Clowes(1971) in some sense lines are taken as the primitives and line-junctions seem intuitively the most obvious candidates to choose as phrases which include several instances. The intuition arises partly because the affinity between lines is manifest in a most concrete way - they actually touch. Comparing them with Becker's phrases, do they mean anything in their own right?

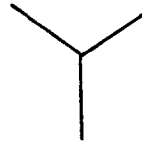


Fig. 4

For example consider the FORK junction in Fig. 4 Clearly in the polyhedral domain this can be taken to "mean" the corner of a cube. However compare this with the following sentence:-

THIS IS NOT TO SAY THAT / ALL MEN HAVE / HAPPY LIVES

which can be taken as comprising three phrases.

Now consider the following three words:-

NOT - MEN - HAPPY

Together they capture more of the meaning of the sentence than any one of the phrases alone because the structure comprising the three words in order captures some minimal part of the meaning of each of the three phrases. The structure is not of much general use in constructing or analysing sentences since it is a characterisation applicable to only a few sentences. More importantly, unlike each of the three phrases it is ungrammatical. The suggestion here is that the line-junction of Fig. 4 is more closely identifiable with the three word structure than with a lexical phrase, since together the lines capture some part of the nature of the three surfaces and how they relate, and that the surfaces are better candidates for being the parallels of Becker's phrases.

To continue the comparison, in the same way that only a few phrases are needed to generate a meaningful sentence, only 3 faces of a cube are visible

compared with 7 linejunctions. Finally, Becker suggests that speech is generated by "stitching together" appropriate phrases, and it is interesting to note that most people when asked to draw a cube, complete surfaces rather than vertices, i.e. typically a sequence like (a) rather than (b) in Fig. 5.

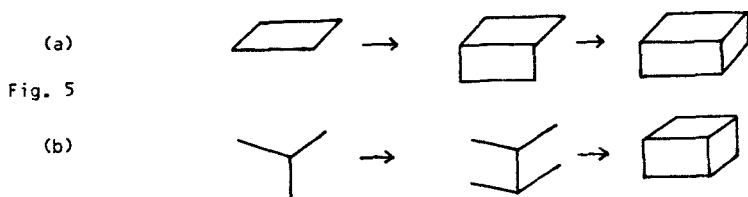


Fig. 5

To return to the comparison with the letter domain, the argument is that surfaces have a "stand alone" meaning and their juxtaposition expresses a compound meaning as an object. Some junctions are particular manifestations of the precise expression of a relationship between two surfaces and as such may or may not capture the compound meaning (E.g. Fig. 6) and are not suitable primitives from which to construct an object model.

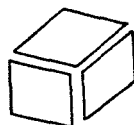


Fig. 6

## CONCLUSION

Regarding all vision as involving the addressing of stored models, it is suggested that models should be defined in terms of relationships between "natural elements" which have meaning in their own right, rather than in terms of objects derived from the manifestation in the picture of relationships between such elements. This implies that it is of prime importance to search for those picture objects which are most characteristic of the expression of such "natural elements". It does not imply that other picture objects cannot be exploited, but rather that their usefulness lies in what they imply for the relations between the "natural elements".

## ACKNOWLEDGEMENTS

I wish to express my thanks to friends at Sussex for all I have learned from them. Conversations with Steve Draper and Frank O'Gorman have been particularly helpful. Special thanks go to Aaron Sloman for valuable criticism, suggestions and ideas and most of all encouragement. I am also grateful to Anthony Earnshaw and publishers J. Cape Ltd. for permission to include Fig. 2. This work was supported by the U.K. Science Research Council.

References

- Becker J. 1975 "The phrasal lexicon" B.B.N. rep. 3081. A.I. report no. 28  
(Cambridge, Mass. Bolt, Berenek, and Newman)
- Brady M. 1978 "The development of a computer vision system" *Psicologica Recherche*.
- Clowes M.B. 1971 "On seeing things" *Artificial Intelligence*, vol.2 no.19.
- Earnshaw A. "Seven Secret Alphabets" Jonathan Cape. 1972.
- Huffman D.A. 1971 "Impossible Objects as Nonsense Sentences". *Machine Intelligence 6*. ed.Meltzer B. and Michie D. (Edinburgh University Press).
- Sloman A. et al. "Representation and Control in Vision" in *Proc. A.I.S.B./G.I Conf.* 1978.



UNDERSTANDING ENGLISH DESCRIPTIONS OF PROGRAMS

Allan Ramsay  
Dept. Of Artificial Intelligence  
University Of Edinburgh  
Edinburgh, SCOTLAND EH1 2QL

1. ABSTRACT

A considerable amount of work has been done on verifying that computer programs fit their specifications. However, providing formal specifications is itself a difficult and tedious task, so that programs are generally only documented incompletely and imprecisely. This paper presents a computer system which accepts English descriptions of procedures and relates them to LISP programs that are supposed to implement them. This system is intended to illustrate how "informal" techniques may be used to provide a rough analysis of a program for which incomplete specifications are provided.

Keywords - automatic program verification, symbolic evaluation,  
natural language programming

2. WHAT DOES SH4 DO ?

The system SH4 described in this paper is used for relating English descriptions of procedures and LISP programs that are supposed to implement those procedures. What "relating programs and procedure descriptions" means here is that the system creates two sets of records to represent the description and the program and then suggests and investigates links between records in the two sets. It is recognised that it is not very useful to provide a user with quite large sets of records with links between them, saying "Here you are, this is how your program works." Hence, when the system has built up these sets, it uses them to derive a copy of the object program annotated with comments derived from the description, and a set of flow charts representing the described procedures, with fragments of code attached to show how the procedures are implemented.

The inputs that the system accepts for a very simple program and the output that it generates are given in Figs. 2.1 - 2.4. This example is not the limit of the system's ability to deal with programs, it is the limit of what can be shown in a paper this length.

There is a procedure called testmember. This procedure takes an object called target and a set called pool. If the set is empty then the procedure returns false. If not then it gets an object called testobj from the set. If testobj and target are equal then testmember returns true. If not then it performs testmember on target and the rest of pool.

Figure 2-1: Input Procedure Description

```
[DEF TESTMEMBER
  [TARGET POOL]
  [COND [EQUAL POOL NIL]
        FALSE
        [EQUAL TARGET [CAR POOL]]
        TRUE]
  [TESTMEMBER
    TARGET
    [CDR POOL] ] ] ]
```

Figure 2-2: Input Program

```
comment TESTMEMBER implements testmember
comment TARGET represents target
comment POOL represents pool

[ DEF
  TESTMEMBER
  [TARGET POOL]
  [ COND
    [EQUAL POOL NIL]      if set is empty
    [FALSE]              then testmember returns false
    [ EQUAL              if target and testobj are equal
      TARGET
      [CAR POOL] ]       then it gets object from set
    [TRUE]               then testmember returns true
    [ TESTMEMBER         then it performs testmember on
      TARGET             target and rest
      [CDR POOL] ] ] ]   rest of pool
```

Figure 2-3: Commented Version Of Program

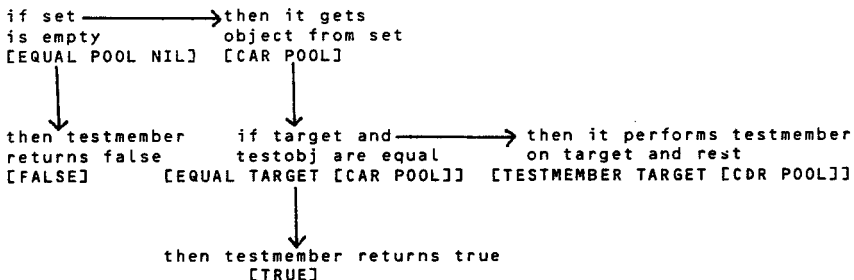


Figure 2-4: Flowchart Illustrating Implementation

### 3. THE SYSTEM

The system starts work by interpreting the English description as follows. The text is dealt with sentence by sentence. Each sentence is parsed by a fairly orthodox top-down left-to-right fastback parser - see [Ramsay 1980] for details - and then the parsed structures are interpreted by choosing and instantiating schemes from a fixed, predefined set to represent the items referred to by the noun groups and possibly the main verb group of each clause, and applying a set of rules associated with the verb of the main verb group to construct links between the various instantiated structures. This process builds up a network of records of various fixed types, connected by pointers from slots in records to other records. A similar network is built to represent the object program. Given these networks, the problem of relating English sentences to LISP expressions becomes a matter of suggesting links between items in the two sets of records.

In this context, where we have two unconnected sets of records representing the procedures we are interested in and the program that we hope implements them, it is impossible to make any use of program verification techniques such as recursion induction [McCarthy 1960] or structural induction [Burstall 1968] to show that the program fits its specifications - the whole problem is that at this point we do not know which parts of the program are supposed to fit which specifications. Similarly, it is hard to see how the system could use any form of symbolic evaluation [Sussman 1970], [Goldstein 1974] to find out what the various parts of the program are supposed to be doing, since this technique is only really useful if you can describe what the environment is expected to look like; and again, we simply do not know this yet.

#### 3.1. Making Guesses

In order to find out enough about a program for symbolic evaluation to be useful, we use a collection of "hypothesisers" - routines that use superficial characteristics of programs and procedures to suggest links between them. These hypothesisers use ideas such as looking for recursive functions in the program to link with recursive procedures in the description, or looking for non-atomic data structures in the program to link with non-atomic objects referred to in the description. Several of them use links suggested by others, e.g. by connecting a procedure and a function on the grounds that the inputs to the procedure have already been linked to the arguments of the function. They are all fairly unreliable, so the system has to be able to cope with cases where items have been linked incorrectly. However, the only part of the system that can check links, the symbolic evaluator (see Section 3.2) cannot be invoked until links have been suggested for most of the procedure set and the program, and even then it can only check links between procedure descriptions and function definitions - it cannot say anything about hypotheses linking objects and data structures, etc. This makes it difficult to maintain

a useful, coherent tree showing the dependencies between hypotheses. Instead, we allow the system to build a set of hypotheses without keeping track of how they were derived, and without worrying about whether they are correct or even whether they are consistent. The interactions of the hypothesisers are controlled by a scheduling algorithm derived from ideas used in vision and speech understanding systems [Sloman et.al. 1978],[Erman & Lesser 1975],[Woods 1976]. The scheduler makes sure that hypothesisers that use subsidiary hypotheses behave sensibly, and it also notices when sufficiently many links have been suggested for analysis via symbolic evaluation to be practical.

When this happens, the scheduler invokes a module that checks hypotheses by evaluating the code along "paths" [King 1969] through functions (a function is split into paths by considering the possible sequences of expressions that may be evaluated when it is called; loops are dealt with by turning them into conditional expressions with virtual recursive function calls on their main branches). In SH4, the snapshot environments described by the symbolic evaluator are defined in terms of actions that have been performed. The system tries to deal with function calls by considering them as single complete actions and looking for procedure calls that have already been connected with them through hypotheses. This is similar to Ruth's [1976] approach of matching "actions" in a description of a general algorithm against expressions in a particular object program, with the difference that Ruth's system knew exactly which action it was trying to match against each expression, whereas SH4 only has hypotheses about which procedure call a function call is supposed to implement.

#### 4. SUMMING UP

SH4 is seen as a complement to systems that check program correctness by proving assertions that must always (or possibly sometimes [Manna & Waldinger 1976]) be true at various points in the execution of the program. These systems can be awkward to use, as they require the programmer to provide precise specifications for his program, which can sometimes be hard and is always time-consuming. They also require considerable theorem proving powers. Gerhart [1979], for instance, describes a system which recognises that the deductions needed for verifying significant programs are too complex for contemporary theorem proving systems to deal with in an acceptable time, and hence works in conjunction with the user by applying deductive rules as he suggests, and maintaining for him a tree that shows what steps were taken where and what the result of taking them was.

SH4 is intended to show how to use less detailed, less precise descriptions of programs to get an overall picture of what is supposed to be going on, either for its own sake or for use with a formal program verification system. The procedure descriptions that the system works with are, unfortunately, too detailed for it to be a practical tool as it stands. Furthermore, there are a number of

important aspects of programming that it does not deal with at all, notably the definition of new types of data structure. Nonetheless, it does indicate how programs may be analysed in terms of informal descriptions, and it has been used on several non-trivial programs (e.g. a simple garbage collector and a package of standard set manipulation functions).

#### REFERENCES

- Burstall, R.M. Proving Properties Of Programs By Structural Induction  
Exptl.Prog.Report 17, Edinburgh 1968
- Erman, L.D. & Lesser, V.R. A Multi-level Organisation For Problem Solving  
Using Many Diverse Cooperating Sources Of Knowledge  
Carnegie-Mellon 1975
- Gerhart, S.L. (ed) AFFIRM Collected Papers  
USC Inf.Sci.Inst. 1979
- Goldstein, I. Understanding Simple Picture Programs  
Ph.D. thesis, MIT 1974
- King, J.C. A Program Verifier  
Ph.D. thesis, Carnegie-Mellon 1969
- Manna, Z. & Waldinger, R. Is "Sometime" Sometimes Better Than Always In  
Proving Program Correctness ?  
Stanford AI memo 281, 1976
- McCarthy, J. Correctness Of A Compiler For Arithmetic Expressions  
CACM 1960
- Ramsay, A.M. Understanding English Descriptions Of Programs  
Forthcoming Ph.D. thesis, Sussex 1980
- Ruth, G.R. Intelligent Program Analysis  
Artificial Intelligence 7, 1976
- Sloman, A., Owen, D., Hinton, G., Birch, F. & O'Gorman, F.  
Representation And Control In Vision  
Proc. AISB/GI Conf., Hamburg 1978
- Sussman, G.J. A Computer Model Of Skill Acquisition  
Ph.D. thesis, MIT 1970
- Woods, W. Final Report To ARPA On Speech Understanding  
BBN Report, 1976

..

PARSING ENGLISH TEXT

Allan Ramsay  
 Dept. Of Artificial Intelligence  
 University Of Edinburgh  
 Edinburgh, SCOTLAND EH1 2QL

1. ABSTRACT

This paper presents a technique for parsing English text according to a grammar specified as a set of rewrite rules. The paper describes a compact way of representing such a grammar and presents a program which uses this representation to parse text without backtracking and without repeating work that it has already done.

Keywords - deterministic parsing, control structures

2. REPRESENTING SETS OF REWRITE RULES

The program described in this paper deals with grammars specified as sets of rewrite rules, with properties of the main structure and its constituents described by values for sets of labels. This formalism is very similar to the Direct Clause Grammars described in [Pereira & Warren, 1979]. It is possible to represent a grammar specified as such a set of rewrite rules in a compact manner that makes it easy to delay decisions about which rule applies to a piece of text until enough of it has been parsed for the correct rule to be chosen. If we look at a rule as a description of a path through a piece of text, it is evident that we do not want to choose which of two rules describes the correct path through a given piece of text until the two diverge. For instance, consider a grammar that contains the following two rules.

```
vgroup [tense ?t voice !actv]
    <= aux [atype !be tense ?t] participle [tense !present]

vgroup [tense ?t voice !pass]
    <= aux [atype !be tense ?t] participle [tense !past]
```

Figure 2-1: Two Similar Rules

where the first rule describes the first verb group in the sentence "I was walking home on Saturday night when I was beaten up again" and the

second rule describes the second verb group in this sentence. The right hand sides of these rules are identical apart from the final set of labels, so that it is not possible to find out which of them applies until the participle has been found and studied. However, if their similarity is recognised when they are read in, they may be represented by the following tree.

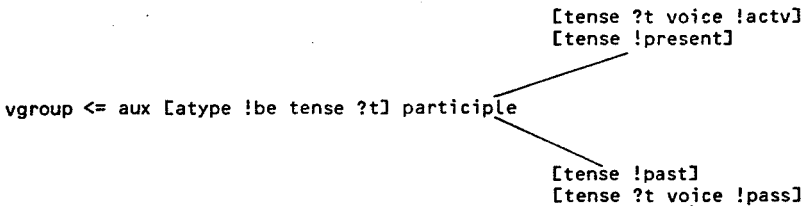


Figure 2-2: Combination Of Two Rules Into One

Using this rule, NRP has no need to decide which branch to take until it has found an auxiliary and checked its type, and has found a participle and is ready to look at its labels.

Thus by representing the rules of a grammar in such a way that similar rules are combined, the system is able to defer choosing between rules until it has available to it the information that it needs to make the decision.

This technique, which is similar to GSP's treatment of ambiguities in the parse tree itself [Kaplan 1973], enables the system to act "deterministically" once it has chosen a tree representing what was originally a family of rules. However, there is no guarantee that all the rules describing the decomposition of some type of structure will form a single family. In order to see how the system copes with the uncertainty that is introduced when it has to build a structure for which it has several trees of rules, we will have to consider its control structure.

### 3. CONTROL WITHIN NRP

A common strategy for top down parsers is for the parser to call itself recursively, asking for constituent structures, until it has built up a complete legal structure or found that it cannot do so, and to return either the structure that it has built or a flag saying that it has failed to build anything [Woods 1970], [Winograd 1972], [Kay 1973], [Ramsay 1980]. This technique leads to problems when it is not

clear what sort of structure is required as the next constituent of the current partially built structure, particularly if it is possible for a structure to be constructed and returned incorrectly. Parsers that follow this strategy sometimes build the same intermediate structure several times, if it occurs as a constituent of several structures that are partially built and then abandoned, and they generally have to keep a trace of (at least some of) their decisions and the state of the parse at the time when the decisions were made, so that they can backtrack when things go wrong.

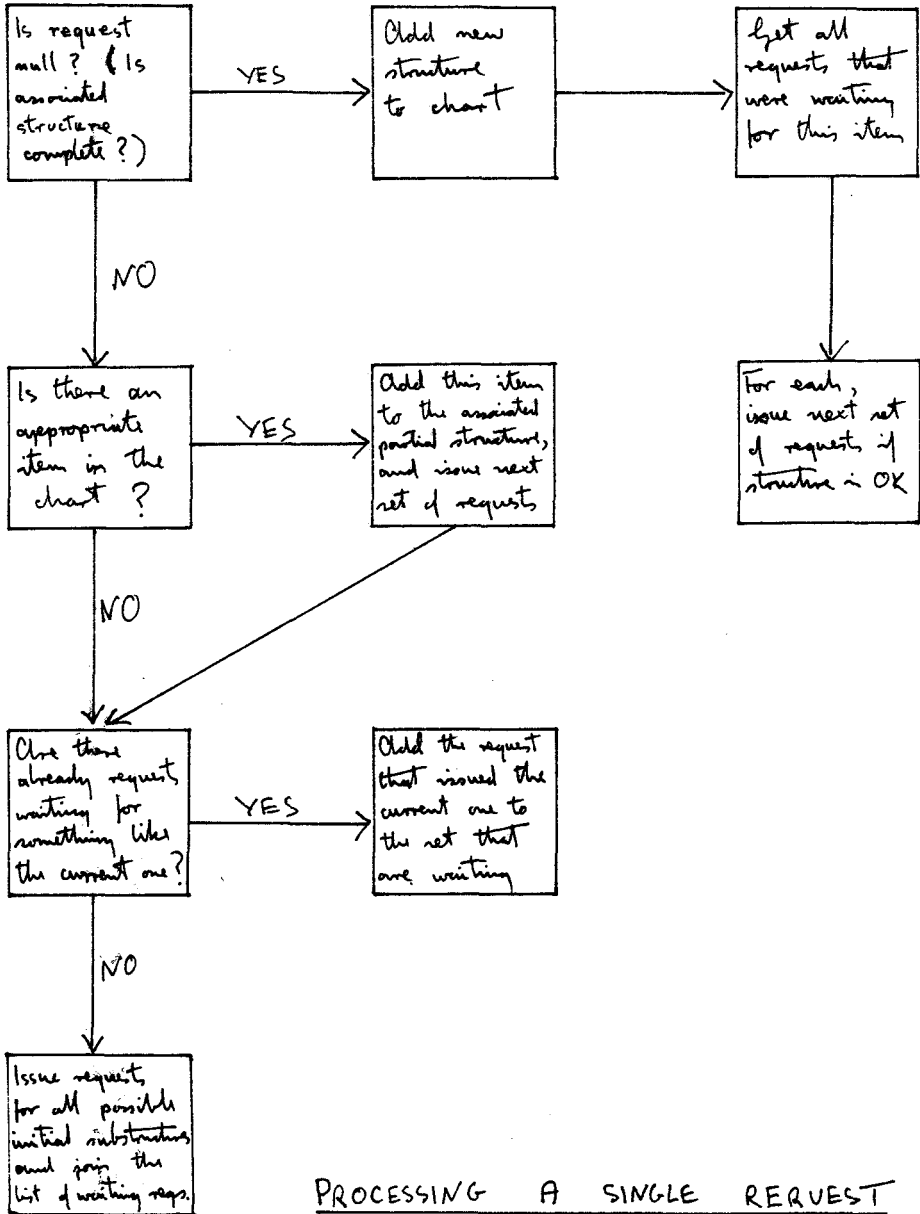
NRP's strategy, on the other hand, is as follows. It starts with a "request" for some sort of structure, where a request consists of the following parts:- the partially built structure to which the requested structure is to be added, the tree of rules that is to be used for building the new constituent, the position in the sentence that the structure is to start at, and a list of all the requests that are waiting for the partially built structure to be completed.

When such a request is issued, the system looks through two stacks of requests that it maintains. One of these stacks contains requests that are waiting to be processed and the other contains ones that have been processed and are waiting for a constituent to be built. If the system finds, in either of these stacks, any request which has the same position and same tree of rules as the given one, it simply adds the higher level request that initiated the current one to the list of requests that are waiting for the one that was found in the stack. Only if nothing that matches the current request is found in either of the stacks is it added to the stack of ones that are waiting to be processed.

The system then takes the first request from the stack of ones that are waiting to be processed. It looks at the "chart" [Kay 1973] that contains all the structures that have been built so far to see if it contains anything suitable; if it does, then this is added to the partial structure contained in the current request and a new set of requests is issued for all the possible ways of continuing the new partial structure. Otherwise, requests are issued for all the possible initial constituents of the currently requested structure. In either case, any new request is treated as described above.

Eventually the parser will process a request whose tree of continuations is null. This indicates that the partial structure associated with the request is complete. In this case, the now complete structure is added to the chart. In addition, the requests that were waiting for the current one to finish are accessed; for each of them, if the new structure has features that fit the request's rule then a new partial structure containing the request's old partial structure and the newly completed one is constructed, and requests for items extending this new structure are issued.





PROCESSING A SINGLE REQUEST

4. SUMMING UP

The above algorithm may sound complex and liable to lead to combinatorial explosion, with enormous stacks of requests being built up. However, the representation of sets of rules as trees provides a strong constraint on the number of requests that will be made at any point. Furthermore, the search through the stacks for similar requests whenever a new one is issued guarantees that no intermediate structure will be built more than once; the fact that all possible requests are always issued means that it is not necessary to keep track of decisions and the state of processing when those decisions are made - any legal parse will necessarily emerge, while wrong decisions will not have any ill effects; and finally, since the routine that processes requests is not recursive, it is possible to write efficient code to implement it.

REFERENCES

- Kaplan, R.M.      A General Syntactic Processor  
in "Natural Language Processing", ed. R. Rustin 1973
- Kay, M.            The MIND System  
in "Natural Language Processing", ed. R. Rustin 1973
- Marcus, M.        A Theory Of Natural Language Understanding  
MIT Working Paper 1979
- Pereira, F. & Warren, D.    Definite Clause Grammars Compared With ATN's  
DAI Research Paper 116, Edinburgh 1979
- Ramsay, A.M.      Understanding English Descriptions Of Programs  
Forthcoming Ph.D. thesis, Sussex 1980
- Warren, D.        Implementing Prolog  
DAI Research Reports 39 & 40, Edinburgh 1977
- Winograd, T.      Understanding Natural Language  
MIT Ph.D. Thesis 1972
- Woods, W.        Transition Network Grammars For Natural Language  
Comm. of the ACM 13, 1970

## The fuzzy set fallacy

Peter Schefe  
 Fachbereich Informatik  
 Universitaet Hamburg  
 D 2000 Hamburg 13

## Abstract

"Fuzzy set theory" and "fuzzy logic" have been proposed to be useful for applications in pattern recognition and artificial intelligence. It is argued that the concept of a "fuzzy set" is due to an intuitive fallacy induced by a threshold probability distribution. A sketch of an alternative model for dealing with applications considered to be "fuzzy" ones so far is presented. The problem of handwritten character recognition is used as an example.

Keywords and phrases: knowledge representation, reasoning with vague concepts, recognition of handwritten characters, concept learning.

## 1. Introduction

Fuzzy set theory initiated by L. Zadeh [1965] has been proposed as an useful tool for artificial intelligence and pattern recognition. Its main advocate goes so far as to claim "that once it (fuzzy logic P.C.) is understood, it will be widely adopted." [Zadeh, 1989]. I doubt that. On the contrary, I believe that all issues addressed by fuzzy set theory can be handled within the framework of classical logics and probability theory. However, anyone who doubts the intuitive base for fuzzy sets theory should provide with an alternative model useful for practical application. Accordingly, I shall give

- an explanation of what I should like to call the "fuzzy set fallacy"
- give an outline of an alternative model
- give a sketch of a procedure for dealing with the phenomena considered as "fuzzy" ones so far, using the recognition of handwritten characters as an example.

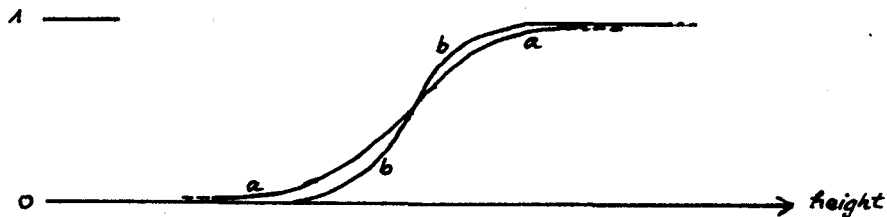
## 2. The roots of the "fuzzy set fallacy"

Let's briefly recap the main idea of a "fuzzy set". In everyday as well as in scientific cognition, it is often difficult to decide, whether a certain individual belongs to a class defined by a predicate or its scope withit is rather difficult if not impossible to give a precise extensional definition of a predicate. There are always borderline cases. L. Zadeh tried to cope with both phenomena by creating "the dialectical synthesis of continuously graded degree of membership to a set... a natural generalisation of the characteristic function..." [1976].

What is the fallacy in this seemingly nice idea? Firstly, Zadeh tries to capture a fundamental uncertainty as to membership by an arbitrarily precise continuous membership function. How can we be sure that uncertainty starts at a certain point and ends at a certain point on a scale? How can we be sure that the membership degree is exactly, say, .7 on a certain point on the scale? If we fix the borderline cases this way, we have to answer the question about the uncertainty of the boundary of the borderline. This leads to an infinite regress. Thus, we fail to capture uncertainty at all.

A well understood means for describing uncertainty adequately is probability theory. Let us look at the problem this way using the familiar example of "tall men". If we take the physical measuring scale for height, it is impossible to locate the boundary between "tall" and "not tall" with certainty. The reasons for this are, shortly, the impossibility to recognise height to an arbitrary degree of precision on the one hand, and, on the other hand, the lack of an explicit definition, which would be required for a scientific experiment, but is useless in everyday life because of the huge variety of contexts. However, if people are forced to make a yes-no-decision - and mostly they force themselves to do so - their behaviour will not be consistent, but probabilistic.

Asking people to indicate the boundary on the scale of height in a sufficiently restricted context would typically yield a Gaussian distribution. Asking the more natural question "Is this man tall?", and showing a sample of men to a sample of subjects should be expected to yield the cumulative normal distribution, theoretically. Obviously, each subject considering the boundary to be to the left of the height value represented by a certain man will answer "yes". Experiments of this sort were conducted by Hersh and Caramazza [1976] yielding the expected distribution. Unfortunately, Hersh and Caramazza being inspired by fuzzy set theory interpreted the proportion of "yes"-responses for a sample object as "grade of membership". However, their experimental results are in favour of a probabilistic interpretation.



Typical theoretical (a) and empirical (b)  
agreement probability distributions

The objective probability distribution of verbal behaviour is reflected in some way as a subjective probability distribution in

the individuals (see below). In other words, subjective uncertainty pertaining to the applicability of a vague predicate is fallaciously interpreted as an objective degree of membership. This is the root of the "fuzzy set fallacy", i.e., the intuitive appeal of the concept of a continuous membership function.

### 3. Outline of an alternative theory

The basic idea or postulate is that subjective uncertainty is a matter of degree, and that it can be measured. Although such a measure can be defined within the framework of probability theory [Scheffe, 1979], the subjective "behaviour", i.e., evaluation of evidence in a certain context, as to compound statements is different from "ordinary" probability theory, e.g., the max- and min-operators apply to disjunction and conjunction of "agreement events", respectively. We are assuming that a reasonable person will "behave" according to this model. Primarily, these measures are attached to factual propositions, i.e., they are epistemic indicating a degree of belief in the truth or falsehood of a proposition. Fuzzy set theory, however, is aiming at providing a semantics of vague predicates. This is a different issue. In the agreement-probabilistic model, vagueness is interpreted as subjective uncertainty pertaining to the applicability of a predicate in a certain context ("Is 'tall' applicable to John?"), and not to the truth of the proposition containing this predicate ("Is John tall?"). The difference could also be described by the contrasted pair "assertional - definitional". Definitional uncertainty must not be confounded with semantics. However, it has to be coped with in AI-systems.

I would state the hypothesis that the cumulative probability distribution exhibited by the observable linguistic behaviour is similar to the subjective certainty distribution generated by self-assessment, and that differences are due to thresholding.

### 4. Do we need fuzzy set theory for applications?

The answer to the above question is clearly "No, on the contrary, because it may be misleading". However, we need a special probabilistic application model for subject areas that are considered "fuzzy", intuitively. E.g., pattern recognition of handwritten characters should not be dealt with by the same procedures which are used in the recognition of natural objects. Instances of handwritten characters cannot be considered as possible members of a class X, but as objects that can possibly be ascribed the meaning of "being the character 'X'", i.e., of being inside the tolerance space represented by 'X'. This is a crucial difference.

Consider two instances of handwritten "H" and "A", which may be physically almost undistinguishable in isolation but interpreted correctly in an appropriate context. Thus, a contextfree recognition procedure should not be based on parameters derived from the statistical distribution of the whole pattern or its features. Because letters are meanings, there are contrastive pairs in one or more than one dimension. Thus, uncertainty is not about stochastic deviations from an ideal geometric pattern

(if it is existing at all), but about borderline cases. E.g., there is no uncertainty of "being a 'H'" with instances, in which the two lines are connected in the "typical" way but varying in length or proportion of lengths. However, inclination is a crucial aspect, because there is the contrastive 'A'. Hence, what is needed is a probabilistic decision procedure for borderline cases in the various dimensions ('H' and 'N' are also contrastive e.g.). Since Zadeh's [1976] "fuzzy membership functions" conceived of on purely intuitive grounds is a good approximation to the cumulative normal curve, pattern recognition procedures based on this concept are likely to be successful. However, this is not meant to be a success of fuzzy set theory.

##### 5. Sketch of a "fuzzy" decision and learning procedure

The framework for solving the problem of "fuzzy" pattern recognition is given by the linguistic approach, especially the semantic view as outlined by Clowes [1971], and the learning paradigm of the "near miss" proposed by Winston [1975]. I use the example of recognizing the letter "T". Firstly, the letter is parsed syntactically into two strokes, A, and B, say. Secondly, these elements join into certain semantic relations, and have to meet certain semantic restrictions, e.g., the strokes have to be connected, they have to be straight, the angles have to be equal, etc. Using the programming language FUZZY [LeFaivre, 1977], these predicates can be interpreted as patterns invoking DEDUCE-procedures, which compute the probabilities that "being equal", "being straight" can be applied to the instances, e.g.:

```
((STRAIGHT A) . 0.7)
((STRAIGHT B) . 0.9)
((EQUALANGLE AB BA) . 0.9)
etc.
```

How can these probabilities be learned? Firstly, the program may be given a description representing the "ideal" "T" or a prototype, from which it can generate the description. To get the range of possible expressions of a "T", "near misses" are presented to the program. For example, we may have a difference of angles of 30 as a near miss of "being equal". Then, the program generates a heuristic probability function assigning the value of .5 to 30, and, similar to the approximations, as used by Siy and Chen [1974] or Kickert and Koppelaar [1976], assigning .1 to 0, and 0. to 60, say.

How shall one deal with the probabilities assigned to features, in order to get the probability for the pattern? There is no point to take the product for the resulting probability, as is done in the Bayesian approach to class assignment. Instead, to take a conservative decision - of being inside the tolerance space indicated by the near misses - I will commit myself only and exactly to the minimum of the probabilities of being inside the tolerance space in one dimension. In other words, the probability of error is at worst the maximum of the probabilities of being outside the space in one dimension.

On the other hand, if there are alternative spaces indicated by alternative features, I can commit myself to the maximum of the probabilities of being in one of the spaces. In this case, the probability of error is at worst the minimum probability of being outside one of the spaces. Thus, the min and max operators of "fuzzy sets theory" have a well founded probabilistic base.

## REFERENCES

- M.B. Clowes. On seeing things. Artificial Intelligence, 2, 79-116, 1971.
- D.R. Gaines. Foundations of fuzzy reasoning. Int. J. Man-Machine Studies, 8, 623-668, 1976.
- H. M. Hersh, and A. Caramazza. A fuzzy set approach to modifiers and vagueness in natural language. Journal of Experimental psychology: General, 105, 254-276, 1976.
- W.J.M. Kickert, and H. Koppelaar. Application of fuzzy set theory to syntactic pattern recognition of handwritten capitals. IEEE Transactions on System, man, and Cybernetics, 6, 148-151, 1976.
- R.A. LeFaivre. FUZZY reference manual. Rutgers University, Computer Science Department, 1977.
- P. Scheffé. On foundations of reasoning with uncertain facts and vague concepts. Bericht IFI-III-B-56/79, Fachbereich Informatik, Universität Hamburg, 1979. (Also to appear in Int. J. Man-Machine Studies).
- P. Siy, and C.S.Chen. Fuzzy logic for handwritten numeral character recognition. IEEE Transactions on Systems, Man, and Cybernetics, 4, 570-575, 1974.
- P.H. Winston. Learning structural descriptions from examples. The psychology of computer vision, P.H. Winston, Ed., New York: McGraw-Hill, 157-210, 1975.
- L.A. Zadeh. Fuzzy sets. Information and Control, 8, 338-353, 1965.
- L.A. Zadeh. A fuzzy-algorithmic approach to the definition of complex or imprecise concepts. Int. J. Man-Mach. Stud. 8 (1976), 249-291.
- L.A. Zadeh, Individual position statement, SIGART 70 (1980), Special issue on knowledge representation, 48-49.

WHY VISUAL SYSTEMS PROCESS SKETCHES

Aaron Sloman and David Owen [\*1]  
 Cognitive Studies Programme,  
 School of Social Sciences,  
 University of Sussex,  
 Brighton, BN1 9QN, England

Abstract

Why do people interpret sketches, cartoons, etc. so easily? A theory is outlined which accounts for the relation between ordinary visual perception and picture interpretation. Animals and versatile robots need fast, generally reliable and "gracefully degrading" visual systems. This can be achieved by a highly-parallel organisation, in which different domains of structure are processed concurrently, and decisions made on the basis of incomplete analysis. Attendant risks are diminished in a "cognitively friendly world" (CFW). Since high levels of such a system process inherently impoverished and abstract representations, it is ideally suited to the interpretation of pictures.

1. Is the study of impoverished pictures relevant to 'real' vision?

AI vision work concerned with pictures, including digitised photographs, straight-line drawings and cartoons, etc. has recently been criticised as irrelevant to visual perception of objects in the environment, Clocksin [1978]. Related themes can be found in Horn [1978]. It can be argued that studying impoverished pictures with great local ambiguity leads to overemphasis on top-down, knowledge-guided visual processes, as in Shirai [1975] and Minsky [1975], and on complex control structures, as in POPEYE [\*2]. Lack of detail in artificial images causes difficulties of interpretation which, it may appear, do not arise in ordinary perception, where disambiguating detail is provided by colour, stereopsis, optical flow, etc. Admittedly images interpreted by most A.I. programs lack many features available even in monocular perception of static scenes, from which useful information can be extracted with powerful algorithms and computational resources. [Marr 1976, Horn 1978 and papers cited therein]. Horn's claim: 'we may have closed our eyes to the raw image for too long', is reasonable, and supported by his own excellent work on images. But we mustn't now close our eyes to all else.

Extraction of low level image and scene features is but a sub-process of the visual mechanism. That a powerful subsystem is normally used does not imply that it is essential for vision. Stereopsis certainly occurs, and needs to be explained, but our ability to perform everyday tasks with just one eye also needs explanation. Similarly, we can often recognise things when detail is missing, or spurious information added, through poor visibility, eye defects, strong back-lighting, restricted view angle, or intervening shrubbery. Normally, we use perceived detail to segment the scene into objects, but sometimes the grouping must go beyond consideration of image continuities and discontinuities because of occlusion of some objects by others, camouflage, shadows or spurious juxtapositions. All this suggests considerable modularity: various sub-systems produce information, perhaps partly duplicated by other sub-systems, and less precise information may suffice if the ideal is not available. This modularity could allow a component which ideally should be driven by the data, to be driven instead by prior knowledge activated by other data. This might explain both our facility with impoverished pictures and the occurrence of misperceptions even in excellent conditions (well-known examples



are the hollow mask [Gregory 1970], and the triangle containing "PARIS IN THE SPRING").

How can we function so well when so much detail is lost? Recognition of sketchy drawings can be rapid and effortless [Hochberg 1978, page 193, citing Ryan and Schwartz]. Perhaps, when we look at pictures, intermediate results of the interpretation process are similar to some intermediate (sketchy) results of the processes of normal perception? Perhaps normal perception uses mechanisms with built-in characteristics designed to cope with abnormal, specially difficult, situations? Our central idea is that visual systems process many different domains of structure in parallel. So analysis of relatively "high-level", abstract, incomplete, representations, sometimes occurs in parallel with detailed analyses of visual data. [\*3] Higher level processes would then be driven in part by prior knowledge of specific sorts of objects (e.g. generalised cylinders, humanoid figures), lower levels mainly by very general (implicit) knowledge about 3-D surfaces, lighting, motion, etc. Occasionally such high-level processes would reach conclusions which are overturned by more detailed analysis, e.g. the "double take". However the different processes would normally produce compatible results, making possible the modularity referred to above. How?

A basic assumption is that the visual system has evolved to work in a "Cognitively Friendly World", a CFW, (which may be very unfriendly in other respects). Here are examples of cognitive friendliness:

- (A) The optic array is rich in useful information about the environment -- as noted above. This is due in part to the sorts of surfaces objects have, in part to a plentiful supply of short wave-length radiation and a transparent atmosphere. (N.B. the last two conditions are very variable.)
- (B) The space of physically possible objects and processes is sparsely instantiated in the actual world (unlike science fiction), i.e. there is limited independent variation of features and relations: this makes images redundant. This is illustrated by planarity, continuity, rigidity, etc. (Marr [1979]) and the fact that no animal has the ear of a zebra and the body of a giraffe.
- (C) Confusing coincidences (e.g. accidental alignments and juxtapositions) are rare. This depends both on the kind of environment and on the low probability of such viewpoints for any given scene.

To make use of (A) a visual system needs good detectors for features of the optic array. Since these depend on laws of physics they don't vary much from one part of the world to another and can be usefully compiled into hardware. If we have evolved mechanisms to take advantage of (A), might we not also have evolved mechanisms to take advantage of (B) and (C)? Using (B) requires using knowledge of what sorts of objects actually occur, e.g. knowing about cylinders, about rigidity, and about zebras and their ears. Some of this (e.g. many objects are locally rigid) is useful in nearly all environments, and might be built into genetically determined mechanisms, whilst some (e.g. what sorts of plants, animals, or buildings, are common) will vary considerably and must be left to individual learning. Making use of (C) involves having good process organisation, to find the 'best' percepts [Hinton 1977].

A consequence of (B) and (C) is that usually any good interpretation of a visual image will be unique, and therefore the best one. (B) and (C) could also justify higher level processes jumping to knowledge-guided conclusions on the basis of partial results from lower levels. This could enable good decisions to be made in poor viewing conditions, and in good conditions would enable decisions to be made faster. (All of this is demonstrated in a very simple world, by the Popeye program [\*2].) So, assumption (A) is of use in good viewing conditions where objects are unfamiliar, whilst (B) and (C) are of use where

conditions are bad but objects are familiar. A system designed with this flexibility might acquire a speed advantage where all of (A) (B) and (C) are satisfied, if different sub-systems work in parallel. It would still have to be basically data-driven (bottom-up) if serious mistakes are to be avoided, but it need not be pass-oriented, with each layer waiting for lower levels to "complete" their analysis (if completion has any meaning).

If higher level systems can operate thus on impoverished data available in adverse conditions, and on incomplete, partial, results of lower levels in good conditions, they should also be able to interpret some highly impoverished artificial data, such as we find in pictures. If so, the interpretation of pictures is not merely a culturally specific, learned, process. If ordinary perception of objects and relationships requires learning, then interpretation of pictures of the same objects will not normally require additional learning, on this view: toddlers we have observed respond naturally to cartoon drawings of familiar situations, without anything like the struggle which characterises learning to read. [Cf. Hochberg and Brooks 1962.] This is not the theory criticised by Gombrich [1960] and Goodman [1969] that realistic paintings and drawings produce the same visual stimulus as the things depicted.

## 2. Unarticulated, semi-articulated, and articulated representations.

We have claimed that vision requires far more than efficient detection of features of the optic array, and that several different domains of structure are processed. To explain why, we must ask: what is vision needed for? An animal, or robot, uses perception to make decisions in pursuit of its goals and to tell whether they have been achieved. It also needs to detect unexpected dangers and opportunities. All this requires construction of representations which articulate the environment into objects with properties and relationships of varying sizes and degrees of abstractness. Rarely will the detection of a particular feature in a particular location on the retina be very significant. Similarly, huge data-bases of unarticulated information, like depth-maps, surface colour or texture maps, surface orientation maps, primal sketches [Marr 1976], can be of little use without considerable further processing. They are effectively new, enhanced, images, even though they may contain 3-D information. Though important for further processing, these unarticulated databases are not directly useful for decision and action: only generalisations related to global image statistics can be learned or invoked e.g. 'lots of green', but not 'plum on tree', might be recognised.

To some extent groupings of fragments of information into larger wholes can be achieved by parallel "local" computations, e.g. relaxation techniques linking items subject to constraints [Hinton 1977, Radig 1978, Frisby and Mayhew this conference]. If the links exist without explicit description of the properties and relations of the linked groups, the database is semi-articulated. The process of growing such links may enable some useful global statistics to be collected, but represents objects only implicitly. Though providing a useful intermediate stage, a semi-articulated database does not explicitly represent one object as above, inside, between, or able to fit into, others. Such information is then not available for deciding, planning and learning. (Compare Marr's 'principle of explicit naming' [Marr 1976]. The same point was made in Minsky [1961].)

Further study of visual articulated representations requires analysis of types of actions performed by different animals. (Some birds can learn to use a foot to depress one end of a lever, exposing food behind the other end. This probably involves articulating the lever into parts, e.g. ends, capable of different though causally linked motions.) It seems unlikely that a small number of mathematically simple structures (e.g. generalised cylinders) with a small

number of mathematically simple relationships (e.g. equations linking co-ordinates) will suffice for human perception. Besides crumpled newspapers (despaired of by Marr [1979]) we see fields and forests. Similarly, cluttered scenes made even of "clean" cylinders will have messy structure at larger scales, like large sets of axioms in a theorem-prover's database. To discern significant objects and relations in large and messy collections of image and scene features we need a much richer descriptive vocabulary than AI vision programs have hitherto incorporated. This is why multiple domains are important.

### 3. Multiple domains

Clowes [1970, 1971] and Stanton [1970] stressed that visual perception and picture interpretation do not simply involve description of image structures. They described "mapping rules" linking different non-isomorphic domains. A domain is a class of structures defined by a "grammar" or set of axioms (e.g. 3-D Euclidean geometry). Scenes have quite different "grammars" from images. This needs to be generalised (as in Hearsay and Popeye) to allow many domains, with different though possibly overlapping grammars. Very briefly, this is because using many different domains allows: (a) 'structure sharing' between processes of recognising different sorts of objects, (b) intermediate results of processing to be relatively secure even if back-tracking is required at higher levels, (c) higher levels to recognise important scene features before lower level processing is complete (see below) (d) high level recognition despite poor low-level detail, (e) data derived from an image to be usefully structured (compare 'Scripts' and 'Frames'), (f) goal-directed activation or de-activation of large chunks of knowledge (e.g. 'mental set'), and (g) communication between different sensory modalities.

A.I. vision work has so far focussed on a small number of mathematically tractable special cases. A good survey of the different domains of structures useful in visual perception is still lacking. Likely relevant domains include 2-D arrays of changing colour and intensity, 2-D configurations of lines and regions and of texture, domains involving patterns of motion in both 2-D and 3-D, overlapping 2-D silhouette shapes [Paul 1976], curved and flat 3-D surfaces, both 2-D and 3-D stick figures [Palmer, 1975], various domains involving forces and a variety of cause-effect relations, intentional actions etc., properties like flexibility, rigidity, elasticity, hardness, etc. Besides plane surfaces, edges, vertices and generalised cylinders for representing shapes, we probably need generalised spheres, hemispheres, bags, tubes, strings, etc. In addition we need models for significant parts of such objects and their surfaces, like: hollows, grooves, holes, lumps, ridges, openings, rims, etc., and models for relating one to another (the groove runs across the hollow). Features and relations invariant under non-rigid transformations are particularly important in our world. We also need a large collection of schemas for types of motion and action: moving towards, moving away from, moving into, flattening, twisting, folding. Compare Hayes on 'naive physics' [1979]. Studies of pictures and cartoon movies can yield useful insights into the structures deployed in perception [Draper 1980]. Of course, it is hard to specify how such models may be represented, invoked, etc. in a working, system. Is all this "cognition" relevant to vision? A major feature of visual learning is linking new domains into the visual system - e.g. learning to see the muscular structure of human bodies, for artistic or medical purposes, learning to see when it is safe to cross the road. There is no sharp boundary between practically useful vision and cognition.

#### 4. The domain of images

The structure of the 2-D image domain is important for both picture interpretation and normal vision: why? Goodman [1969 p.38], rejecting the idea that pictures and objects produce similar visual input, accounts for the "realism" of some pictures in terms of familiarity. But this fails to explain why even a two year old child can learn some pictorial styles, whilst others, though mathematically equally adequate, seem much harder. The human visual system does not work with arbitrary combinations of image elements, but, as the Gestalt psychologists noted, is largely constrained to use continuity, proximity, smoothness, concurrency, symmetry, containment, and other geometric and topological relationships, for linking low-level features into cues which invoke more abstract or global representations, which may themselves be similarly treated. A grasp of such relationships is required for interpreting pictures also. However, much richer image description languages are required than existing AI programs can handle: many can only describe the topology, and a few metrical properties, of networks of straight lines or picture regions. Others provide a simple semi-articulated description with no grasp of the implied structure [e.g. Radig 1978].

Further, articulated 3-D interpretations, required for planning actions, can be linked to image structures to facilitate processing. For instance, to answer the question "What is Y going to hit?", "Will I pass near A if I go straight towards B?" one can "traverse" the relevant part of the image to find the relevant bit of the 3-D interpretation. Moreover, our theory implies that in visual perception and in picture interpretation, descriptions of parts of a complex 3-D scene are built up in parallel. The linking of incomplete descriptions of different parts of the scene to form larger structures, will be facilitated if the 3-D structures are closely related to the network of descriptions of 2-D image structure - the latter providing indexing or addressing routes. [\*4]. This applies to both real vision and interpretation of pictures. (More on this below.)

So, against Goodman we claim that "familiarity" of pictorial representations is not a matter of frequency, but depends in part on the way 2-D relationships are used in normal vision. Of course, mere similarity of domains does not suffice to explain facility with pictures. Maps also make use of 2-D structures and relationships, yet learning to use a map to find one's way around is harder than interpreting pictures. This is partly because our stored knowledge of objects is addressable by means of the kinds of articulated representations produced by both retinal images and artificial pictures, whereas our 'cognitive maps' of familiar surroundings are not normally addressable by the kinds of structures created when we look at maps. Things might be different if we could fly!

#### 5. Reasons for using impoverished articulated representations

There are additional reasons why impoverished picture structures might be related to normal vision. We have already given a general reason why a visual system needs to be able to cope with impoverished representations: articulation of the scene implies reduction of information. Other reasons concern processing, the purposes of vision and the environment:

5.1. Some details may interfere. Much of the detail available to the eye arises from variable conditions, including lighting, atmosphere, viewpoint, non-rigid motion, and changing relations. The use of abstract schemas implies less memory space, faster matching, smaller searches among stored specifications and enables recognition of individuals or types (abstracting from individual details) in novel circumstances. It also provides the basis for forming generalisations.

5.2. Some details aren't needed in a "cognitively friendly" world. It may be possible to distinguish objects on the basis of only a few features. E.g. a colleague once remarked that he could recognize a zebra with just its ear visible. In a CFW where the space of possible structures is known to be sparsely instantiated ((B) above), inferences can be made from fragmentary evidence.

5.3. Details may be missing or spurious. As already noted, poor visibility, natural or artificial camouflage, eye defects, rapid motion, or the presence of visual obstacles, can produce degraded images. Injury can remove stereopsis. Optical flow is not always available. Stereopsis and optical flow don't help with distant stationary scenes. Extracting global features (e.g. silhouette descriptions) from such degraded data sometimes enables recognition of useful cues to overcome the difficulties. Once again, this depends on friendliness: e.g. important objects having distinctive outlines from most views. This requires assumptions (A) and (B).

5.4. Shared structure in memory entries. The system may share recognition processes between different objects by using a discrimination net. As partial specifications are built up, the set of remaining possibilities narrows. [Birch 1978 describes such an extension to Popeye.] Different recognition processes thus share significant sub-processes, minimising back-tracking or breadth-first searching. This uses incomplete descriptions, i.e. intermediate nodes in the discrimination net.

5.5. The need for speed. Even in a CFW, unfriendly circumstances may demand rapid decisions. The next section discusses the relevance of incomplete data.

## 6. Speed and the processing of incomplete representations

Complex articulated representations cannot be created instantaneously. Fast parallel processing at low levels depends on each processor being concerned with a relatively small well-defined portion of the data, and being able to work independently or co-operate with a relatively small set of neighbours. Thus, even data-flow channels can be 'hard-wired'. (Such mechanisms permit certain non-local interactions, via information propagated through the net.) But locality and independence do not characterise the process of articulating a mass of data into objects whose contributory regions change from one image to another. Portions of images relevant to a triangle or tiger vary in size and shape, and may be split into separate regions by intervening objects. Hence data-flow cannot be pre-determined, and organising data from particular images will therefore take a significant amount of time, compared with localised parallel computations. Though detectors for all possible edges may be 'hard wired' in advance, detectors for all possible triangle or tiger shapes could not be similarly pre-determined, partly because of the explosion of connections, partly because not all environments include them. The task of segmenting, aggregating, recognising, and building useful scene descriptions is therefore inherently much slower than low-level tasks. Thus there are limits to the speed-up available from hard-wired parallelism, and other mechanisms to speed things up could be useful: milliseconds may matter when life, or food, is at stake.

Cues invoking previously computed information can speed things up. This old idea [e.g. Roberts 1965] is now associated with the 'frames' theory [Minsky 1975]. Compare the idea of a 'phrasal lexicon' [Becker 1975]. But the theory leaves many questions unanswered: on encountering a new scene where should one start looking for cues in the image? At which level of analysis (in which domain) will the most useful cues be found? How can cues be recognised rapidly? The last question is very difficult, and will not be answered here. Our answer

to the first two is that as far as possible analysis should proceed simultaneously in many locations and at many levels, since the location or domain of the most useful cues cannot be predicted. This should be concurrent with general purpose image processing. Analysis of higher-level domains cannot begin until after some flow of data from lower-levels, but it need not wait for completion. The structure of such a network of processes will vary from image to image, so time and resources may be saved if its growth can be constrained, eliminating or suppressing portions which are not required, and giving priority to those yielding useful results — e.g. activating and deactivating whole domains. This can be achieved if high-level structures (where the networks are relatively small) can be recognised whilst lower level networks are still incomplete. Thus construction of the network of communicating sub-processes which interpret the image, may itself be controlled by partial interpretations.

If, at any level, there is a lot of partially processed information, things may be speeded up by treating the partial results as a new image, in which gross features provide useful higher-level cues: using redundancy in a CFW [Slovan 1978, ch 9]. A specific purpose (e.g. finding a tool) might be achieved using this gross structure, without waiting for details [\*5]. So, in some CFW environments, allowing many domains of structure to be analysed in parallel, could speed up actions. Even marginal advantages may influence biological evolution when resources are scarce, or predators plentiful. There is a kind of recursion in our argument, and possibly also in biological evolution. Where speed is important, the pressure towards further decomposition into parallel sub-systems is great, provided images have sufficient redundancy, i.e. provided it is a CFW.

We have not claimed that higher level processes can influence lower levels, except perhaps by aborting, or re-directing them. But it may be useful for partial results to affect some thresholds or even the invocation of specific forms of analysis, at low levels. Alternatively, cognitive processes may simply control the direction of attention, without modifying the nature of the processing. Even if animal physiology permits no direct downward influence on the processes which generate, say, a primal sketch, there might still be good reasons for designing artefacts differently. It would be no different in principle from making high levels influence direction of gaze, dilation of pupils, convergence of two eyes, etc. all of which affect the low-level image.

### 7. Some implications

In a CFW, multi-layered processing can improve flexibility, graceful degradation and speed. This applies to any kind of activity requiring intelligent analysis and interpretation of a large amount of data, based on expertise in the field, e.g. solving a complex mathematical problem, debugging a program, etc. One consequence is that demands on sub-systems are relaxed. For instance, if processing of level P has to be completed before processing at level Q can be begun, then it is important that P terminate. However, if Q can get started early, then it does not matter if P refines its analysis indefinitely! In vision, input is continuous, so lower levels cannot "finish" their analysis. Thus higher levels must in any case operate in parallel with them.

Moreover, in a CFW, mistakes at lower levels can be tolerated without disaster. The system must be conservative about transmitting items to higher-level domains, i.e. only sending well-supported reports. Then occasional mistaken reports will not combine usefully with other reports received at that level: (compare the role of 'impossible fragments' in Birch 1978). If a relatively large object is recognised on the basis of several different fragments reaching a high level, then the chances of it being a mistake will be small, assuming limited independent variation of object features. So the system need not

guarantee finding the best interpretation of any image, as in Woods [1977], since any good one will normally be unique, as we noted above. So it will often pay to accept a high level decision, abandon lower level analysis, and re-direct attention to the next task [\*5].

All this depends on knowledge enabling fragmentary evidence to invoke specific larger structures, i.e. the principle of limited independent variation. General-purpose knowledge about 3-D structures and the principles by which they map into 2-D images does not constrain the space of possible scene structures so as to permit the inference that any good interpretation of an image is probably the best one. E.g. it does not rule out the existence of animals combining features in bizarre ways. Without specific knowledge of the world, detection of a zebra's ear would not rule out an animal with a trunk, six legs and two tails. The world would then not be a CFW. (This is like employing frequently useful theorems as well as axioms, to control search for proofs in a theorem-prover.) Our arguments are not relevant to the design of a machine whose visual system will never need to act quickly, which will always have perfect viewing conditions and which will often be transferred to a totally new environment where only the most general and primitive knowledge of 3-D structure, lighting, etc. will be of use to it.

Of course, our parallel, schema driven, system will sometimes make mistakes: but people make mistakes and sometimes learn from them. How? Decomposition into sub-systems processing different classes of structures provides opportunities for learning about new rules for linking the different domains, and for inhibiting the invocation of schemas, as well as defining new types of structures in terms of previously known substructures.

#### 8. Problems of incompleteness

This theory raises many unanswered questions. Frank O'Gorman has pointed out in an unpublished manuscript that in a pass-oriented system, where each level of analysis is completed before the next begins, incompleteness of information at a certain location and level has a definite meaning: i.e. it represents the absence of something in the image. We have found it important to distinguish two sorts of incompleteness. It is not too difficult to cope with a gap in a known structure, for instance a hypothesised letter "E", for which the lower "ell" junction has not yet emerged from lower levels. We call this explicit incompleteness: a filler is missing for a slot in a frame. Here there are only two candidate letters "E" or "F", and the word-recogniser can decide which is correct on the basis of other letters which have emerged - even if they too are ambiguous. This depends on limited independent variation of letters in the domain of possible words. Implicit incompleteness occurs when trying to link features together to form cues to drive recognition -- for instance two previously unattached strokes to form a stroke-junction. Whether such features should be linked often depends on which other features are present nearby. From the absence of neighbours it cannot be decided whether this is because there is no evidence at lower levels, or because processing in that region has not yet finished.

In early versions, every level of Popeye[\*2] simply used whatever information had already emerged, and then relied on context, or later bottom-up processing, to correct mistakes. Errors were reduced by delaying processing of any one level until a certain amount of information had been received at that level, using thresholds determined by image statistics. But even this left the garbage-collection problem of undoing mistakes and their consequences. So higher levels confronted with this incompleteness were allowed to ensure that everything up to that level, within a restricted region of the image, had been processed, making use of image-related addressing routes. This caused the focus of attention to

jump about, centering on important image features such as junctions between "bars". A better, more psychologically realistic solution, might be to let each level constantly recompute its hypotheses on the basis of the most recent information from other processes. This would be a generalisation of mechanisms using local co-operative processes, like relaxation. It could be very expensive on current computers, and hard to control.

#### 9. Testing the theory experimentally

The fact that very young children learn to interpret cartoons and other 'impoverished' pictures so easily seems to support this theory. More detailed studies of what they find easy might be helpful. There is some additional evidence for our claim that higher level processing begins before lower level analysis is complete. People often think they've recognised a person or object, then spontaneously realise that a mistake has been made, even after the object has passed from view. Informal experiments with messy pictures of overlapping capital letters forming a word suggest that people often see the word before seeing all the letters. More detailed studies could provide clues as to domains and analyses being processed in parallel, in ordinary vision. Studies of brain damage might indicate which domains of structure (section 3) can be selectively disabled. Useful evidence should come from a study of visual errors. Our theory predicts that even in good visibility, humans and other animals moving rapidly will make more mistakes in an environment containing unfamiliar sorts of objects. (Testing this could be difficult, expensive and dangerous!) Experiments could test whether increasing familiarity improves performance (of survivors!). Different mixtures of familiar and unfamiliar features could be used, to find out if more obvious familiar features lead to errors concerning the other features. Additional experiments would vary lighting, foggy atmosphere, etc. as well. In poor viewing conditions, our theory would predict that visual judgements (especially at speed) would be more accurate when the environment contains familiar objects. Comparative studies might show that only some animals with visual systems possess the ability to process a variety of different domains in parallel.

#### FOOTNOTES

##### [\*1] Acknowledgements:

This work is supported by the U.K. Science Research Council. We have benefitted from discussions with: Geoffrey Hinton, Frank O'Gorman, Steve Draper, Margaret Boden, Max Clowes, Monica Croucher, Steve Hardy, Christopher Longuet-Higgins, David Hogg, Larry Paul, Phil Pettitt, John Rickwood, Robin Stanton and Sylvia Weir, among others. Mike Brady and an anonymous referee made useful comments on a previous version. Judith Dennison helped with production.

[\*2] Preliminary reports on POPEYE can be found in Sloman and Hardy [1976], Sloman et. al. [1978], Birch [1978], and chapter 9 of Sloman [1978]. See also Owen [1980]. Popeye analyses artificially generated dot pictures representing words made of overlapping cut-out capital letters. It can recognise words whilst much of the lower level processing is incomplete. Details will be reported elsewhere. The 1978 conference paper discusses differences between Popeye and the Hearsay system [Erman and Lesser, Hayes-Roth and Lesser], which have much in common. In particular, both process different domains of structure in parallel, though Popeye eschews the 'blackboard' concept. A similar philosophy has been used in the 'Visions' system (IJCAI-5, pp 642-647).

[\*3] Marr makes a similar but different claim in justifying his theory of the 'primal sketch', [e.g. Marr 1979]. He postulates a progression, from image to primal sketch to 2.5D sketch to 3D model, whereas we propose many more domains, processed in parallel. In Popeye, the domains mainly form a hierarchy, but there are two main routes from image data to letter hypotheses and both feed the



word recogniser. We suspect that real visual systems require a far more elaborate network of routes through domains.

[\*4] In Popeye the need for this arises often, e.g. when two parts of a letter are separated because of occlusion. The two parts can sometimes only be related by using a combination of (a) geometrical relationships and (b) partial recognition of the letter, since there are no image cues for linking, like 'back-to-back' tee junctions. So having recognised what may be, say, an E or an F, the program works out roughly where in the image evidence of a missing bottom stroke might be found, and this constrains searching.

[\*5] In Popeye, processing can be aborted when the highest level decides it has recognised the depicted word; lower level analysis will often be incomplete.

TRUNCATED BIBLIOGRAPHY  
(Compressed owing to page limit)

- Becker, J.D. 'The Phrasal Lexicon' T.I.N.L.P. Eds. R.C. Schank and B.L. Nash-Webber. June 1975.
- Birch F. in Sleeman 1978.
- Clocksins, W.F., in Sleeman [ed] 1978.
- Clocksins, W.F. 'A.I. theories of vision', AISB Quarterly 1978
- Clowes, M.B. 'Picture syntax' in Kaneff, 1970.
- Clowes, M.B. 'On seeing things', in A.I. Journal vol 2, no. 1 1971.
- Draper S.W. 'A reply to Clocksin', AISB Quarterly, 1979.
- Draper, S.W. 'Psychological relevance..', AISB Quarterly, 1980
- Erman L.D. and V.R. Lesser in IJCAI-4, M.I.T 1975.
- Gombrich E.H. Art and Illusion Phaidon Press, 1962.
- Goodman N. Languages of Art, Oxford University Press, 1969
- Hayes, P.J., 'The naïve physics manifesto', in D.Michie (ed), Expert Systems in the Microelectronic Age, Edinburgh Univ. Press, 1979.
- Hinton G.E. Relaxation and its role in Vision, Ph.D. thesis, 1977.
- Hochberg, J and V Brooks, 'Pictorial recognition as an unlearned ability' Am Jour Psych, 1962.
- Hochberg, J Perception (2nd Ed) Prentice Hall, 1978.
- Horn, B. Overview lecture on vision, in Sleeman [ed] 1978.
- Kaneff S. Picture Language Machines Academic Press, 1970.
- Marr, D. 'Early processing of visual information', in Royal Society 1976.
- Marr, D, Proceedings IJCAI 1979.
- Minsky, M.L. 'Steps towards artificial intelligence' 1961
- Minsky, M.L. 'A framework for representing knowledge', in Winston [1975]
- Norman D.A. and D.E. Rumelhart Explorations in Cognition, W.H. Freeman 1975
- Owen, D.B. 'Intermediate representations in POPEYE', this volume 1980.
- Palmer S.E., in Norman and Rumelhart 1975.
- Paul, J.L. 'Seeing puppets quickly' Proc AISB Conference, 1976.
- Radig, B., in Sleeman [ed] 1978.
- Roberts, L.G. in Tippet et al, Electro-optical Information Processing, 1965
- Shirai, Y., in Winston [1975]
- Sleeman D (ed) Proc. AISB/GI Conference, Hamburg 1978
- Slooman, A. and S. Hardy, Proc AISB Conference, 1976.
- Slooman A, The Computer Revolution in Philosophy, Harvester Press, 1978.
- Slooman A, and D. Owen, G. Hinton, F. Birch, in Sleeman 1978.
- Stanton, R.B. 'Plane regions...', in Kaneff.
- Winston, P.H. (ed), The Psychology of Computer Vision, McGraw-Hill 1975.
- Woods, W.H. in Proc. IJCAI-5, M.I.T. 1977.

## THE ROLE OF WORLD KNOWLEDGE IN PLANNING

N.S. Sridharan, C.F. Schmidt, and J.L. Goodson  
 Departments of Computer Science and Psychology, Rutgers University  
 New Brunswick, N.J. USA 08903

**Abstract:** Common-sense planning demands a rich variety of world knowledge. We have examined here the view that world knowledge can be structured to form the interface between a hierarchy of action types and a hierarchy of types of objects. World knowledge forming this interface includes not only the traditional statements about preconditions and outcomes of actions, but also the normal states of objects participating in the actions and normative actions associated with the objects. Common-sense plans are decomposed into goal-directed, preparation, and the normative components. This has heuristic value and may serve to simplify the planning algorithm. The algorithm invokes world knowledge for goal customization, action specification, computation of preconditions and outcomes, object selection, and for setting up subgoals.

In a recent survey of research on tactics for problem solving Sacerdoti (Sacerdoti, 1979) concludes that "... the best strategy for advancing the state of the art might well be to focus on integrated systems for plan generation, execution and repair." While agreeing with this conclusion we also note that attempts have been made to design integrated planning and execution systems as far back as 1973 (Nilsson, 1973). However, aside from the work of the SRI-AI group, little research in this area has gained prominence.

Implicit in the quotation above is the suggestion that the agent executing the plan and the agent generating/monitoring the plan are one and the same - as it would be in an integrated robot system. In tackling the problem of one agent monitoring the plan execution of another agent, without explicit and prior communication of the plan, we retain the challenge of dynamic plan repair and goal resetting. In addition, we invoke a rich set of tasks concerned with revising assumptions about the knowledge structures shared by the two agents.

Our research on the task of plan recognition, how an observer interprets another's actions, was started in 1973. In the initial period emphasis was placed on the structure of motives for everyday actions (Schmidt & D'Addamio, 1973) and the process of inferring a goal from a knowledge of what motivates people to act. After a brief attempt at explicating the structure of social (interpersonal) action (Bruce & Schmidt, 1974), recent work has concentrated on an observer process that hypothesizes and revises plan structures for single actor physical action sequences (Schmidt, Sridharan & Goodson, 1978).

Previously at this conference series, we have reported on our representation of hierarchical, non-linear plans (Schmidt, Sridharan & Goodson, 1976) that a) provides the generality needed to monitor and revise plans and goals; and b) logically couples the representation to the attributed beliefs, knowledge and intentions of the actor.

Most recently (Sridharan & Smith, 1978), we reported on the design of a plan hypothesisizer that generates plans emphasizing the teleological order among the components of a plan. The temporal ordering was represented by a set of ordering constraints computed by critics similar to those of Sacerdoti (1975). However, (a) we permitted more than binary constraints (e.g. x not-between y and z), and (b) we employed as critics, logical definitions of the constraint predicates.

In this paper we share with the readers our recent understanding of the role of world knowledge in planning in a common-sense domain. The problems we discuss are not manifest in tidy domains and thus have not arisen in the earlier research on planning. However, we feel these problems cannot be avoided if we retain ambitions of extending goal-directed planning techniques to real problems (Raphael, 1976, p153). We invite the reader to consider a few of the complications arising from a common-sense domain and to share our ideas for dealing with them. The central role of common-sense reasoning in human thinking makes it an important form of reasoning to study and describe.

## 1.0 WHAT IS WORLD KNOWLEDGE?

The problem environment for a problem solver such as STRIPS or NOAH consists of a description of an initial situation and a set of operators for transforming one situation into another. A task for the problem solver is specified by a goal statement. The problem solver attempts to produce a plan of action that would transform the initial situation into one in which the goal statement is true. The problem solver uses world knowledge broadly classifiable into action knowledge (operator descriptions) and object knowledge (a conjunction of statements in first order predicate calculus). Action knowledge usually associates with each named operator its parameters, preconditions, outcomes and a goal designation. The planner's knowledge of the specific situation at hand contains identification of objects relevant to the situation, the type of objects they are and their attributes, and relationships they bear to other objects in the situation. This specific and changing knowledge we refer to as the world model (WM). Sometimes the WM may contain general properties of predicates or general physical principles true in all situations and these may be used to deduce other facts in the WM. For example, a common statement used in the BLOCKS world is

$$(\text{Forall } x \ y \ z) [\text{On}(x,y) \ \& \ \text{At}(y,z) \Rightarrow \text{At}(x,z)].$$

In the context of common-sense planning, knowledge used about actions and objects takes on a much richer structure. The objects mentioned in the WM may be classified variously according to observable or structural characteristics shape, size, color, material, location, contents, components and so on. The object types are often hierarchically setup

Our plan generator (Venkataraman, 1979) is implemented in the language AIMDS (Sridharan, 1978).

providing a variety of levels of descriptions and inheritance rules across these levels. Actions and their case structures are also typed and hierarchically arranged into levels. The action hierarchy gets interfaced to the object hierarchy by means of the attributes of the objects mentioned in the preconditions and outcomes of the descriptions that are associated at any level. The body of general world knowledge (WK) so relating objects and actions is rich and substantial. It is advantageous to view it apart from and as augmenting the world model.

A plan imposes a functional point-of-view over objects and places, rendering the name of an object or place to be of little importance. What is important is the role that the object or place plays within the plan. For example, an object is needed to cut another object, or an object is needed to contain a liquid object, a place is needed on which to cut an object, and so on. In this way, a plan provides expectations concerning the properties that the objects entering into the plan will satisfy. However, we rarely infer the functional characteristics of objects from their observable characteristics. In fact, many of these functional characteristics are not readily inferable by visual inspection. Rather, we are taught that some objects are edible and others are not, some objects make good containers for some liquids and others are ill-suited as containers. This is, of course, what is commonly referred to as "world knowledge."

Aside from such generic knowledge about objects and places, our world knowledge also includes more specific knowledge about the normal or typical aspects of objects or places, about the norms that specify how objects or places should be, and values that specify desirable aspects of objects and places. We discuss in the subsequent sections the representation of world knowledge in a form readily useable for common-sense planning and discuss the ways in which the planning algorithm invokes this knowledge. The action sequence presented below provides for this paper the context from which the main examples are drawn.

Fred is in the living room.  
 The lights go out.  
 Fred reaches into his pocket.  
 Fred takes out a matchbook.  
 Fred lights a match.  
 Fred lights a candle.  
 Fred takes the candle.  
 Fred goes to the basement.  
 Fred approaches the fusebox.  
 Fred opens the fusebox.  
 Fred flips the circuit breaker.  
 The lights come on.  
 Fred closes the fusebox.  
 Fred goes to the living room.  
 Fred puffs out the candle.

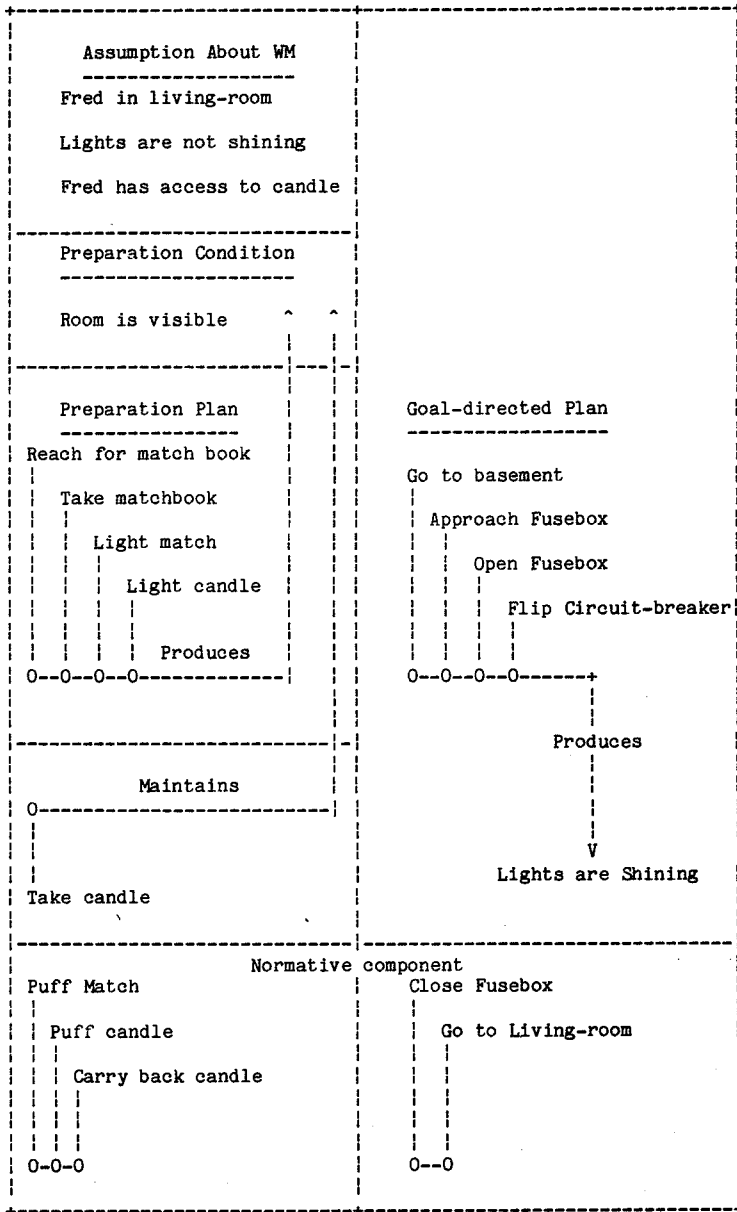


Figure 1. Components of a common-sense plan.

## 2.0 COMPONENTS OF COMMON-SENSE PLANS

A plan consists of the primary goal-directed component, the preparatory component, and the normative component (Figure 1). The primary component is the part which is well-understood in AI and contains a goal and a set of subgoals forming a partial order. The actions (plan units) mentioned in the plan may differ in the level of detail they convey. The final action will have as goal the goal of the entire plan, e.g. lights are shining (see figure). Each subgoal for which no action is planned indicates that the actor believes those subgoals will obtain at the time of execution, e.g. the circuit-breaker switch will be off.

In the example, the goal-directed actions require visibility of the local area as a precondition and the preparation plan achieves this by lighting a candle and then adding other actions to maintain this visibility, i.e. by carrying the candle to the basement. Note that the preparation condition is not a subgoal for any plan unit in the primary plan but is a condition that needs to be achieved and maintained throughout.

The normative component includes actions which do not contribute to the main goal but are included out of considerations such as safety, economy, politeness, or the role assumptions of the planner, e.g. customer, manager, security guard, etc. A set of norms and conventions including setting and role information is required for common-sense understanding of human actions.

We have rejected the design option of extending each action with preparation conditions and normative rules in favor of viewing the whole plan as being dissected into components. This choice is motivated by our desire to reduce the structural complexity of the overall plan and to control the complexity of the reasoning processes involved in planning. Since the same preparation condition, visibility, is needed for every action in the goal-directed component, it is more economical to let this condition be planned for just once. The actions in the goal-directed component may falsify the preparation (e.g. going to the basement) and we add actions needed to maintain the preparation condition (e.g. take the candle). Further, in a planning situation where the search that constructs the goal-directed component requires backtracking, our design strategy pays off. The preparation actions are not involved during this search and simplify the search. This strategy is a heuristic one and may force a penalty should it turn out subsequently that the preparation condition could not be maintained throughout a fully developed goal-directed component. The known fact that lights went off drives the planning mechanism to set up "visibility" as preparation condition for any plan i.e. any goal. It is interesting that this preparation condition is used even when the main goal is to get the lights back on! (See Section 5.1). By using a fact-driven set up for preparation, rather than a goal-driven one, we avoid listing a large number (\*) of needed conditions that would normally be true in the WM, and which would normally be irrelevant to subgoal generation.

Note that the actions in the normative component are temporally constrained only weakly in relation to the actions in the main parts of the plan. The plan monitoring algorithm can compute the needed additional

---

(\*) Some argue that such a list can never be complete.

constraints and these are not discussed here.

### 3.0 REPRESENTATION OF ACTIONS AND THE IMPLICIT HIERARCHY

The meaning of the verb "to light" captures the general meaning of a diverse collection of specialized actions. Lighting a match from a matchbook requires preconditions such as that the agent should hold a matchbook, the matchbook should have an exposed striking surface, and so on; whereas lighting a candle with a match requires preconditions such as that the agent should hold a lighted match, the candle should be accessible and so on. We require the action representation to permit the specific meaning (preconditions, goal, outcomes, norms and other information) to be customized for the various attributes of objects and locations participating in the action.

To give a name for each possible combination of action type specialized by parameter types would lead to serious combinatorial explosion apart from yielding a representation that is somewhat unnatural. This approach has been adopted in controlled tidy domains where it is feasible. This yields acts such as LIGHT-CANDLE, LIGHT-MATCH, GO-WITH-CANDLE-INHAND etc. (cf. PUSH, PUSH-BOX, PUSH-BOX-THRU-DOOR etc. in the STRIPS world). We have chosen to represent actions at a level of generality that corresponds to the English verbs TAKE, OPEN, LIGHT, GO, GET and so on. We create an implicit specialization hierarchy by defining a language in which specialized actions may be described. Such descriptions are used for indexing into a rule base for calculating the preconditions, outcomes and other knowledge associated with any action. The conjunctive semantics given to the descriptions provides a simple mechanism of inheritance down the specialization hierarchy.

The process of action modelling has been extended so that the precondition, goal, and outcome calculations are done and the act is customized using available knowledge. In programming this mechanism, we have carefully separated the mechanism from the collection of statements about objects and actions that are accessed by the mechanism. This permits us to examine how varying available knowledge influences the plan hypothesizer and plan recognition programs.

#### 3.1 Mechanism For Act Customization

The meaning of the act is computed by retrieving a set of rules whose left sides are patterns that test attributes of the instances (objects, locations, and persons) mentioned in the case relations (arguments) of the action. The right sides (after appropriate substitution for variables) yield the set of preconditions. Consider as an example our specification of the preconditions for the action LIGHT. Similar rules are stated to describe the Goal and Outcomes.

Case Relations: LIGHTing of an OBJECT  
                   with an OBJECT  
                   at a LOCATION  
                   by an agent PERSON.

Precondition rules:

1. (LIGHT of (OBJECT M type MATCH  
   in (OBJECT B type MATCHBOOK))  
       by (PERSON P))  
    => (P holds B)
2. (LIGHT of (OBJECT C type CANDLE)  
       with (OBJECT M type MATCH)  
       by (PERSON P))  
    => (P holds (OBJECT M status LIT)  
       has-accesssto C)
3. (LIGHT of (OBJECT O type PIPE)  
       by (PERSON P))  
    => (P holds (OBJECT O (contains (SUBSTANCE type TOBACCO))))

Rules are presented above in a simplified syntax, with lower case words giving the relations. Consider the first precondition for LIGHT. It has M,B and P for variables which will be bound during matching. In matching a given action, M and P will be bound from the case relations of the action. The object bound to M is tested further in the WM yielding a binding for the variable B. If a given action matches the left side, a precondition (P holds B) will be included after substitution for the variables. Note that as a result of using nested descriptions on the left side a precondition is introduced about B which is not an object directly involved in a case relation. The second precondition rule exhibits a nested description on the right side which is treated as a conjunction.

Additional example:

Case Relations: POURing by an agent PERSON  
                   from an OBJECT  
                   to an OBJECT  
                   of a SUBSTANCE.

Precondition Rules:

1. (POUR by (PERSON P)  
       from (OBJECT O))  
    => (P holds O)
2. (POUR by (PERSON P)  
       to (OBJECT C))  
    => (P has-accesssto C)
3. (POUR from (OBJECT O type COVERED-CONTAINER))  
    => (O status OPEN)
4. (POUR to (OBJECT C type (AMONG GLASS CUP)))  
    => (C on (OBJECT T type (AMONG TABLE COUNTER)))
5. (POUR from (OBJECT O) of (SUBSTANCE S))



=> (0 contains S)

#### 4.0 AUGMENTED WORLD MODEL

The world model is always a partial specification of the planner's current beliefs about the state of the world. The planner augments this with his expectation of what the "normal" values of attributes are. A precondition may be neither true nor false in the world model, i.e. it is unknown in the WM. Knowledge of normal states can be used to decide whether to plan for achieving the precondition as a subgoal. For example, in a plan to drink water he may assume there are glasses in the cabinet, he may assume they are clean and thus not plan for these two propositions, yet he must assume that cabinets are normally closed and add a plan unit for opening the cabinet door. Thus, it should be evident that knowledge of normal states determines greatly the contents of a common-sense plan. Whereas these assumptions do contribute to the fallibility of the plan, there are norms we follow to help promote reliability of such assumptions. There are a very rich set of norms including sincerity, honesty, politeness for interpersonal actions; however, in the restricted case of a single person carrying out a sequence of everyday physical actions, many of the normative actions can be explained by one norm: restore objects to their normal state if an action in the plan will alter them. Some of the common injunctions a planner follows are: if you turn it on, turn it off; if you plug it in, unplug it; if you open it, close it; and so on. The role of what is normal in the world is essential to this form of reasoning.

#### 5.0 PLANNING TACTICS AND ALGORITHM

The planning algorithm we use works on the same broad principles as that of Sacerdoti (1975). The given goal is used to select a plan expansion and the expanding plan forms a network of actions. The structural connections of the network indicate, in our case, which action serves to accomplish which subgoal of other actions. The temporal constraints are computed after each round of expansion and are maintained as a list of constraint assertions. The critics we use are primarily the "Resolve conflicts" and the "Redundant subgoal" critics which compute ordering constraints. These are stated using assertions of the form "P not next to Q" and "P not between Q and R". We have found that the "Use existing objects" critic leads to premature and unwarranted optimization of the plan for monitoring purposes and thus we do not use it. Most optimization decisions are left to the algorithm that uses the plan for monitoring observed actions.

The discussion below avoids certain important aspects of the planning algorithm, i.e. the interaction between the plan hypothesizer, the action monitor and the hypothesis revision mechanism. These will be discussed elsewhere; The focus here is on ways in which additional world knowledge is invoked by the planning algorithm and the influence this has on the planning algorithm.

## 5.1 Goal Customization Rules And Action Specification

The goal statements admitted in the representation of common-sense plans have the same generality and looseness of meaning as that associated with actions (discussed above, Section 2.0). Action specification is the familiar technique of selecting the action to be used to achieve a particular goal and is done using rules stated in the form similar to those above for preconditions etc. Some goal propositions require further customization in order to generate an action specification. For example, to achieve the goal "(OBJECT A) near (OBJECT B)" one may move A to where B is, or move B to where A is, or even move A and B to some new location. The choice depends on characteristics of the objects A and B and the locations of A and B. The more easily moveable object will be moved. If there is not enough room to put one object next to the other, both may be moved to a suitable third location. When both objects are easily moveable, convention (norm) may dictate which should be moved. Let us say B is to be moved to LA, the location of A. We make the action specification in two decision steps: (goal customization) (B loc LA) and then (act specification) (MOVE object B dest LA). Goal customization is done through a rule set that makes choices by examining the characteristics of the objects mentioned in the goal proposition and the current state of the world model. It provides a specialized goal proposition that is considered appropriate to the situation at hand. The customized proposition is then used for action selection.

Using goal customization has a notable consequence. In common-sense plans, it is possible to admit circular subgoals, i.e. a goal G is achieved by means of an action sequence S one of whose subgoals is also stated to be G. In traditional planning algorithms, detection of circularity prompts failure of the last selection made. We do not permit failure so readily. For example, to attain the goal "visibility" one may choose to light a candle; yet to light the candle one may have a general statement about the visibility of the room as a subgoal. (Formulating the actions to avoid circularity is possible and would be uninteresting as a solution). We admit the circularity but require that the subgoal, on its second occurrence, be customized to a more narrow proposition. Lighting a match gives enough visibility to satisfy the subgoal set up by lighting the candle, which gives enough visibility to satisfy the subgoal set up for turning the lights back on.

## 5.2 Object Selection Rules

The action selected from the customized goal proposition is often a partially specified action, for example "Light a candle". The choice of which candle to light does not involve world knowledge but is based on the heuristic of picking one that is available or readily accessible. World knowledge is used to select a match or lighter as the candidate object classes for the instrument to be used in "Light a candle". An example rule is given below. Object selection rules cannot mention (by name) any specific instance of an object in the world model, but refer to objects in the world model by descriptions. These descriptions are taken as specifications for finding an object and the heuristic of using an available one then is applied to pick an instance.

```

[[ (LIGHT of (OBJECT C type CANDLE))
  (convention
    (LIGHT with (OBJECT L type LIGHTER status LIT))
    (LIGHT with (OBJECT L type MATCH status LIT)))]
[[ (DRINK of (OBJECT L type LIQUID status HOT))
  (cleanliness
    (DRINK from (OBJECT C type CUP status CLEAN)))]

```

The alternatives available are grouped and tagged with the norm that governs the selection. Presently norms are not taken into account by our program; if available they are used only to offer an explanation for the choice of the object. A forthcoming extension of the program will plan under different settings of the norms such as economy, safety, cleanliness etc. that may govern the formation of one-person physical action plans.

## 6.0 CONCLUSION

Common-sense planning demands a rich variety of world knowledge. We have examined here the view that world knowledge can be structured to form the interface between a hierarchy of action types and a hierarchy of types of objects. World knowledge forming this interface includes not only the traditional statements about preconditions and outcomes of actions, but also the normal states of objects participating in the actions and normative actions associated with the objects. Common-sense plans are decomposed into goal-directed, preparation, and the normative components. This has heuristic value and may serve to simplify the planning algorithm. The algorithm invokes world knowledge for goal customization, action specification, computation of preconditions and outcomes, object selection, and for setting up subgoals.

Our representation of world knowledge is tailored to the planning algorithm yet is separate from it (cf. Fahlman, 1974). Other researchers have described representations suitable for various comprehension (Carbonell, 1979; Charniak, 1978) and recognition (McCarty & Sridharan, 1980) tasks.

We note that currently we permit functional attributes of objects such as "edible", "food container", "fusebox", and so on to be introduced into the WM. Thus the functional attributes required by the plan are tested directly against the WM. We are now endeavoring to restrict the WM to contain only strictly observable or structural attributes and to require the action monitoring and hypothesis revision processes to provide an inferential bridge between the observable and functional attributes.

Acknowledgement: Our work on plan generation has benefited directly from the participation of Don Smith and K.W. Venkataraman. This research was supported by Grant RR-643 to the Rutgers Research Resource on Computers in Biomedicine from the BRP, Division of Research Resources, NIH.

## REFERENCES

1. Bruce, B. & Schmidt, C.F. (1974), Episode understanding and belief guided parsing, Report CBM-TR-32, Department of Computer Science, Rutgers University, July 1974.
2. Carbonell, J.G. (1979), Subjective understanding: Computer models of belief systems. Ph.D. Thesis, Yale University, 1979.
3. Charniak, E. (1973), On the use of framed knowledge in language comprehension. Artificial Intelligence, vol. 11 (3), December 1978.
4. Fainman, J.E. (1974), A planning system for robot construction tasks. Artificial Intelligence, vol. 5 (1), January 1974.
5. McCarty, L.T. & Sridharan, N.S., (1980), The Representation of an Evolving System of Legal Concepts: I. Logical Templates, Proceedings of the Third National Conference of the Canadian Society for Computational Studies of Intelligence, (Victoria, BC), May 1980.
6. Nilsson, Nils J. (1973), A Hierarchical robot planning and execution system. Technical note 76, SRI AI Center, April 1973.
7. Raphael, B. (1976), The Thinking Computer. W.H. Freeman Company, San Francisco.
8. Sacerdoti, E.A. (1975), Non-linear nature of plans, Advance Papers of the 4th International Joint Conference on Artificial Intelligence (Tolisi, USSR), MIT AI Lab Publications, September 1975.
9. Sacerdoti, E.A. (1979), Problem Solving Tactics, Proceedings of the 6th International Joint Conference on Artificial Intelligence (Tokyo), p1077-1083.
10. Schmidt, C.F. & D'Addario, J.A. (1973), A model of the common-sense theory of intention and personal causation, Advance Papers of of the 3rd International Joint Conference on Artificial Intelligence (Stanford, CA), p465-471.
11. Schmidt, C.F., Sridharan, N.S. & Goodson, J.L. (1976), Recognizing plans and summarizing actions. Proceedings of the Summer Conference on Artificial Intelligence and Simulation of Behavior, Edinburgh, July 1976, p291-306.
12. Schmidt, C.F., Sridharan, N.S. & Goodson, J.L. (1978), The plan recognition problem: An intersection of psychology and artificial intelligence. Artificial Intelligence, vol. 11 (1 & 2), p45-83.
13. Sridharan, N.S. and Smith, D. (1978), Design for a plan hypothesizer, Proceedings of the AISB/GI Conference (Hamburg, BRD), p315-324.
14. Sridharan, N.S. (1978), AIMDS User Manual, Version 2, Report CBM-TR-89, Rutgers University, June 1978. Revised report to be issued September 1980.
15. Venkataraman, K.N. (1979), A Planning system used for plan recognition in a common-sense domain. Rutgers University CBM-TM-83, September 1979.

AN ALGORITHMIC ACCOUNT OF ENGLISH MAIN CLAUSE CONSTRUCTIONS

M. J. Steedman	&	A. E. Ades
Department of Psychology		Max Planck Gesellschaft
University of Warwick		Berg en Dalseweg 79
Coventry, ENGLAND CV4 7AL		Nijmegen, NETHERLANDS

"The fundamental aim in the linguistic analysis of a language L is to separate the grammatical sequences which are the sentences of L from the ungrammatical sequences which are not sentences of L and to study the structure of the grammatical sequences. The grammar of L will thus be a device that generates all of the grammatical sequences of L and none of the ungrammatical ones." (Chomsky, 1957)

1. Introduction

Transformational Grammar (TG) proposes that the grammatical constructions of a language are those which can be generated by a base grammar of context-free phrase structure rules, augmented by a set of transformations. Transformations are rules which map the sentence-structures generated by the base onto other structures, and ultimately onto surface grammatical structures of the language in question. For example the following sentences are regarded by transformationalists as being derived via different transformations from a single canonical deep structure generated according to the base grammar.

(1) a. I will marry her.    b. Will I marry her?    c. Her I will marry!

The suggestion has a considerable appeal. The canonical deep structure, which is rather similar to the surface structure of (1a), can be thought of as directly related to the propositional content which all of these sentences share, and which they respectively assert, question and contrastively emphasise the object of. Quite apart from purely formal justifications for such rules, it is tempting to identify the idea of the transformation which produces (1b), for example, with the process of relating such a question to the proposition which must be verified by a hearer if they are to answer the question.

However, as the Transformationalists themselves were the first to point out, such an analysis raises as many questions as it answers. If there are transformations in the grammar of English which produce the sentences of (1), why are there not transformations which generate (for example) the following constructions?

(2) a. \*Will her I marry.    b. \*Will marry her I.    c. \*Marry will her I.

A transformation is a rule which can map any structure into any other. It follows that there is nothing to prevent such a rule being devised to perform any of the twenty-four possible arrangements of these four elements. Yet nobody could seriously propose that the five or so grammatical arrangements are a random selection from among the twenty four. The non-grammaticality of such non-sentences as the above therefore remains unexplained by the basic Transformational theory of grammaticality.

Because of the non-existence of all but a tiny fraction of the possible transformations, and because of other considerations, such as the complexity of the task of acquiring a Transformational Grammar, and certain unexplained constraints upon the application of the transformations that do occur in English, such as Relativisation, Transformationalists have produced a wide variety of generalisations about constraints upon the form and application of transformations. Such generalisations constitute, in part, the data upon which the present study rests. However, until they are shown to emerge as

inevitable consequences of a particular device or mechanism, the job of explaining the facts of grammaticality remains unfinished. In the present paper, we shall largely confine ourselves to the question of parsing English main clauses, although we believe that our account has implications for the analysis of complex sentences, for models of production, and for languages other than English. In particular, we confine ourselves to roughly that class of constructions which Emonds (1976) has characterised as being derived via "root" transformations. We believe, like Bresnan (1976) and other adherents of the Base-generation hypothesis, that many of the remaining constructions, such as passive and "tough movement", should be directly handled in the base rules. However, unlike the Base-generativists, we concur with the standard TG account in viewing sentences such as (1) as being in a sense rearrangements of one another. We shall show that displaced constituents can be restored to their canonical relationships using nothing more powerful than a certain kind of left to right parsing algorithm, operating upon a single push-down store, or stack.

## 2. The Stacking Constraint

It is unanimously agreed that whatever else may characterise it, the human parsing mechanism works from "left" to "right" - that is, from the earliest parts of the sentence to the latest. There are a number of ways of parsing sentences left to right according to grammars of Context Free Power - for example, the recursive transition network parser which is the basis of the ATN. But how could such a parser be augmented in order to allow it to parse the constructions involving displaced constituents, which motivated the introduction of transformations? One simple way would be to provide the parser with a store in which to keep constituents which are displaced. When in its left-to-right traverse of the sentence it encountered a constituent not in the canonical position defined by the CF grammar, the parser could put it into this store. When, later on in the sentence, it failed to find that constituent in its canonical position, it could retrieve it from the store.

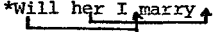
If the main clause constructions of English are examined in the light of such a model, two striking generalisations emerge. Consider for example the five grammatical arrangements of a subject, an auxiliary, a transitive verb and an object.

- |                                       |  |
|---------------------------------------|--|
| (3) a. I will marry her               | d. <u>her</u> I will marry <u>↑</u>          |
| b. will <u>I</u> marry her?           | e. <u>whom</u> will <u>I</u> marry? <u>↑</u> |
| c. (marry her) <u>I</u> will <u>↑</u> |  |

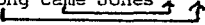
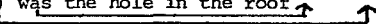
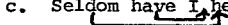
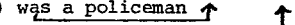
All of these constructions can be restored to canonical form with such a parser. That is to say, they all involve what a Transformationalist would call "leftward movement" of the displaced item, which is exactly the kind of displacement that such a parser can handle. The restoration of the displaced constituents to canonical position is indicated by a convention in which an arrow represents the transfer of the constituent to the temporary store and its re-emergence into canonical position. It is also striking that in the case of sentence (3e), the order in which the displaced object whom and auxiliary will enter the store is the reverse of the order in which they are retrieved into canonical position. The last in is the first out, as can be seen from the fact that the arrows do not cross.

Both of these observations turn out to be quite general. That is to say that all English main clause constructions can be explained in terms of "leftward movement", which is what would be expected if they had to be parsed by such a mechanism as the one under discussion. What is more,

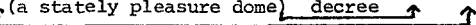
it is always the case that, when more than one constituent is displaced, the last item to enter the temporary store is the first to be retrieved. No rearrangement of the elements of (3) which violates these constraints is grammatical. For example,

- (4) a. \*Will her I marry  c. \*Marry will her I   
 b. \*Will marry I her 

And all of the remaining single clause constructions characterised by Emonds (1976) as root-transformed appear to obey the constraint.

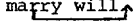
- (5) a. Along came Jones   
 b. (Far more serious) was the hole in the roof   
 c. Seldom have I heard a more revolting suggestion   
 d. (Leaning against the bedpost) was a policeman 

Even the non-standard poetic or archaic constructions of English seem to be subject to the same constraint, as for example:

- (6) (In Xanadu) did Kubla Khan (a stately pleasure dome) decree 

Such a constraint is of course exactly what would be expected if the particular store in question were a push-down store, or stack.

The mechanism that we have just described is a generalisation of the way in which the ATN handles another leftward movement transformation, namely relativisation, and is in keeping with certain suggestions of Woods (1973). However, although parsers of this kind will parse all the main clause constructions of English, it is not the case that they will parse only those constructions. For example, they will allow the following sentence.

- (7) \*I marry will her 

There are good reasons to suppose that no language will ever be found to require the kind of rearrangement that such a construction would involve. It is clear that this type of mechanism is still too unconstrained.

It is, of course, the case that a parser for a recursive context free grammar also requires a stack, so the machine that we have described above has two stacks, one for parsing and another for storing displaced constituents. In the next section we present a left to right parser that only uses one stack for the two purposes of storing displaced constituents and accomplishing context-free parsing, and uses no other store. The additional constraints that explain the ungrammaticality of sentences like (7) emerge as a corollary of this dual function of a single stack.

### 3. The Model

Five rules for the combination of morphemes, words and constituents will be presented in the following pages. The rules are of a predominantly "bottom-up" nature, rather than top-down and predictive as the ATN is, and are a variant of a "shift and reduce" parser using a Categorical Grammar. A Categorical Grammar has the same power as a CFPS grammar, and takes the form of a series of lexical entries specifying the syntactic role of each morpheme in the language. The entries will be given in the following form:

- (8) <morpheme> : X/Y

Such an expression means that the morpheme is the "leftmost edge" of a constituent of type X, and will combine with the constituent Y. For example, transitive verbs bear the category VP/NP. For certain primitive categories, Y may be null: the category of a noun is simply N.

A category X/Y is to be thought of as a function. For example, the category VP/NP identifies a transitive verb as a function which maps NPs onto VPs. Each item, whether it is a primitive item like a verb, or a compound

category like a VP, has such a category associated with it. Each item is also to be thought of as being associated with a meaning representation. The nature of such meaning representations is not our immediate concern; they may be thought of as deep structures, or procedures, or anything else. It will become clear later that we have in mind a semantically interpreted meaning representation, rather than an autonomous syntactic structure.

Since the only store the machine has is the stack, the most basic rule is one which introduces the next word of the sentence to the top of the stack.

(9) Rule 1: The Input Rule

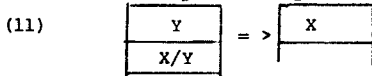
Place the next word on the Top of the Stack.

The second rule is the basic rule for combining stacked items.

(10) Rule 2: The Forward Combination Rule

If the topmost item on the stack is a Y, and the item beneath is an X/Y, ("an X lacking a Y"), then combine the two to yield an X, and return it to the Top of the Stack.

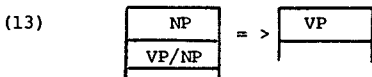
Rule 2 can be represented pictorially as:



Consider for example the following fragment of a categorial lexicon:

(12)        her        : NP  
              marry    : VP/NP

In parsing the phrase marry her, the words marry and her are put onto the stack in order, by successive operations of Rule 1. At this point, the condition for the Forward Combination Rule 2 applies: (the semantic interpretation of) a VP is constructed from (the semantic interpretations of) the words marry and her, and is replaced on top of the stack, as in the following diagram:



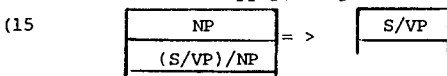
Rule 2 is a general-purpose rule for combining objects of all categories and their associated semantic representations, rather than an expansion of a particular category, like a rewrite rule. It is the words and the constituents themselves that determine the category of the result.

The category of the tensed verb will is slightly more complex. It inherits the category from a combination of the category of the verb stem and that of tense itself, of course, but in the present paper we shall ignore this part of the process. The category that results is the following:

(14)        Will + present : (S/VP)/NP

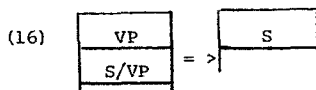
Such a category defines tensed will as a function from (subject) NPs onto a function from VP's onto sentences. (In short, it is something that combines a subject and a verbphrase, in that order, to give a sentence.)

If the sentence Will I marry her is subjected to the rules given above, they can parse it in the following sequence of operations. First the words will and I must be put onto the stack in succession to yield a configuration to which Rule 2 can apply, to yield a sentence lacking a verb phrase.



Next, the words marry and her can be put on the stack by rule 1, and can combine as before to yield a VP, leaving the stack in the following state, in which rule 2 can again be applied to yield a complete S.



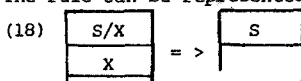


We have deferred all discussion of the algorithm that applies these rules. However there is only one order in which they can apply in this example, and only one result that they can yield.

The subject may precede the tensed verb, as well as succeed it. In fact, there is a widespread tendency for certain combinations to occur both in left-right and right-left order. (Compare 3a and c, for example). It follows that there is a need for a second combination rule, where a function combines with an argument which is beneath it in the stack. The appropriate combination can be effected by the following rule:

- (17) Rule 3: The proposed Constituent Rule  
 If the topmost item on the stack is a S/X, ("an S lacking an X"), and the item beneath it is an X, then combine the two to yield an S, and return it to the Top of the Stack.

The rule can be represented with the usual sort of diagram:



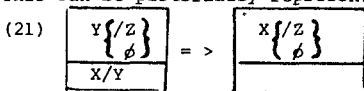
It is restricted to sentences, unlike Rule 2, since nothing else appears in English to be able to take its arguments from beneath - that is, to have them preposed. Noun phrases like frog the are not well formed, and nor are verb-phrases like the frog eat.

Rule 3 can be made to handle all cases of preposing if Rule 2 is generalised in a certain way. The form of Rule 2 given above implies that, for a right-branching structure such as a verb-phrase, no combination can occur until the rightmost lowest word has been entered to the top of the stack, as in the preceding analysis of Will I marry her. However consider the Topicalised object her in

- (19) Her I will marry  
 If such a preposed object, once stacked, were separated from the verb by an arbitrarily long sequence of constituents, it could only be recovered by a device capable of looking arbitrarily far down the stack, and extracting the object NP. Hence, since the stack is the only storage space available to the mechanism, there is no alternative to making I will marry, I might have married, I might have been marrying, and so on, be single entities of the type S/NP - that is, functions from (object) NPs onto sentences. Such objects can be constructed with the following generalisation of the Forward Combination Rule:

- (20) Rule 2 : The Forward Combination Rule  
 If the topmost item on the stack is a  $Y\{\frac{Z}{\phi}\}$   
 ("a Y, possibly lacking a Z"), and the item beneath it is an X/Y ("an X lacking a Y"), then combine the two to yield a  $X\{\frac{Z}{\phi}\}$  ("an X, possibly lacking a Z"), and return it to the Top of the Stack.

This can be pictorially represented as:



Such a rule subsumes the earlier version, as a special case where Y is complete - that is, where /Z is null. Like the earlier rule, it is to be thought of as combining the semantic representations associated with the

two items to which it applies into a single semantic representation. In the most general terms, what the rule does is to compose the two functions  $X/Y$  and  $Y/Z$  into a single function  $X/Z$ .

The claim that there is an identifiable stage in the parsing when a semantic representation is built for the incomplete sentence John will have should not be mistaken for a claim that such an object corresponds to a constituent. A parser of this kind could be made to build the familiar kind of syntactic structure, with all the usual constituency relations. The claim would then be simply that there are stages in parsing where incomplete constituents are assembled.

Such a construct may find its object immediately above it on the stack, and may combine with it by the Forward Combination Rule 2. On the other hand, the object may be underneath it, as in (19) in which case it may combine by Rule 3. However, Rule 3 is unlike Rule 2 in that it does not generalise to the case where the preposed item is incomplete. This restriction is necessary to prevent the mechanism accepting such anomalous sentences as (22) \*Marry I will her, where such incomplete categories as VP/NP are preposed.

Two further rules are required to complete the main clause parser. We shall not discuss these rules in any detail here, and refer the reader elsewhere for a complete presentation (Ades and Steedman, 1979). First, a rule is required which will achieve the aforementioned combination of a verb stem with an affix, such as Tense, or the participial affixes -en and -ing. (Such a rule was taken for granted in the earlier analysis of Will, where its category (S/VP)/NP was assumed to derive from the application of such an affix combining rule to Tense and the modal stem). Second, it is notorious in the Transformational literature that there is something special about noun-phrases with respect to so called movement. By and large, you cannot move things out of nounphrases (Bach & Horn, 1976). So, although you can (in the terms of TG) move an object NP out of a verbphrase - as in (23) Her I will (marry)

- you cannot move a noun out of the corresponding position in a nounphrase (24) \*Frog I will eat (the)

Our explanation of these phenomena is that verbs are in fact functions from nounphrase referents onto verbphrases. The article the, on the other hand, is categorised as a function which maps nouns onto nounphrase senses, written

(25) the : NPS/N

Since a sense is a different kind of object from a referent, and NPS is not the same symbol as NP, Rule 2 can only apply once the nounphrase sense is complete, so that the referent can be found and replaced on the stack as an object of category NP. Then and only then can a verb combine with it. The conversion of a complete nounphrase sense into a referent is accomplished by a fifth rule, which assumes an account of reference similar to that pioneered by Winograd (1972). The rule renders impossible any so-called movement out of noun phrases, such as those producing the anomalies which have provoked Transformationalists to introduce a number of constraints on transformations, such as the Complex NP Constraint of Ross (1967). Again we must refer the reader to the fuller account of the model.

The five rules are intrinsically unordered. Since in this paper we are solely concerned with questions of grammaticality and not with questions of resolution of local ambiguity during parsing, we can assume the trivial nondeterministic algorithm "apply any rule you can, until no further rule can apply". (However, a more efficient algorithm does exist, and has been

programmed). We cannot emphasise too strongly that questions of local ambiguity are logically quite separate from questions of grammaticality, and in concentrating on the latter, and excluding the former, the present model is to be contrasted with such accounts as Marcus (1978) and Frazier & Podor (1978) which are predominantly concerned with local ambiguity. It is quite simply neutral as to which among mechanisms such as "backtracking" or "waiting and seeing" are the more appropriate.

#### 4. Basic Sentence Patterns

The simple model outlined in the previous section will, given appropriate and on the whole uncontentious lexical categorisations, accept a wide variety of grammatical English main clause constructions. More important, it will refuse to accept virtually all of the non-grammatical rearrangements of the same elements. As an example, we shall consider exhaustively the arrangements of a simple clause including subject, auxiliary and transitive verb phrase.

In Section 2, some time was devoted to considering why certain arrangements of the words I, will, marry, and her are allowed in English, while others are not. In order to demonstrate the model advanced in Section 3, it will be useful to consider in detail the workings of the algorithm on all of the twenty-four possible orderings of these four words. The categories of I and her are assumed to be NP, that of marry to be VP/NP, and that of tensed will to be (by the Affix Cancelling rule) (S/VP)/NP.

In the following examples, the working of the algorithm will be illustrated using the convention that, where two items are combined under a given rule to yield a given result, then those items will be underlined. The result of the combination will be written beneath the line, which will also be indexed with the number of the rule in question. Of course, the result may itself be combined with something else by a further application of a rule, indicated by the same underlining convention. Thus, the successive underlinings down the page represent successive stages of the parsing. Since the original words are put on top of the stack in strict left-to-right order, the operation of the Input Rule 1 is not indicated in these examples, and nor is the operation of affix Cancelling. Consider first the "canonical" form of the sentence

(27) a. 
$$\begin{array}{ccccccc} \underline{I} & & \underline{will} & & \underline{marry} & & \underline{her} \\ \text{NP} & & (\text{S/VP})/\text{NP} & & & & \\ \hline & & \text{S/VP} & & & & \end{array}$$

The NP I is first placed on the Top of the Stack (TOS) by Rule 1. In this situation the only rule that can apply is again Rule 1, so will is entered on TOS. Now the condition for Rule 3 applies: there is a preposed NP, which is absorbed into the S-node as specified in the rule, leaving S/VP on TOS. The only rule that can now apply is Rule 1, so marry is entered to TOS:

(27) a'. 
$$\begin{array}{ccccccc} \underline{I} & & \underline{will} & & \underline{marry} & & \underline{her} \\ \text{S/VP} & & & & \text{VP/NP} & & \\ \hline & & & & & & \\ \hline & & & & & & \end{array}$$

The conditions for Rule 2 are now met and the partial combination with marry is performed, leaving S/NP on TOS. The only rule that can apply next is Rule 1, which places her on TOS, so that the final combination can be made, under Rule 2.

(27) a''. 
$$\begin{array}{ccccccc} \underline{I} & & \underline{will} & & \underline{marry} & & \underline{her} \\ \text{S/NP} & & & & & & \text{NP} \\ \hline & & & & & & \\ \hline & & & & & & \end{array}$$

The analysis of the question form is very similar.

- (27) b. 
$$\begin{array}{cccc} \text{Will} & \text{I} & \text{marry} & \text{her ?} \\ \hline (S/VP)/NP & NP_2 & VP/NP & NP \\ \hline S/VP & & & \\ \hline S/NP & & & 2 \\ \hline S & & & 2 \end{array}$$

The process is just the same as for (27a) except that will and I are combined by Rule 2 (Forward Combination) rather than by Rule 3 (Preposed Constituents). Indeed, it is clear that the Subject and the Tensed verb may occur in either order, as long as they are adjacent.

- (27) c. 
$$\begin{array}{ccc} \text{Her} & \text{I} & \text{will} & \text{marry} \\ \hline NP & S/VP & VP/NP_2 & \\ \hline S/NP & & & \\ \hline S & & & 3 \end{array}$$
 d. 
$$\begin{array}{ccc} \text{Who (m)} & \text{will} & \text{I}_2 & \text{marry} \\ \hline NP & S/VP & VP/NP_2 & \\ \hline S/NP & & & \\ \hline S & & & 3 \end{array}$$

(As it stands, of course, the model does not explain why (27d) is only possible with a WH-element, as opposed to ?Her will I marry, which is comprehensible, but archaic. Neither does it explain why the subjects of (27e), and indeed of (27c) must be "given" anaphors such as pronouns, rather than "new" NPs. Such questions are in the domain of semantics, rather than syntax). The parser handles the remaining standard construction, Verbphrase preposing, as follows:

- (27) e. 
$$\begin{array}{ccc} \text{Marry} & \text{her} & \text{I} & \text{will} \\ \hline VP/NP & NP_2 & NP & (S/VP)/NP_3 \\ \hline VP & & S/VP & \\ \hline S & & & 3 \end{array}$$

The model allows just one further construction, and no more. Although it is not standard in modern English, the following is accepted.

- (27) f. 
$$\begin{array}{ccc} \text{?Marry her} & \text{will} & \text{I}_2 \\ \hline VP & S/VP & \\ \hline S & & 3 \end{array}$$

It should be obvious that no model of this kind which will accept (27d, e) - that is which will allow VP preposing and the Object Wh-question construction - will rule out (27f). We suggest that the unacceptability of the above is directly parallel to the "unacceptability" of ?Her will I marry. That is, whatever thematic fact it is that makes construction (27d) be a Wh-question has the same effect on (27f). However, since there is no Wh-pronoun for a VP, there is no grammatical construction of this form in standard English.

Not one of the remaining eighteen permutations of the four words is accepted by the algorithm. The reason is that the form of the rules imposes very powerful constraints on possible rearrangements, which appear to correspond directly to the grammatical possibilities. The most important constraint, which immediately eliminates most of the eighteen ungrammatical constructions, is a direct consequence or corollary of the fact that both of the combination rules operate only upon the two adjacent topmost items on the stack, and that the same stack is used both for movement and for building constituents. It is the following:

(28) The Adjacency Corollary

The rules will be unable to combine two items which are separated on the stack by a third, unless that intervening item can first be combined with one or the other of them.

(It is to be stressed that this is a corollary of the model, not an additional assumption). Because all tensed verbs have a category of the form (S/X)/NP, (where X is the category that the verb requires as its complement), once tense and verb have combined under the Affix Cancelling Rule, the next item to combine with the S node must be the subject NP. But because the model imposes the above general constraint, it follows that if any item intervenes between Subject and Tense, neither rule 2 nor rule 3 will be able to combine them and the analysis will block. (We are ignoring adverbials such as frequently, merely for the sake of simplification).

Among the eighteen remaining permutations, the following are predicted to be ungrammatical for just this reason: some item is interposed between Subject and Tense. (In several cases there are other reasons as well).

- |      |                             |                             |
|------|-----------------------------|-----------------------------|
| (29) | g. *I <u>marry her</u> will | m. *Will <u>her marry I</u> |
|      | h. *I <u>marry</u> will her | n. *Her I <u>marry</u> will |
|      | i. *I <u>her</u> will marry | o. *Her will <u>marry I</u> |
|      | j. *I <u>her marry</u> will | p. *Will <u>marry I</u> her |
|      | k. *Will <u>marry her</u> I | q. *Marry will <u>her I</u> |
|      | l. *Will <u>her I</u> marry | r. *Marry I <u>her</u> will |

Another consequence of the Adjacency Corollary and of the form of the rules and categories is the following: Any complement of a verb, such as an object NP, must follow the verb immediately, unless the verb is already absorbed into the S-node and the complement is topicalised, as in Wh-Question and Topicalisation (27c & d). This is because marry her in isolation must be parsed by the Forward Combination rule 2, since: (a) the

Preposed Constituent Rule 3 applies only to S/X,

- |      |                      |                      |
|------|----------------------|----------------------|
| (30) | s. *Her marry I will | u. *I will her marry |
|      | t. *Her marry will I | v. *Will I her marry |

and (b) Rule 3 will only allow complete constituents to be preposed.

- |      |                             |                             |
|------|-----------------------------|-----------------------------|
| (31) | w. *Marry <u>I will her</u> | x. *Marry will I <u>her</u> |
|      | VP/NP S/VP NP               | VP/NP S/VP NP               |

## 5. Conclusion

The rules together with the given lexical categories, predict exactly which of the 24 strings are grammatical and which are not. We show elsewhere (Ades & Steedman, 1979) that a similar result can be obtained for the permutations of the other main clause constructions. With some minor exceptions, the rules accept all and only the grammatical constructions.

Some of these exceptions show that the model has room for improvement. In particular, the system's acceptance of

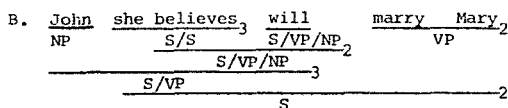
- |      |                 |                    |
|------|-----------------|--------------------|
| (32) | a. ?Slept John? | b. ?What eat they? |
|------|-----------------|--------------------|

shows that our treatment of tense, affixes and auxiliaries is imperfect, and its acceptance of

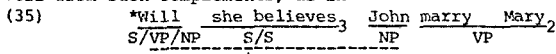
- |      |                       |                           |
|------|-----------------------|---------------------------|
| (33) | a. ?Her will I marry. | b. *This book her I gave. |
|------|-----------------------|---------------------------|

indicates further questions concerning the meaning of such thematic constructions as topicalisation. And it is clear that, although the model generalises readily to some constructions involving subordinate clauses, a number of further problems remain to be solved. (For example, although all main clause constructions can be described as arising from "leftward movement", certain complex sentences have been ascribed by transformationalists to "rightward movement". It remains to be shown that the mechanism that we have described will handle such movements.) Nevertheless, it is perhaps worth pointing out how straightforward some extensions, at least, are. The "movement" of relativised constituents in relative clauses can be handled with no extensions at all to the rules as presented. It has already been pointed out that the acceptance of these constructions will be limited to the ones allowed by such constraints as the Complex NP constraint. The same apparatus will also handle the "unbounded" extraction of constituents from embedded sentential complements. For example, on the plausible assumption that the category of the tensed verb believes is S/S/NP (a function which combines a subject and a complement S in that order to yield a sentence), it will accept the following

- |      |         |                               |             |                             |
|------|---------|-------------------------------|-------------|-----------------------------|
| (34) | A. John | she believes                  | Mary will   | marry                       |
|      | NP      | <u>NP S/S/NP</u> <sub>3</sub> | NP          | <u>S/VP/NP</u> <sub>3</sub> |
|      |         | <u>S/S</u>                    | <u>S/VP</u> | <u>VP/NP</u>                |
|      |         | <u>S/VP</u>                   | <u>S/VP</u> | <u>S/VP</u>                 |
|      |         | <u>S/NP</u>                   | <u>S/NP</u> | <u>S/NP</u>                 |
|      |         | <u>S</u>                      | <u>S</u>    | <u>S</u>                    |



Interestingly, it will correctly not accept the extraction of the tensed verb from such complements, as in



It is a hitherto unexplained constraint upon movement theories of these constructions that the movement of tensed auxiliaries is, unlike the others we have been considering, "bounded" to a single clause.

#### References

- ADES, A.E. & STEEDMAN, M.J. (1979) "On word order", Ms., Dept. of Psychology, University of Warwick, Coventry, U.K.
- BACH, E. & HORN, G.G. (1976) "Remarks on 'Conditions on Transformations'" Linguistic Inquiry, 7, 265-299.
- BRESNAN, J. (1978) "A realistic transformational grammar", in M.Halle, J. Bresnan & G. Miller, eds., Linguistic Structure and Psychological Reality, MIT Press, Cambridge, Mass.
- CHOMSKY, N. (1957) Syntactic Structures, Mouton, The Hague.
- EMONDS, J.E. (1976) A Transformational Approach to English Syntax: Root, Structure Preserving and Local Transformations, Academic Press, N.Y.
- FRAZIER, L. & FODOR, JANET DEAN (1978) "The sausage machine: a new two-stage parsing model", Cognition, 6, 291-325.
- MARCUS, M.P. (1977) A Theory of Syntactic Recognition for Natural Language, unpublished Doctoral Dissertation, MIT.
- ROSS, J.R. (1967) Constraints on Variables in Syntax, Unpublished Doctoral Dissertation, Department of Linguistics, MIT.
- WINOGRAD, T. (1972) Understanding Natural Language, Edinburgh University Press.
- WOODS, W. (1973) "Transition network grammars", in R. Rustin, ed., Natural Language Processing, Courant Computer Science Symposium 8, Algorithmics Press, New York.

Ades was supported by a fellowship from the Royal Society European Science Exchange Programme, held at the Max Planck Gesellschaft Projektgruppe für Psycholinguistik, Nijmegen. The research was also supported by a grant from the Social Science Research Council to Steedman.

## DESCRIPTION TYPES IN THE XPRT-SYSTEM

Luc Steels  
MIT, A.I. Lab.

ABSTRACT (currently at Schlumberger-Doll Research  
Ridgefield, Connecticut 06877, USA)

The XPRT-system has been designed as a basis for implementing knowledge-based expert systems. This paper introduces the description types that are currently available for communicating with this system.

## PREFACE

The XPRT-system is a collection of LISP-programs designed to serve as a basis for implementing knowledge-based expert systems. The system is fully documented in Steels (1979a). It contains mechanisms for building up constraint networks, for performing reasoning over these networks in order to solve particular problems and for communicating with a potential user. Two kinds of interactions are possible. The user can supply new information by giving general properties of objects or by describing specific problem situations. We will call these interactions predications. Or he can ask questions: (i) informative, or wh-questions, (ii) non-informative, or yes/no questions, and (iii) questions for justification. Predications cause the formation or expansion of constraint networks used to represent knowledge inside the system. Questions cause queries, expansions of the network based on consequent reasoning, etc. What these networks look like and how they are formed will not be discussed here. Instead we will concentrate on the communication language itself. An important feature of this language is that it is natural language like.

The introduction of natural language like constructs has become an important feature of recent knowledge representation languages. The prime reason for incorporating such constructs is to make the language easier to use. It is hoped that the linguistic intuitions of the user will help him to learn the language, think in terms of it, understand or remember more easily what is being expressed, etc. Once this position is adopted there are two ways to proceed. The ideal, of course, is to use a parser that takes genuine natural language expressions and a producer that is able to report back in natural language. But although this seems theoretically feasible, the required cost in manpower, resources and time which is required to construct such systems exceeds at present the resources and time limits in expert-systems research. There remain two options. Either we can use keyword parsers which can be very crude like the one used by Davis (1977) or more sophisticated as the Yale-systems. Or we can gradually incorporate natural language like constructs like articles, case mechanisms, etc. in an artificial language. This path has been explored by Bobrow and Winograd (1977), Hewitt, Attardi and Lieberman (1979) and has also been adopted in the present system.

The reason is as follows. Although the unrestricted natural language input is achievable with keyword parsers, they are not 100% reliable (at best they yield an educated guess). This is intolerable in real world applications. Using artificial languages with natural language like constructs we can approximate natural communication but the communication remains at all times accurate. There are also certain dangers with this approach. A user may have other intuitions about the workings of a particular natural language, or out of the range of meanings a certain word can have, only one particular meaning might be used in the artificial language. That is why it is important to give clear descriptions of the semantics of each construct that is being incorporated.

There is another point. The introduction of natural language like features is a powerful way to study the semantics of natural languages. Indeed by learning how to incorporate the communication tools used in natural language we gradually obtain a better understanding of natural language itself (and this will help us to deal with unrestricted natural language later). The important thing is not so much that the syntax is similar but that the underlying semantic mechanisms are the same. A similar position is taken by logicians such as Montague (1971) or Creswell (1973) who try to treat portions of natural language as a formal language.

The rest of the paper is organized as follows. First I will discuss frames and how descriptions are related to them. Then I will discuss some important properties of frames that are the basis for heuristics in finding the referent of a description. Finally I will discuss various types of descriptions and conventions for each type and conclude with a brief comparison to predicate calculus and other AI-systems.

## INTRODUCTION

## JAMES

The communication language of the XPRT-system is a frame-based language. A frame is a set of important questions about a certain subject matter. This subject matter can be anything that is viewed as a conceptual unit: an object, an action, a relation, a situation, etc. The questions of a frame are called its aspects. The aspects can be such things as parts, arguments, important properties, or related entities. Thus a frame for RESISTOR could have aspects for a resistance and two terminals. A frame for SUM could have aspects for the addend, augend and result. A frame for the action of putting a block on another one could have aspects for an actor, object, begin-situation, end-situation, old-support and new-support. Each frame has also a self-aspect which is the situation (object, relation, ...) covered by the frame. In the case of a ROOM-frame the filler of the self aspect could be the room itself, in the case of a PUT-ON frame the PUT-ON action itself, etc.

The structure of a frame is denoted as follows

```
FRAME <name-of-frame>
  (WITH <aspect-name-1> ) ... (WITH <aspect-name-n>))
; in
FRAME PUT-ON
  (WITH SELF)
  (WITH ACTOR)
  (WITH BEGIN-SITUATION)
  (WITH END-SITUATION) ...)
```

The next question is what kind of things can function as answers to the various questions posed by a frame. In other words what kind of entities can "fill" an aspect of a frame. We will employ two kinds of entities: individuals and prototypes. Individuals are members of the domain of discourse. However the individuals we will use here differ from those used in logic because they lack the uniqueness property. So each individual is a member of the universe of discourse but at any time it may turn out that it is equal to other individuals. The individuals will therefore sometimes be called "anonymous objects". A prototype is an abstraction that functions as stand-in for a whole class of entities which inherit properties from this prototype. The union of all individuals and all prototypes constitutes the domain of objects of the representation system.

A particular frame determines a set of n-tuples from all possible sets of n-tuples formed with objects from this domain. The tuples determined by a frame with n aspects are n-tuples. For example, the set of tuples of a frame for FATHER with aspects for the self and the child consists of pairs of objects which are in the father-child relation. Each tuple in this set specifies a set of objects that are in the relation the frame is about and because the tuple is ordered what role each object plays in the relation. The set of tuples of a frame is furthermore divided into two subsets: those which contain only anonymous objects (further called instantiations) and those which contain at least one prototype (further called specializations). Each of the latter type of tuples stands actually for a (possibly infinite) set of tuples.

## DESCRIPTIONS

A description is a construct used to refer to an object by saying that it fills a certain position in a tuple of a frame. The aspect associated with this position is called the view. So a description has two functions: (i) it picks out one n-tuple of the set of n-tuples of a frame and (ii) it picks out one element of this n-tuple.

Suppose for example that the set of tuples for FATHER includes (JOHN, JAMES), then there could be a description that picks out this pair. Suppose furthermore that the view of the description is self when this description would further pick out JOHN. If on the other hand the view is child, the description would pick out JAMES.

Restrictions on fillers can be provided to help pick out what tuple is intended. The restrictions are expressed by supplying descriptions of those fillers. For example the child aspect of a tuple of the FATHER frame could be further specified by saying that it is described as JAMES. Then only pairs of the form (?, JAMES) are potential candidates for the tuple referred to by this description.



The syntactic structure of a description is:

```
<description> := (<first-article> <view> OF <second-article> <frame-name>
                  (WITH <aspect-name-1> <description>) ...
                  (WITH <aspect-name-n> <description>))
```

as in

```
(THE FATHER OF A FAMILY (WITH CHILD JAMES))
```

FAMILY is the name of the frame, FATHER is the aspect which functions as view and JAMES is a further restriction on the filler of the child aspect of the n-tuple referred to by the description. In the present paper the <first-article> will always be THE. Moreover if the view is self, we will leave out THE SELF OF. For example we will write

```
(A FATHER (WITH CHILD JAMES))
```

instead of

```
(THE SELF OF A FATHER (WITH CHILD JAMES))
```

In another paper [Steels, 1979b] I will show how we can use prepositions, verbs and adjectives to get more condensed (and more natural language like) descriptions, so that we get expressions like

```
JOHN SELLS (A CAR) TO JONES FOR (30 DOLLARS)
```

instead of

```
JOHN IS (THE AGENT OF A POSSESSION-TRANSFER
         (WITH OBJECT (A CAR))
         (WITH RECIPIENT JONES)
         (WITH TRANSFER-OBJECT (30 DOLLARS)))
```

## 2. ASPECT SPECIFICATIONS

There are a number of properties of frames that are important for dealing with descriptions based on these frames. These properties will be shown to interact with certain articles in order to determine the meaning of a certain description.

### PROJECTIVITY

First of all we observe that it is usually possible to divide the set of instantiations of a frame into groups, termed instantiation-groups. Each instantiation in such a group has a certain set of aspects in common.

Here is an example. Consider a frame for family with aspects for the mother, the father and a child. Because every family has only one mother and one father, but possibly many children, it makes sense to divide the instantiation-set of the family frame into groups where each group corresponds to one family. All instantiations in this group have the same father and the same mother. They differ in that the child-slot could be filled by a different individual. We will call the aspects that have the same filler in a given instantiation-group projective aspects. The other aspects are called non-projective. It is important for the reasoner to know this because as soon as it knows one instance of an instance-group it can infer the slot-fillers of the projective aspects of all members of that group. It turns out that this knowledge is crucial for two reasons: (i) it is an important aid in determining whether a description is definite and in consequently finding the referent of the description, and (ii) it is one of the tools with which two objects can be shown to be identical.

For example, if a certain person is described as

```
(THE MOTHER OF A FAMILY (WITH FATHER JOHN))
```

and a little while later this same person is described as

```
(THE MOTHER OF A FAMILY (WITH FATHER MR-JONES))
```

then we know that John and Mr-Jones are referring to the same person because the father aspect of a family frame is projective. This means that either if Mr-Jones was already known to be a certain object and John was already known to be a certain different object, then from now these objects should be considered identical. We say in such a case that the two objects are merged. Or if Mr-Jones was not yet known, then the description Mr-Jones should be predicated for John or vice-versa.

In the present paper I will assume that all aspects are projective. In a separate paper devoted to sets, their representation and use, I will discuss nonprojectivity.

### CRITERIALITY

Now how does the reasoner know under what circumstances two instances belong to the same instantiation group? In other words, how do we specify what the criteria for dividing the set of instantiations into groups is? Another property of aspects takes care of this. Often a certain aspect can only once be filled by a certain individual. Somebody can only be the mother in one family for example. We say in such a case that this aspect is criterial. But sometimes more than one aspect has to co-operate in order to find what instantiation-group is intended. In a frame of LINE-SEGMENT like

```
(FRAME LINE-SEGMENT
  (WITH SELF)
  (WITH BEGIN)
  (WITH END)
  (WITH LENGTH)),
```

begin and end are criterial because there are no two line-segments with the same begin and the same end. Moreover there might be different collections of "criterial aspects". In the same frame the self aspect is also criterial. But the begin aspect on its own is not criterial, because there can be many lines with the same begin.

The importance of knowing what collections of aspects are criterial is that they provide essential information for finding the instantiation group of a description.

Take for example the following description:

```
(A LINE-SEGMENT (WITH BEGIN POINT1) (WITH END POINT2))
```

In order to find out what line-segment is referred to, the reasoner looks at a series of aspects that are criterial and for which the description contains specific information. In the example given here the reasoner knows that begin and end are criterial, it can therefore look whether the objects point1 and point2 are described already as the begin and end of a line-segment. If that is so it knows already about this particular instance of the line-segment frame and can infer the other individuals in this instantiation.

Criteriality is also the prime mechanism to know whether two descriptions have to be merged or not. Two descriptions are merged if they refer to the same individual. For example, when a particular object is described as

```
(A LINE-SEGMENT (WITH BEGIN POINT1) (WITH END POINT2))
```

and later it is described as

```
(A LINE-SEGMENT (WITH LENGTH 3 CM))
```

then we know that each time we are talking about the same line-segment. We know this because the SELF-slot of line-segment is criterial and therefore the object can only once be described as a line-segment.

In contrast,

```
(A FATHER (WITH CHILD GEORGE))
```

and

```
(A FATHER (WITH CHILD JOHN))
```

cannot be merged because somebody can be the father of more than one child.

### UNIQUENESS

A strong from of criteriality is when it is not only the case that an individual can only once fill a certain aspect in an instance but that there is only one individual in any domain or world that can ever fill this aspect. In such a case we say that this aspect is individuality. We will call a frame whose self-aspect is individuating an individuating frame. Descriptions based on such a frame function as rigid designators.

The importance of knowing whether an aspect is individuating is that it tells the reasoner which descriptions refer to a unique individual. The reasoner builds up a list of individuals which are accessible by way of these individual descriptions, so that the referent can be retrieved when the description is encountered.

Properties like projectivity, criteriality and uniqueness are called the properties of a frame. They are defined when the frame itself is being defined by adding specifications of the form

(~~<type-of-specification>~~ ~~<range-of-application>~~)  
after the list of aspects. The ~~<range-of-application>~~ indicates the aspects (or lists of aspects) having a specification of that type. The type-specification is

CRITERIAL if the range-of-application indicates series of aspects which are criterial.

NON-PROJECTIVE if the range-of-application indicates non-projective aspects.

INDIVIDUATING if the range-of-application indicates individuating aspects.

The default for projectivity is projective. The default for uniqueness is non-individuating and the default for criteriality is non-criterial. Here is an example

(FRAME MOTHER (WITH SELF) (WITH CHILD) (CRITERIAL (CHILD)))

### 3. DESCRIPTIONS

We adopt the subject-predicate structure familiar from natural language for expressing predications or questions. Thus for predications we write

<subject> IS <complement>

for non-informative questions we write

IS <subject> <complement>

and for informative questions we write

<question-word> IS <complement>

Question-words include WHO and WHAT.

An important property of the system is that predications may themselves act as objects in the system. This capability will however not be discussed in this paper. A <subject> and a <complement> are both descriptions. In this section we discuss the various types of descriptions currently in use.

First we make a distinction between referential and predicative descriptions. Referential descriptions are used to refer to a particular entity (which could be an individual or a prototype). Predicative descriptions are used to say that some object is an instance of some other object.

Subjects of a predication are always referential. For example in

JOHN IS A PERSON

John is used referentially. Or in

EVERY PERSON IS MORTAL

"Every person" is used to refer to the prototype of person. When the complement of the predicate has a particular object as its referent then it is referential, otherwise predicative. For example if we say

JOHN IS A PERSON

"a person" is a predicative description. John is said to be an instance of person. On the other hand if we say

JOHN IS THE FATHER OF GEORGE

the complement refers to a specific individual (George has only one father) and is therefore referential. In this case we do not say that John is an instance of "the father of George" but rather that JOHN IS the father of George.

Note that descriptions that are "attached" to the aspects of another description are actually complements. Thus in

(A FATHER (WITH CHILD (A GIRL)))

"(a girl)" is used predicatively. The description is viewed as a shorter form for

(A FATHER (WHOSE CHILD IS (A GIRL)))

Some descriptions are always referential whereas the function of others depends on the environment. First we discuss referential descriptions.

#### 3.1. REFERENTIAL DESCRIPTIONS

##### PROPER NAMES

The simplest class of descriptions are the proper names. Proper names are descriptions whose underlying frame is individuating. Proper names are written without articles.

Thus we simply write JOHN  
 based on the frame  
 (FRAME JOHN  
   (WITH SELF)  
   (INDIVIDUATING SELF)

Proper names always have a particular individual in the model as referent. Note that being an individual does not imply existence, existence is considered to be a predicate.

#### GENERIC DESCRIPTIONS

Let us call a frame-name together with a set of restrictions on the fillers of some or all of its aspects a description-structure. With every description-structure ever used there corresponds a unique tuple of prototypes which is called the prototypical tuple of that description-structure.

Thus assuming a frame for FATHER with a self and a child aspect then for the description with no restrictions there is a pair containing a prototype for every father and a prototype for the child of a father. Or if we have a description-structure based on the father frame with the restriction that the child is a girl, there would be a prototypical pair containing a prototype for "every father whose child is a girl" and a prototype for the child of such a father. If the restriction on an aspect in a description-structure is an individual that aspect is not filled by a prototype but by that particular individual itself.

There are clearly two ways in which a description-structure can be used: to refer to the prototypical tuple or to refer to a specialization or instance. Descriptions of the first type are called generic descriptions. A generic description is a description whose tuple is the prototypical tuple of its description-structure. Generic descriptions will be represented with the article EVERY.

Thus the description  
 (EVERY FATHER)

picks out the prototypical tuple corresponding to a description-structure with no restrictions attached, and picks out the filler of the self aspect in this tuple.

(THE CHILD OF EVERY FATHER)  
 picks out the filler of the child aspect of the same tuple.

There is no distinction in behavior between a generic description used predicatively or referentially. Thus assuming a frame for LOVER with aspects for the self and the loved-one, we can say (EVERY WOMAN) IS (A LOVER (WITH LOVED-ONE (EVERY MAN))) to express that every woman loves every man. The filler of the loved-one aspect is the filler of the self-aspect in the prototypical tuple of man. The referent of "(every woman)" is the filler of the self aspect in the prototypical tuple of WOMAN.

English also uses THE, A or AN, ALL and plurals to indicate generic descriptions. Such descriptions have to be paraphrased here. For example,

The president has lived in the White House since 1800  
 has to be paraphrased as

Every president has lived in the White House since 1800.  
 And

A bird is an animal  
 or  
 (All) birds are animals  
 have to be paraphrased as  
 Every bird is an animal.

#### DEFINITE DESCRIPTIONS

A definite description is a description that contains fillers for each member of one set of criteria as restrictions on its aspects. For example, the child in a frame for father is criterial because a child can have only one father. If we therefore have a description where a specific child is attached, this description is definite because the instantiation can be identified. So the following description is definite:

(A FATHER (WITH CHILD JAMES))

Definite descriptions are written with A or AN (and the reasoner will check whether the description is definite or not) but in order to improve readability, the user is also allowed to use THE instead of A with definite descriptions, as in

(THE FATHER (WITH CHILD JAMES))

or

(THE LENGTH OF THE LINE-SEGMENT (WITH BEGIN POINT-1)(WITH END POINT-2))

The latter description is definite because begin and end form a criterial set and this enables us to identify which line-segment is intended. It must be noted however that the use of THE is not taken into consideration by the system, i.e. it is syntactic sugar.

## INDEFINITE DESCRIPTIONS

An indefinite description is a description whose tuple is an instance of its underlying tuple. But the tuple cannot be recognized uniquely (and therefore "anonymous" individuals have to be postulated). Indefinite descriptions are written with the article SOME.

For example if we want to say that there is a person which is the agent of a buy-event, we can say  
(SOME PERSON) IS (THE AGENT OF A BUY-EVENT)

The referent of (SOME PERSON) is an anonymous object which is a person. Similarly we can express that there is a particular man which is the loved-one of every woman by saying:

(EVERY WOMAN) IS (A LOVER (WITH LOVED-ONE (SOME MAN)))

The filler of the loved-one aspect in this case is a particular anonymous individual which is the loved-one of every woman. In other words every woman loves this same man.

### 3.2. PREDICATIVE DESCRIPTIONS

As said before, a predicative description expresses that the subject of the predication fills an aspect in an instance of the prototypical tuple underlying the description structure of the description. The aspect it fills is the view of the predicative description. A predicative description is written with the article A or AN, as in

JOHN IS (A PERSON)

which expresses that John is a filler of the self-slot in an instance of the prototypical tuple of PERSON. Similarly in

X-1 IS (A LINE-SEGMENT (WITH BEGIN (A POINT)))

the begin of the line is described as an instance of the prototypical point.

A final example will make the various distinctions clear. Suppose we abbreviate

IS (A LOVER (WITH LOVED-ONE ...))

as LOVES (the mechanisms for doing that will be fully explained in Steels (1979b)), then we can say

(EVERY WOMAN) LOVES (A MAN)

to express that for every woman there is a man such that she loves him. It does not follow that this man is the same one for every woman. The description only predicates of the loved-one that he is a man. Compare this with

(EVERY WOMAN) LOVES (EVERY MAN)

which expresses that every woman loves every man, and

(EVERY WOMAN) LOVES (SOME MAN)

which expresses that every woman loves the same particular man.

### 3.3 ANAPHORIC DESCRIPTIONS

A final class of descriptions contains those which correspond to indefinite pronouns in natural language. These anaphoric descriptions refer to objects which were mentioned somewhere else in the text. Natural language has no good solution to the pronoun-problem. Instead it relies on the intelligence of the reader to see that referent is intended. Obviously a different solution is required here.

We introduce two types of anaphoric descriptions: Anaphoric descriptions based on the frames in which the object plays a role and anaphoric descriptions based on unique-names introduced elsewhere in the text. In principle using anaphoric descriptions based on unique-names constitutes a general method that will work in all cases. The first type is introduced to improve readability.

# ANAPHORIC DESCRIPTIONS BASED ON FRAMES

The structure of a frame-based anaphoric description is

(THE <NAME-OF-ASPECT> OF THE <NAME-OF-FRAME> )

where <name-of-frame> is the name of a frame used somewhere in a certain text i.e. a sequence of interactions, and <name-of-aspect> is the name of an aspect. When the <name-of-aspect> is SELF we simply write

(THE <NAME-OF-FRAME> ).

The referent of an anaphoric description is the object that is filling the aspect indicated by the <name-of-aspect> in the latest description that used the frame with name <name-of-frame>. When there has been no description that used the frame in an anaphoric description, the description will be considered to be indefinite.

o if we have the following conversation:

> X-1 IS (A RESISTOR)

> (THE VOLTAGE OF THE TERMINAL

(WHICH IS (THE TERMINAL1 OF THE RESISTOR))) IS 0.0

When the referent of (THE TERMINAL1 OF THE RESISTOR) is the terminal1 of X-1 because the only resistor mentioned so far is X-1. Or if we say

(EVERY FATHER) IS (A PARENT (WITH CHILD (THE CHILD OF THE FATHER)))

then

(THE CHILD OF THE FATHER)

is an anaphoric description which refers to the filler of the child aspect in the prototypical father-tuple invoked earlier. The predication expresses that the child aspect of the father-specialization is co-referential with the child aspect of the parent-specialization.

Finally in

(EVERY GRANDFATHER)

IS (A FATHER

(WITH PARENT

(A PARENT

(WITH CHILD (THE GRANDCHILD OF THE GRANDFATHER))))

The anaphoric description

(THE GRANDCHILD OF THE GRANDFATHER)

refers to the filler of the grandchild-aspect in the grandfather tuple invoked by

(EVERY GRANDFATHER)

## ANAPHORIC DESCRIPTIONS BASED ON A UNIQUE-NAME

If the same frame occurs more than once in a text, one can assign a name which is unique within that piece of text to the tuple we want to refer to later. This is done by writing a string after the frame-name used to introduce the tuple. The syntax for referring to this tuple later is

(THE <NAME-OF-ASPECT> OF <UNIQUE-NAME>)

which will pick out the element filling the aspect with name <name-of-aspect> in the tuple assigned to <unique-name>. If the aspect is the self aspect we write only the unique-name. For example, using the more elaborate language of Steels (1979b) we can say

> (EVERY OBJECT

(WHICH IS

(MOVED TO (A LOCATION B)

(AT (A TIME T))))

IS (IN B (AT T))

where B and T are anaphoric descriptions based on unique names.

## 1. CONCLUSION

In this paper I have introduced the description types that are currently being used in the XPRT-system. Of course I only discussed one part of the story. The other part is concerned with the mapping into the network representation and the behavior of these various constructs in the reasoning system. This matter will not be taken up in this paper.

A fairly straightforward mapping can be constructed to translate expressions from predicate calculus into the present language and back. Universal quantifiers translate into generic descriptions, existential quantifiers into a predicative description if governed by a universal quantifier, otherwise into an indefinite description. Definite descriptions correspond to functions and proper names correspond to constants.

The most striking difference with classical predicate calculus is that we have adopted a new ontology. In predicate calculus only individuals of the domain of discourse are allowed as referents of terms. The present system operates on the basis of "anonymous" individuals and prototypes. The adoption of a different ontology allows us to remain syntactically closer to natural language. However more important than these syntactic issues is the importance of this new ontology for reasoning. Whereas axiomatic proof theories (such as Gentzen's natural deduction system) are a natural choice for mechanizing predicate calculus based formalizations, the ontology presented here has lead towards systems based on model theoretic proof theories. This will be discussed more extensively in other papers.

The language discussed here builds further on object-oriented representation systems such as Simula Dahl, (1970) and Smalltalk (Goldberg and Kay, 1976) and frame-based knowledge representation efforts: cf. Minsky (1974), Bobrow and Winograd (1976), Martin (1979), a.o.). Detailed comparisons are however beyond the scope of this paper.

Many shades of meaning used in natural language articles are not present in their usages here. For example EVERY has the connotation of referring to the collective - as opposed to EACH which stresses individuality. Nevertheless I believe that the shades of meaning that were incorporated have been identified and used properly. Another paper is in preparation that discusses other articles, such as definite numbers (like 3) and indefinite ones (like "a few"). This can only be done within the context of representation and reasoning mechanisms that deal with sets. Still another paper will deal with negation leading to articles like NO. There it will be shown how we can deal with this in an object-oriented system by postulating "negative inheritance links".

#### REFERENCES

- Bobrow, D. and T. Winograd (1977) KRL, A Knowledge Representation Language. Cognitive Science, 1.1.
- Reswell, H. (1973) Logics and Languages.
- Ahl, O., et.al. (1975) Common Base Language, Norwegian Computing Centre, Publications S-22, Oslo.
- Davis, R. (1977) Interactive Transfer of Expertise: Acquisition of new Inference Rules. in IJCAI-1977, p. 321-328
- Goldberg, A. and A. Kay (eds.) (1976) SMALLTALK-72 Instruction Manual, Xerox Palo Alto Research Center, SSL 76-6
- Swart, C., G. Attardi and H. Lieberman (1979) Specifying and Proving Properties of Guardians for Distributed Systems. MIT-AI Working Paper 172a.
- Martin, W. (1979) Roles, Co-descriptors and the formal representation of quantified English expressions. MS. MIT-LCS, 1979.
- Minsky, M. (1974) A framework for representing knowledge. MIT-AI memo.
- Montague, R. (1971) The proper treatment of quantification in ordinary English. in Montague, R. Collected Papers, University of Chicago.
- Steels, L. (1979a) Reasoning modeled as a society of communicating experts. MIT-AI TR 542.
- Steels, L. (1979b) Using case in knowledge representation languages. To appear.

Knowledge

I am indebted to several members of the MIT Artificial Intelligence Lab for discussing the contents of this paper, especially Roger Duffy and Bill Martin who read earlier versions of this document.

## Computer Aided Evolutionary Design for Digital Integrated Systems

Gerald Jay Sussman, Jack Holloway and Thomas F. Knight, Jr.  
Massachusetts Institute of Technology  
Artificial Intelligence Laboratory

Abstract:

We propose to develop a computer aided design tool which can help an engineer deal with system evolution from the initial phases of design right through the testing and maintenance phases. We imagine a design system which can function as a junior assistant. It provides a total conversational and graphical environment. It remembers the reasons for design choices and can retrieve and do simple deductions with them. Such a system can provide a designer with information relevant to a proposed modification and can help him understand the consequences of simple modifications by pointing out the structures and functions which will be affected by the modifications. The designer's assistant will maintain a vast amount of such annotation on the structure and function of the system being evolved and will be able to retrieve the appropriate annotation and remind the designer about the features which he installed too long ago to remember, or which were installed by other designers who work with him. We will develop the fundamental principles behind such a designer's assistant and we will construct a prototype system which meets many of these desiderata.

Keywords: Computer-aided design, integrated circuits, VLSI, dependencies, constraints, engineering problem solving, layout.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's Artificial Intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research contract N00014-75-C-0643. This work was supported in part by the National Science Foundation under Grant MCS77-04828, and in part by Air Force Office of Scientific Research Grant AFOSR-78-3593.

The Problem

The integrated circuit revolution has led to a vast increase in the complexity of the electrical artifacts which can be constructed monolithically. In the design of hardware systems, we are rapidly approaching the complexity barrier which has for long been apparent in the design of software systems. The turn-around time for realization of a new design, from conception, through synthesis and debugging has become excessive; hence we are not developing new designs at a reasonable rate. This is not particularly a problem of integrated circuits, or of programming systems, but rather a fundamental problem which can best be viewed in a larger context. There are inherent limitations to the complexity that the unaided designer can control in any engineering situation - from a complex electrical system to a space vehicle or a nuclear power plant. The thrust of our proposal must be viewed as attacking the problems associated with integrated systems from this larger context.

[1]



The evolutionary nature of large engineered systems is a crucial feature of their complexity. The specifications change, the design changes, and as bugs are discovered, the implementation changes to correct them. The changes are required because it is not possible for the designers, or the potential users of a system, to foresee all of the opportunities for using the system. Also, the environment in which the system will operate is itself subject to change. Besides this external reason for the evolutionary nature of large systems, there is also an internal reason. If all of the relevant constraints were considered at once in order to arrive at a perfect solution in the first place, the details would overwhelm the designer's cognitive abilities. A more effective strategy is to start with a solution which is reasonably close to correct and modify it repeatedly until an acceptable solution is reached. [2]

What is needed is a computer aided design tool which can help an engineer deal with system evolution from the initial phases of design right through the testing and maintenance phases. We imagine a design system which can function as a junior assistant. It provides a total conversational and graphical environment. It remembers the reasons for design choices and can retrieve and do simple deductions with them. Such a system can provide a designer with information relevant to a proposed modification and can help him understand the consequences of simple modifications by pointing out the structures and functions which will be affected by the modifications. The designer's assistant will maintain a vast amount of such annotation on the structure and function of the system being evolved and will be able to retrieve the appropriate annotation and remind the designer about the features which he installed too long ago to remember, or which were installed by other designers who work with him. We will develop the fundamental principles behind such a designer's assistant and we will construct a prototype system which meets many of these desiderata.

### Engineering Problem solving

One necessary subgoal of our integrated system research program is to further develop our theory of how skilled people (such as engineers and technicians) understand deliberately constructed technological artifacts. In most engineering disciplines there is already an extensive theory of how the physical principles which underlie the operation of the artifacts are applied in any particular design. In fact most of the formal knowledge taught in engineering classes is a (mathematical) theory of how the artifacts work -- how their behavior may be derived from fundamental physical principles. But an engineer knows much more than just the physical principles and their consequences. He has a great deal of "tacit knowledge" which allows him to apply his physical knowledge efficiently to solve problems of design, synthesis and analysis. This tacit knowledge is not taught explicitly in engineering classes nor is it written in engineering texts. It is usually considered informal and unteachable, except by actual experience.

There is almost no formalized theory of how the engineer himself operates -- how he must proceed in evolving a design when given a set of requirements or even how he must proceed in understanding an existing design. There is a "competence theory" of the engineered structures, but there is no "performance theory" of the engineering process [3]. This is not surprising. The performance theory is fundamentally imperative, but

before people began to study algorithms as a subject there were no formal languages in which it was convenient to express such a theory. In fact, before this time it was not even realized that such languages were necessary. The advent of programmable computing machines placed great emphasis on the development of convenient and expressive formalisms for describing procedures. We have developed performance theories for some aspects of engineering. Such a theory is a set of rules which guide the behavior of engineers. We test our theories by implementing computer programs based on these rules which model the behavior of engineers. Successful theories are directly of practical value because they automate newly understood parts of the engineering process and can thus be turned into engineering tools.

The development of a theory of engineering performance knowledge is of considerable significance.

1. Understanding this currently tacit knowledge will result in the construction of powerful computer-aided systems for automating the routine aspects of design, construction, testing, and maintenance of complex systems. Such aids cease being luxuries and quickly become essential as the complexity of systems increases. We are already beginning to hit the complexity barrier in the long turn-around time for design of integrated circuits. We have long been on the wrong side of this barrier in the design of large software systems.
2. Making the tacit knowledge of engineers more explicit will result in the development of more effective design methodologies. We are now in the descriptive phase of development of our theories. Predictive results will improve both computer driven and human performance in developing complex systems.
3. Making the tacit knowledge of engineers more explicit will improve our ability to describe, explain, and teach the process of engineering.
4. Engineering design is an almost ideal domain in which to learn about how experts reason, and how students learn to be experts. Much of the actual competence knowledge is already formalized. Answers produced by a performance theory are thus testable. Much of the structure of, and the motivation for the performance theory is already in place as engineers have an extensive vocabulary of informal descriptions of what they are doing.
5. Results obtained in the study of design methodology for digital integrated systems may be applicable in other problem domains.

Why We Need A Sophisticated Theory of Design

The basic strategy of coping with a complex problem is to find or impose structure on the problem which allows breaking it up into manageable pieces. Each piece can then be worked on separately. This must be done so that a solution to the whole can be composed from the solution to the parts of the problem. Often a system can be partitioned into pieces which are more or less disjoint and which together cover the entire system. The total system can be understood by combining our understanding of the pieces and our understanding of the composition by which the pieces constitute the system. Similarly, each piece may be further partitioned. In this way we derive a single tree-like decomposition of the system -- a hierarchy.

This observation has resulted in a plethora of shallow methodologies which are collectively called "structured design".[4] In structured design the system under development is conceptualized as a single hierarchy where the system is recursively broken into parts, each of which represents a particular segment of its ultimate structure. These theories provide considerable power in organizing the thoughts of designers and in structuring computer-aided design systems, but they must ultimately break down in sufficiently complex real designs.

The problem is that, in sufficiently complex systems, at any stage there is usually more than one way of usefully partitioning a segment. If this is so, then a single hierarchy does not suffice to indicate all of the conceptual pieces of interest in the system. Pieces whose sub-pieces are localized by one decomposition will have those sub-pieces widely dispersed throughout another. Additionally, a single sub-piece may play several roles in each decomposition it appears in. [5]

For example, when designing a simple microprocessor, one way to proceed is to think in terms of a state-machine controller which is used to control a set of registers and data paths. The state machine may be implemented as combinational logic and a state register. In some technologies, e.g., two-phase clock dynamic MOS, a register may be expanded as a pair of linked, clocked inverters and a portion of the combinational logic may be done on each phase of the clock. Thus, in this technology there may be no single physical realization of the state register localized on the chip.

Suppose, further, that we want the register which are controlled by our state machine to be bussed together. The bus is a real conceptual entity about which the data paths are organized. We must have a description of the register array in which the bus is a localized concept so that we can say specific things about it. For example, we may want to make assertions which constrain the communications conventions. However, in a structural hierarchy there is no particular locale for the bus because the bus is structurally distributed throughout the register array.

Even worse, consider the high level block labelled "instruction decoding" in a hierarchical description. Not only is the logic for this box physically distributed, but it is also implemented with techniques which overlap other aspects of the decomposition. A good example is the selective gating of clock signals, overlapping a clock distribution function with a decoding function. Other decoding may be integrated as

part of other functional modules in the system. The decoding of the arithmetic function field, for instance, may be an integral part of the structure of the arithmetic logic unit.

Thus one aspect of developing a more powerful theory of the design process than "structured design" is the development of descriptive mechanisms which capture the power of the decomposition strategy without the restrictions on what can actually be expressed imposed by a simple hierarchical development. These problems with structured design theories are not in any way restricted to the world of digital integrated systems. Engineers in any discipline need to examine the systems they are designing from many points of view. An electrical circuit designer is often interested in the bias model of a circuit, the incremental model, the low frequency model of the incremental model, and the high frequency model of the incremental model, the noise model and the power distribution model. Each of these viewpoints imposes its own decomposition of the system under examination, and each provides structure and information to processes working from other viewpoints.

### Our Developing Theory

We are developing a performance theory of engineering design, that is, a set of rules which characterize the way in which engineers behave when confronted with a design problem. In order to express this theory we are developing a methodology adequate to capture these rules. We call this the Design Procedure System because we will use it to express the procedures which an engineer will go through in the routine development of a design. The design language has two components: the design procedure language and the design plan language. The design procedure language is a very high level language for expressing design procedures -- the sequences of actions a designer goes through in evolving a design. The design plan language provides special data structures for representing the state of a design. These are used for representing the data on which the design procedures operate.

The design plan is essentially a data structure which describes the object being designed at many levels of detail and which captures the various models which are applied to it. The design plan provides locales to hang information such as why a particular goal, say a multiplexer, was implemented in a particular way, rather than using alternative approaches. The design plan contains active data structures, called constraint boxes, which autonomously check and criticize certain aspects of the evolving design and which compute some properties of the design as consequences of others by a process we call propagation. [6]

The language of design plans is crucial to the success of this project. It must be rich enough to allow the description of complex entities which are not entirely hierarchical. It must be possible to capture the various decompositions of a system that a designer wants to think about. An entity may be described in terms of several alternate decompositions into parts. In fact it may have different names from different viewpoints. Additionally, we must be able to specify that a structure is an instance of a prototype which is an element of a previously defined class from which it inherits structure, appropriate procedures, and decompositions, and that classes may themselves be subclasses of other classes. [7] We already have considerable experience with the development of a language of design plans in two domains which are related to

integrated systems -- electrical circuits and LISP programs that manipulate data bases.

The design procedure language is concerned with formalizing the particular tasks that must be performed when attempting to develop a design plan. These tasks are described in terms of rules. The language provides design procedure primitives and means of combining simple design procedures to make compound ones. It also provides abstraction mechanisms which allow one to wrap up and generalize a particular design procedure developed in a particular design. Some of the rules that must be expressed are synthesis rules which tell how particular goals may be implemented. Other rules are for information gathering. These perform analysis on partially instantiated structures. Other rules impose constraints or critics. A critic watches for and complains about violations of rational form or violations of constraints that must be enforced. Other constraints are used to deduce some design parameters from others by propagation of constraints.

Every deduction made by any design procedure or constraint must be annotated by the name of the procedure which made the deduction and the data which went into that particular deduction. We believe that it is essential that the design system which we are envisioning be thoroughly responsible for its behavior. This is essential to the debugging of the design procedures and also it is essential to the control of the deductive system so that we can retract any assumption and all of its consequences. It is critical in the context of evolutionary design. These dependencies are also very useful to a user who wants to discover why the system believes that it does about his design -- especially if it is reminding him of some detail which we has forgotten. We have had considerable experience now with dependency-controlled data bases.

The language of design must be powerful enough to capture such subtle notions as a methodology. For example, the single level polysilicon NMOS process places enough restrictions on relationships between wire levels that we can quickly develop an idea of a "reasonable" geometric methodology. The ground and VDD wires carry power, and except perhaps at their deepest branches, are required, for reasonable voltage drops, to be run in low resistance metal. The direction of these ground and VDD lines defines a local coordinate axis. Other metal lines must be routed parallel to the power wiring, to avoid crossing it. We need a set of wires which can locally travel at right angles to the metal wiring. Either polysilicon or diffusion would serve the purpose. But, in the silicon gate technology, transistors are formed wherever there is polysilicon over diffusion. If both poly and diffusion are to be used as wires, their predominant axes must be parallel, lest unwanted transistors will be formed at their crossovers.

Lead bonding constraints locations of I/O pads to the periphery of the chip. Long standing convention defines the location of certain pads, such as power, ground, and clocks.

With a few simple constraints, we have arrived at a very clear picture of what the major wire orientation of almost any large NMOS circuit is likely to be. We will develop a way to express such geometric conventions, in the same way we describe a methodology for logic design. In the NMOS design situation, the choice available in this methodology is very small. With multi-level metal or a different process, the designer may independently be able to specify a process, a geometric design

methodology, and a logic design methodology.

### Our Design System

Our computer-aided design system is to be built around the design procedure system and an appropriate library of design procedures and design plans. We expect that designers using our system will develop additional plans and procedures which will be added to the library and thus shared with other members of the design community. A designer may at any time either generalize or specialize a procedure or plan for his immediate use. Each element of the library will be indexed for easy reference, and will be annotated by information describing how it was derived, by the application of procedures to other design plans and procedures.

The design plan/procedure languages form an extensible base upon which the designer builds a vocabulary of cliches -- a new language which he then uses to describe his system. This is not just a hardware description language, though it is certainly powerful enough to describe hardware structures. In fact it is a language in which one describes methods of design. The designer tunes the methodology to meet the architecture being implemented. Part of the design specification is generated automatically by instantiating customized abstractions. These fragments can be the basis of a library of commonly used functions and procedures.

One way of providing flexibility in either layout or logical structure is by associating design procedures with generic design plan fragments. The design languages allow one to write custom design procedures that are local to a fragment. In this way it is possible to craft very general abstractions. Simple methods may in fact be just the instantiating and interconnecting previously defined chunks of hardware. One may define more advanced methods such as "the method of running a power wire through a particular kind of register cell" or "the method of computing the pull-up ratio for driving a particular capacitive load".

This is preferable to having a macrocell library consisting of a plethora of minor variations on one theme. The design procedure language provides expressive constructs for developing appropriately tailored instantiations of the general concept. It has control structure, a sub-part naming mechanism and a vocabulary of methods for synthesis and modification. A design procedure may either construct an expansion or modify a prototype according to the parameters of the call. For example, the generic concept "multiplexor" should suffice to implement instances that vary in the number of bits, control buffering, or select decoding method. There should also be flexibility in the layout to accommodate different spacing in the array of input lines.

The design plan language is used to represent the state of a design. We envision the designer as using design procedures to manipulate the current design plan. He may refine it, modify it, extend it, study it, analyze parts of it, and use it to help debug and test hardware described by the plan. The design state of a functional block is a mixture of the instantiation state of the fragment, associated mask layout, constraints relative to other structures, design decisions, annotations, and violations. Some parts of the design may be completely

specified, but others may only exist as uninstantiated or even unspecified fragments. A group of elements can be represented by an abstraction that captures the important external shape and connections, or by an abstraction which captures the essential functionality, but suppresses the internal detail. Deductions such as rough estimates of propagation delay or chip area can be made in the face of incomplete information. This allows the designer to use a top-down approach to a complex design problem when this is appropriate.

In order to allow general design procedures there must be a uniform naming mechanism by which a conceptual entity may be referenced relative to the some current focus of attention. To this end every conceptual entity, be it a physical object or location, or a functional object, has an explicit data structure which designates it. This data structure may have several names, but it has at least one name by which it may be referenced in a uniform manner by any design procedure. Thus there are no "hidden variables" or implicit references in the system. This allows us to attach information (assertions, properties, constraints) to any object, facilitating complete documentation of the design plan.

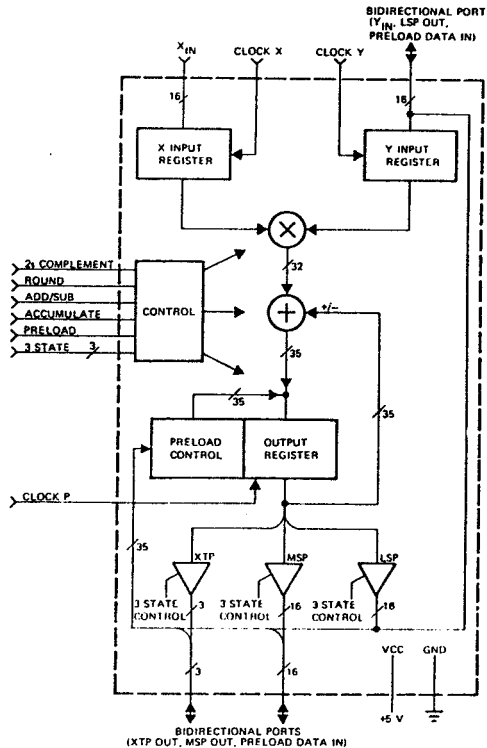
Any fact or value known by the system has a justification which describes why the system believes it. These justifications must be either that the fact was tendered by a user or that it was derived by some design procedure (or constraint) from other known facts. These justifications make it possible for a user or design procedure to consider or make incremental modifications to a design, without disturbing features of the design which are not dependent upon the incremental modification. They allow the user, in an evolutionary setting, to consider consequences of minor modifications. They also allow a user or a design procedure to determine what assumptions any fact depends upon, and how.

The system allows multiple alternate representations of the same entity, and it allows these representations to communicate. In many cases some of the representations are the result of applying design procedures to other representations. For example the maze router may be applied to a partially specified layout of a circuit segment to produce a further specified layout of that segment. Each of these representations is, however, independently manipulable by further design procedures (or by the user calling the design procedure primitives). So if the engineer really doesn't want a particular wire routed by the router to go where it put it, he can change it in the representation which was the output of the router. This will have the effect of updating the justifications in the connections between the two representations so that the new representation will be thought of descending from the old representation through the maze route except that the particular wire was changed.

### An Example of Design

We now display an example of the type of behavior that we expect from our proposed computer aided design system. While we are not entirely sure of the detailed implementation of the capabilities which we indicate, we feel confident that they are feasible.

Consider the problem of designing a parallel multiplier-accumulator which might be a component of a signal processing chip. The designer first produces a rough data path diagram (see below) showing the interconnection of instances of fragments such as registers, adders, and multipliers. This captures one decomposition that seems to be functionally appropriate and which perhaps will reflect the physical layout of the device as well. However, certain information is missing or incompletely specified.



Some deductions can already be made to fill in some of the missing information. For example, consider the adder in the middle of the block diagram. It is shown to have a 35 bit input and a 32 bit input. The adder fragment type can immediately infer the length of the adder chain and thus estimate the maximum time delay, the approximate area that the adder will take up, the power that will be needed to fuel the adder, etc. These approximate deductions can be made without further examination of the details of the adder chain because of default assumptions



stored with the adder fragment. Similar deductions can be made about other blocks in the block diagram. This information can be combined to give estimates of total delay, power consumed, and area needed --all without further refinement. In fact, we can simulate the circuit at this level of detail enough to determine that the proposed design actually has a chance of working.

Simple qualitative analysis of the circuit at this level of detail can point out potential problems and situations which will have to be attended to later. For example, critics associated with the adder fragment will notice that the instance shown does not have the same number of bits on both inputs. This either is an error or will require that the designer decide how to rectify the anomaly. The system will notify the designer of the problem and add it to the queue of pending problems which must be solved before the device is considered ready for cooking.

Next the designer directs the system's focus to a particular part of the design. Normally this is what he thinks will be the most constraining segment. For example, one common design goal in constructing circuits is to compose regular arrays of logic by abutting them. This requires that a common pitch be found. The designer is now going to try to think about his structure in terms of the pitch of the regular substructures which he will have to come up with. This is a different decomposition of the problem from the original one, and will impose different submodule boundaries on the overall device. The first stage is to examine the pitch of the default layouts of the fragments which exist in the original conception of the problem.

We now enter geometric layout space where we are given a bunch of pieces which are the external boundaries of the default layouts with the associated pitch indicated. We place the pieces in such a way as to try to limit the problem of interconnection. A good first idea (which is a default layout that the system starts us up with) is the layout implied by the block diagram. We now abut the pieces and try to adjust the pitch of the abutting pieces so that they match. We do this by communicating the constraint imposed by the unit with the largest pitch to the other fragments, which will perhaps modify their default cell layouts. This constraint is noted so that any later changes to one of the fragments will trigger a check for violations of the pitch correspondence.

A fragment has a choice of techniques by which to respond to a request to adjust its pitch. There may be a general expert that can stretch simple cells, preserving their functionality. Failing that, the fragment may contain a specific design procedure that can modify the placement of some parts in the layout prototype in order to produce a new layout with the needed pitch. Or the designer may have specified internal seams in the specification of the cell where it may be stretched without interfering with the functionality, and where wires that pass through the cell may be added. Or it may choose among a set of predefined cell layouts that the community of designers arranged to be available under the generic fragment type. Finally, the designer can interactively modify a cell to produce a new layout with the correct pitch. Of course, this interaction may be postponed at the designer's choice, with the system maintaining a record of deferred problems.

While the abutting strategy will complete many of the signal connections, there are others that will require explicit routing. For some of the cases where not much optimization is possible or desired, a

simple maze router is provided. In other cases, such as the interconnections among the output register, its preload control, and output buffers, some preplanning is required. Here, a small set of bus signals is shared among the fragments in such a way that we suspect that merging the three original blocks into one will eliminate the need for explicit interconnecting wires. We create a new cell type by editing together the corresponding layouts of the cells that we are merging. Note however that this will change the basic pitch of the system, triggering the adjustment of the pitch of the coupled fragments.

Observe that this merging operation has apparently destroyed the module boundaries implied in the original block diagram. This does not mean that this block diagram is wrong or should be discarded. It is still the right way of describing the functionality of the device, but the same simple hierarchical decomposition no longer suffices to describe both the logical and physical structure. The system must be able to manage this and, for instance, be able to give the correspondence between signal nodes in the logical and physical representations.

All through this process we assume that the system is monitoring various distributed constraints. For example, the actual propagation delay is compared to the design goal value. The dimensions of the power distribution lines depend upon the current estimate produced by the electrical modelling of the system. Changes may result in a readjustment of the line width or a design problem complaint. This monitoring is motivated by an assumption that it is probably effective to start with legal layouts and preserve the design correctness by the enforcement of constraints.

We have used such notions as "the adder fragment" without ever giving an example of what we expect such a piece of the design plan language to look like. Let us now examine the adder fragment in some detail to get a more concrete idea of what we have in mind. At the block diagram level an adder abstractly has three terminals, each of which is a word composed of a number of bits. The number of bits in each words is the same. The three terminals are called the addend, the augend and the sum. The addend and augend are inputs and the sum is an output. There is an extra bit coming out of the adder which is the carry-out and there is an extra bit coming in called the carry-in. We notate this block-diagram level description as follows:

```
(body adder (model: block-diagram)
  (parts: (addend word input)           ; declaration of parts
    (augend word input)
    (sum word output)
    (carry-out bit output)
    (carry-in bit input))
  (constraints: (= (length addend)      ; a constraint
    (length augend)
    (length sum))))
```

This fragment made use of another fragment, WORD, which is an ordered sequence of bits whose length is the number of bits.

```

(body word (model: block-diagram)
  (parameters: (length number)
                (bottom number)
                (top number))
  (sequence (enumerator: n)
            (low: bottom)
            (high: top)
            (generic: (parts: ((bit n) bit))))
  (constraints: (= (- top bottom)
                  (- length 1))))

```

The adder fragment has associated with its block-diagram level several implementation strategies. The simplest is the sequence of full adders such that the carry-out from each significant bit enters the carry-in from the next significant bit. Another implementation strategy is the carry look-ahead adder. We will only show the simple strategy here. Each strategy is attached to the fragment in the same way.

```

(implementation adder (model: block-diagram)
  (strategy-name: simple-sequence)
  (sequence (enumerator: n)
            (low: 0)
            (high: (- (length addend) 1))
            (generic: (parts: ((f n) full-adder))
                      (= ((bit (+ n(bottom addend))) addend)
                        (a1 (f n)))
                      (= ((bit (+ n(bottom augend))) augend)
                        (a2 (f n)))
                      (= ((bit (+ n(bottom sum))) sum)
                        (s (f n))))
            (between: (= (co (f *lower*)) (ci (f *upper*))))
            (= carry-in (ci (f 0)))
            (= carry-out (co (f (- (length addend) 1)))))

```

You may have noticed these descriptions of various aspects of an adder have parts that seem procedural. This is not accidental. There is a fundamental duality between object descriptions and procedures. Here we see that some aspects of an object involve design procedures which describe how to make a data structure describing that aspect of a particular object.

### Complex Design Procedures

The unique aspects of our approach to the computer-aided design of integrated systems are illustrated by the use of design procedures which are considerably more complex than ones which just instantiate parts and connect them together. For example, we can make design procedures which assign propagation delay constraints to the full adders in the implementations shown above to make it possible to get estimates of the propagation delay for the adder. This can be used to decide among the implementations when weighed against other measures such as real estate taken up on the chip and power consumed.

One powerful kind of design procedure is for editing layout structures. These procedures allow us to generalize logic cells and enhance our ability to achieve close packing and logical geometric layout of the

cells.

A way to generalize a fixed geometry is for the designer to include within the cell's definition a class of layout hints. These hints may be specified either when a cell is initially defined, or later, when a more general version is necessary. Some of these hints might concern the options available for connecting this cell with other cells. For example, in the trivial ratioed inverter, the output node is available on any of the three mask levels, metal, poly, or diffusion.

A more useful hint is the concept of a seam. A seam indicates places in the layout that have flexibility for expansion and shows how the expansion is to be done. Seams are conceptual dividing lines in a stick cell layout along which the cell may be expanded and through which specified color stick wires may be routed. The seam specifies the manner in which cell is to be expanded to make room for the newly routed wires. Seam expansion may require cell modifications such as transitions of interfering signals from one layer to another, but in the most common case, merely involves knowledge of what parts of the cell geometry expand in which directions. For example, if a seam goes vertically through a diffusion, the diffusion may be expanded in the direction perpendicular to the seam. Each seam describes what materials can be run through without shorting to a feature of the cell or manufacturing a parasitic component.

Suppose that we had to run a signal up through each cell of the adder in the multiplier-accumulator. We would invoke a design procedure:

```
(for-each cell ((f ?n) (accumulator-adder multiplier-accumulator))
  (invoke Run-Through cell Vertical Poly))
```

This iterates the application of the Run-Through procedure over each part of the accumulator-adder whose name matches (f ?n). These are the full-adders created by implementing it. Run-Through examines the vertical seams of the full-adder fragment, looking for one which can be used. If none can be found a failure message is produced which will inform the caller that he had better look for another way to accomplish his goal or that he should try to edit the full-adder cell to install a seam which can do the job. If there is an appropriate seam, the cell is stretched and the poly is run through. This changes the pitch of the cell and the interdependent objects are informed that they had better adjust to the new condition. Actually, here, things are pretty complicated. We cannot run polysilicon over a diffusion without creating an active transistor. Thus, the design procedure may only do this by changing the wire to a metal one, but this takes up lots of space so it can only be done if there is enough space at the desired pitch.

We can certainly anticipate that a moderately clever program could automatically generate seams within existing logic cells. The use of seams as "hints" to the design system is one example of how we intend to gradually develop a sophisticated design system. In the initial design system we avoid the requirement that very complex programs exist, are debugged, and are practical to use. These design system hints can gradually be augmented by sophisticated programs as they develop, but the success of the design system is not dependent on their development.

### Our Initial Efforts

There are several reasons why we feel it is important to adopt an evolutionary approach to the development of the design system, starting with the implementation of a sophisticated interactive graphic design editor. First, it is important to have some capability to design integrated circuits in the early part of the research project. It is impossible to create the advanced design environment while working in a vacuum. The early design exercises will test the significance of our ideas and also allow us to develop more insight into the nature of integrated system design. Secondly, an evolutionary growth will enforce the principle that each major module must have a clean interface so that it can later be combined with more powerful system components.

Initially we will construct an interactive design editor. We will start from the example of ICARUS. [8] Our editor will handle structural models such as logic diagrams, mask layouts, and design descriptions in a text form. It will be capable of editing several design files simultaneously with a display organized as multiple windows. Both color and high resolution black and white screens will be used.

The editor commands will be interpreted as calls to primitives of the design procedure language which will be operating on various models in the design plan. The interactive component will have a clean interface to the design procedure system. The design plans are constrained to represent meaningful structures. Thus connectedness information from the logic schematic is used during mask editing to insure that the geometric operations don't inadvertently destroy the logical function of the circuit. Paths of diffusion or polysilicon will be stretched and re-layed out as pieces of the structure are moved. The correspondence between nodes in the logical structure and the geometric layout will be maintained automatically.

Incremental corrections that require insertion of new structures will be aided by automatic editing operations that move collections of objects while preserving layout design rules.

The layout model will consist of multiple representations, such as mask geometry, STICKS schematics [9], and logical schematics. These representations can be mixed on a single page, where the more abstract representations stand for what will eventually be mask geometry on the chip. A STICKS compiler transforms the non-metric representation into mask artwork that obeys the process layout rules. Simple design procedures will be included for regular structures such as PLA's and ROMs.

### Some VLSI System Projects

We will develop several projects involving the design of particular VLSI chips. These projects cover a large range of difficulty, speculativeness and utility. Some of our projects are simple extensions or developments of existing technology which will give us some familiarity with the medium and some perspective with actual designs. We believe that it is useless to try to build tools to aid the engineering process, or to study the engineering process in the abstract, without some concrete projects to develop real engineering competence and taste.

For example, we have already developed a prototype LISP interpreter chip which has been through design and fabrication once. [10] We intend to use this chip and its successor -- a full sized interpreter and storage management system -- as a benchmark project to test some of our computer-aided design tools and methodologies as they emerge.

We also wish to design and fabricate a local-network interface chip. This is a similar "service project" chip which will help us improve our computer facilities and which will provide similar engineering exercises.

The MIT VLSI effort will build some considerably more complex systems using our computer-aided design technology. These projects will exercise our systems and methodology. Most of these projects have direct application to real world problems. We expect to support and learn from our interaction with these efforts.

We will also be interested in some specific chips for use in artificial intelligence research. One area that seems ripe for consideration is the problem of implementing processes that operate on very large stores of semantically related information, such as the semantic nets studied by Fahlman [11]. Fahlman was concerned with the fact that most problem solver programs have to labor over very simple deductions which seem instantaneous to a human. For example, if we learn that Clyde is an elephant, we can immediately answer questions such as whether Clyde is grey, or whether he can climb trees, as well as the answers to hundreds of other default facts about him. Fahlman worked out a scheme by which an important subset of these shallow but numerous deductions could be done very efficiently with specially constructed parallel hardware in the form of a network of simple, identical processing nodes with static interconnections. Fahlman's proposal is communication-intensive with almost no processing or memory at the individual nodes. All computations in a Fahlman net are done by "marker propagation". The nodes just have a few bits of memory which are used to store markers which are propagated in parallel along the static interconnections.

Implementation of marker propagation networks would be easy except for the enormous number of nodes required to construct a useful system. We estimate that a useful AI system requires at least 10 nodes. We do not yet know how to build the programmable connections, on the scale required by such a machine. Therefore, to investigate their properties these systems must be simulated, currently on general-purpose computers. Unfortunately the simulations are far too slow to be adequately tested, let alone be used as a support for other parts of a problem solving system.

We have some ideas about how such a system might be implemented and we expect that we will want to work on such an exotic architecture as part of our artificial intelligence research (in cooperation with Fahlman, who is now at CMU). One helpful constraint is that the computation is decomposable into essentially independent computational nodes such that each node's communication with other nodes is limited. When this is true, we may be able to configure a machine so that the computational nodes are allocated to segments of hardware with communications lines allocated to interconnect them. We will investigate a spectrum of such configurable architectures.

If the computational nodes and the communication channels to be established among them can be allocated at the outset, and if the set of nodes which must communicate with a given node is small, we may think of the computational problem as simulating a "wiring diagram". In fact, one interesting problem which breaks up in exactly this way is the simulation and analysis of systems which may be characterized by lumped-parameter models, such as electrical circuits. [12] In a more general setting, one can think of systems of algebraic constraints as networks which can be studied as if they were electrical circuits. A configurable architecture for such problems is constructed from a set of general-purpose processors, each of which is given the computational task of one component of the system. The architecture also requires a "patchboard" which programs the interconnectivity of the components for a problem. The patchboard may be a physical entity, such as a sorting network, or it may be virtual, such as a packet-switched network. One useful task for such a "circuit machine" is as a high-performance digital logic simulator, which can be used for experimenting with unusual computer architectures.

A class of architectures that we will investigate are network simulators. We do not really understand how to make completely parallel network machines, but there is an intermediate position. We imagine a hybrid between the conventional sequential architectures that we understand and the fully parallel architectures that have not yet been developed. With a machine of this type we can at least perform experiments on proposed parallel designs before they are constructed. A module in such a simulator consists of two parts--several large memories defining node types, node states, and interconnections, along with a VLSI interpreter engine that makes a sequential pass performing a processing step on all nodes. With several such modules interconnected, networks of a million nodes can be simulated 2 or 3 orders of magnitude faster than can be done on purely sequential machines.

The performance advantage of the hybrid network simulator comes from several sources. First, the parallelism of the simulator modules provides a straightforward factor of 8, 16 or so. Second, a dedicated memory structure internal to the module provides several times the bandwidth of the memory on a conventional machine. At each processing tick, the node's state and the state of its topological neighbors are fetched in a continuous stream of data pipelined into the interpreter engine. A network simulator in stream mode enjoys much the same advantage over conventional machines as vectorized arithmetic processors such as the Cray-1 enjoy over scalar processors. Third, unlike an instruction stream driven processor, each step of the simulator engine is interpreting an independent node. Thus pipelining and overlap can be freely used without the need of complicated interlock hardware. This freedom allows cascading several microcodable processing stages so that a multi-step node interpretation can be performed in one cycle.

Such network simulators are well suited to experimentation with proposed designs for parallel machines having large arrays of nearly uniform nodes. Some of these problem areas are digital logic simulation, marker propagation in semantic nets, and pattern matching for AI data base systems. However, in addition to being a research vehicle for parallel architectures, the hybrid sequential/parallel computer is a novel architectural paradigm that may have applications in many domains where the natural formulation of computation is object based as opposed to function based. We may find such possibilities in areas such as

signal processing and discrete particle simulations.

### Notes

1. This cognitive complexity barrier has been apparent for some time in the design of large software systems. The development of very high level languages is one approach to controlling this complexity. Software engineers have also developed methodologies such as "structured programming" [Dahl, Dijkstra & Hoare 1972] to help cope with the problem. Our Engineering Problem Solving project is an outgrowth of another approach concerned with the construction of intelligent design tools [Winograd 1973]. We are engaged in related research on the computer-aided design and analysis of analog electrical circuits [Sussman 1977a] and of software systems [Rich, Shrobe, Waters, Sussman, & Hewitt 1978].

2. Sussman [Sussman 1973] [Sussman 1977a] introduced a theory of problem solving, called Problem Solving by Debugging Almost-Right Plans, which is based on deliberately making simplifying assumptions which may introduce "bugs" into the solution. The resulting solution is then debugged until it is right. This theory was induced from observations of engineers and programmers in the process of design.

3. The distinctions between a "performance theory" and a "competence theory" for describing aspects of the behavior of humans was introduced by Chomsky [Chomsky 1965] in the context of natural linguistics. Loosely speaking, a competence theory concentrates on the factual issues of a domain whereas a performance theory is concerned with the issues of control and heuristics.

4. The power of a structured theory of design is demonstrated by Mead and Conway in their beautiful book [Mead & Conway 1979] on the design of VLSI systems. They have isolated a level of language which is natural for the design of interesting classes of NMOS chips. They speak in terms of "state machines", "programmed logic arrays", "bussed register arrays", "multiplexers" and other concepts which are primitives of a much higher level language than the AND, OR, NOT, JK flip-flop level of detail which most digital designers are used to. Using their ideas, students are able to design very complex VLSI systems with only a small amount of practice. Structured programming [Dahl, Dijkstra, & Hoare 1972] has had a similar but more controversial effect on the work of programmers.

5. The use of a special formalism for describing an electrical circuit from several points of view simultaneously, so that an automatic deductive system could make use of information deduced from each model was introduced by Sussman [Sussman 1977b]. Steele and Sussman [Steele & Sussman 1979a] have generalized the notion to be useful for the description of other "almost hierarchical systems" which result from engineering design.

6. "Propagation of constraints" was originally invented as a generalization of "Guillemin's method" of analyzing electrical ladder circuits. It was used in the analysis programs EL [Sussman & Stallman 1975] and ARS [Stallman & Sussman 1976], and in the synthesis program SYN [De Kleer & Sussman 1978]. The basic idea of the method was first described in [Brown 1975] as part of a method for localizing faults in electrical circuits. De Kleer also used propagation analysis in his fault localizer



[de Kleer 1976]. Sutherland [Sutherland 1963] appears to have developed a similar technique (the "One Pass Method") for constraint satisfaction in Sketchpad.

7. SIMULA [Dahl & Nygaard 1966] introduced the "class" as an abstraction mechanism in a programming language.

8. ICARUS is a minimal automated geometric draftsman developed at Xerox PARC by Fairbairn and Rowson [Fairbairn & Rowson 78].

9. STICKS is a semi-geometrical graphical representation of the layout of an integrated design. Features on various mask layers are represented by lines of appropriate color. STICKS diagrams show all topological information and approximate layout, but they suppress most metric information.

10. Our chip [Steele & Sussman 1979b] is an interpreter and storage manager for a dialect of LISP called SCHEME. It was part of the MIT project set for the Fall of 1978. Lynn Conway of PARC was teaching at MIT.

11. Fahlman's semantic memory scheme is described in [Fahlman 1977].

12. John Kassakian [Kassakian 1979] has a neat new approach to the simulation of complex electronic systems which he calls the "Parity Simulator". The basic idea is that he automatically configures a set of universal elements so that each simulates a device in a network and he configures the interconnection between them to be isomorphic to the interconnections in the network being simulated. This turns out to be better for many applications than the traditional approach of simulating the behavior of the equations resulting from an analysis of the network.

## REFERENCES

[Brown 1975]

Brown, Allen L. Qualitative Knowledge, Causal Reasoning, and the Localization of Failures, Ph.D. thesis. MIT (September 1975). Also MIT AI Lab Technical Report 362 (Cambridge, March 1977)

[Chomsky, 1965]

Noam Chomsky, Some Aspects of the Theory of Syntax MIT Press, Cambridge, Mass. 1965

[Dahl, Dijkstra, & Hoare 1972]

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

[Dahl & Nygaard 1966]

O.J. Dahl, and K. Nygaard, "SIMULA -- An ALGOL-based Simulation Language", Communications of the ACM, Vol. 9, No. 9, September 1966

[de Kleer 1976]

De Kleer, Johan. Local Methods for Localization of Faults in Electronic Circuits. MIT AI Lab Memo 394 (Cambridge, November 1976)

[de Kleer & Sussman 1978]

De Kleer, Johan, and Sussman, Gerald Jay. Propagation of Constraints Applied to Circuit Synthesis. MIT AI LAB Memo 485 (Cambridge, September 1978)

[Fahlman 1977]

Scott E. Fahlman, NETL: A System for Representing and Using Real-World Knowledge, PhD Thesis, MIT Department of Electrical Engineering and Computer Science, June 1977 in the MIT Press series in Artificial Intelligence, 1979

[Fairbairn & Rowson 1978]

"ICARUS: An Interactive Integrated Circuit Layout Program", Proceedings of the 15th Annual IEEE Design Automation Conference, June 1978

[Kassakian 1979]

John G. Kassagian, "Simulating Power Electronic Systems -- a New Approach", to appear in Proceedings of the IEEE, 1979

[Mead & Conway 1979]

Carver. A Mead and Lynn A. Conway Introduction to VLSI Systems, Addison Wesley, 1979

[Rich, Shrobe, Water, Sussman & Hewitt]

C. Rich, H.E. Shrobe, R.C. Waters, G.J. Sussman, and C.E. Hewitt, Programming Viewed as an Engineering Activity, MIT Artificial Intelligence Laboratory Memo 459, January 1978

[Stallman & Sussman 1976]

Stallman, Richard M., and Sussman, Gerald Jay. "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis." Artificial Intelligence 9 (1977), 135-196; also MIT Artificial Intelligence Laboratory Memo 380, September 1976

[Steele & Sussman 1979a]

Guy Lewis Steele Jr. and Gerald Jay Sussman, "Constraints", Proceedings of the STAPL sigplan Conference on APL, Rochester, New York, June 1979; also MIT Artificial Intelligence Laboratory Memo 502, November 1978

[Steele & Sussman 1979b]

Guy Lewis Steels, Jr. and Gerald Jay Sussman, "Design of LISP-Based Processors", MIT Artificial Intelligence Laboratory Memo 514, March 1979.

[Sussman 1973]

Gerald Jay Sussman, A Computer Model of Skill Acquisition, PhD Thesis, MIT Department of Mathematics, August 1973; American Elsevier Artificial Intelligence Series, New York 1975; also MIT Artificial Intelligence Laboratory Technical Report 297, August 1973.

[Sussman 1977a] Gerald Jay Sussman, "Electrical Design: A Problem for

Artificial Intelligence Research", Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge Massachusetts, August 1977.

[Sussman 1977b]

Gerald Jay Sussman, "SLICES: At the Boundary Between Analysis and Synthesis", Proceedings of the IFIP Working Conference on Artificial Intelligence and Pattern Recognition in Computer-Aided Design, Grenoble 1978; also MIT Artificial Intelligence Laboratory Memo 433, July 1977.

[Sussman & Stallman 1975]

Sussman, Gerald Jay, and Stallman, Richard M. "Heuristic Techniques in Computer-Aided Circuit Analysis" IEEE Transacation on Circuits and Systems, vol. CAS-22 (11) (November 1975).

[Sutherland 1963]

Sutherland, Ivan E. SKETCHPAD: A Man-Machine Graphical Communication System. MIT Lincoln Laboratory Technical Report 296 (January 1963).

[Williams 1977] J.D. Williams, "Sticks -- a New Approach to LSI Design", M.S.E.E. Thesis, Dept. of Electrical Engineering, MIT June 1977.

[Winograd 1973]

Terry Winograd, "Breaking the Complexity Barrier, Again". Proceedings of the ACM SIGIR-SIGPLAN Interface Meeting, Nov. 1973.

DATA COMPRESSION, GENERALISATION AND OVERGENERALISATION  
IN AN EVOLVING THEORY OF LANGUAGE DEVELOPMENT

J. Gerard Wolff

Department of Psychology, The University, Dundee, SCOTLAND, DD1 4HN.

ABSTRACT

This paper describes a computer model of first language acquisition, program SNPR14, which is a development of an earlier model, MK10/GRAM15, described elsewhere [Wolff, 1978a and b]. Like the earlier model, SNPR14 shows some success in discovering a phrase-structure grammar from an unsegmented, semantics-free, language-like text given only that text as data. SNPR14 is designed to remedy certain weaknesses in MK10/GRAM15. In particular it provides a tentative answer to the problem of how it is that children, in forming syntactic generalisations, can distinguish "correct" generalisations from overgeneralisations and can eliminate the latter whilst retaining the former.

The notion that language development and data compression are intimately related has been extended here: it is suggested that language development is, in part, a process of building a grammar in such a way that the effectiveness of the grammar as a means of compressing cognitive data is maximised for any given size of grammar.

INTRODUCTION

This paper describes in outline a partial and tentative theory of first language acquisition incorporated in a computer model, program SNPR14. The theory and the model are described in more detail in another paper (in preparation). Other work in this and related areas has been reviewed by Biermann & Feldman [1972], by Fu & Booth [1975] and by Pinker [1979]. Also relevant are studies by Anderson [1977], Cook & Rosenfeld [1976], Coulson & Kayser [1978], Gammon [1968], Hamburger & Wexler [1975], L.R. Harris [1977], Z.S. Harris [1955], Kelley [1967], Kiss [1973], Knobe & Knobe [1976], Olivier [1968], Power & Longuet-Higgins [1978], Salveter [1979] and Stolz [1965]. Reference to some of these other studies will be made at appropriate points below.

The main aim of work to date has been to construct a discovery procedure which, given only an unsegmented, semantics-free language-like text as data, can retrieve the grammar used to generate the text. It has been assumed that a solution to this "target" problem would be a useful step towards an empirically adequate theory of language development. An example of the type of grammar used and a sample of the text generated by it are shown in Fig. 1. Segment markers have been omitted because it seems unlikely that there are sufficient cues of this sort in natural language to reveal segmental structure reliably and it has been thought heuristically valuable to see what can be achieved without them. Semantic information probably plays a much bigger part in children's

discovery of language structure but, again, it has been thought useful to see whether grammars can be discovered without it. A working assumption is that insights gained using semantics-free data may help us to understand how children discover non-linguistic cognitive structures [see Wolff, 1976] and may ultimately lead to an integrated model for the acquisition of syntactic and semantic structures.

Fig. 1. A Phrase-structure Grammar and Sample of Text

```
# ----> (1)(2)(3)|(4)(5)(6)
1 ----> DAVID|JOHN
2 ----> LOVES|HATED
3 ----> MARY|SUSAN
4 ----> WE|YOU
5 ----> WALK|RUN
6 ----> FAST|SLOWLY
```

Sample: JOHNHATEDMARYDAVIDLOVESUSANYOURUNSLOWLY...

A solution to the target problem has been found in a pair of programs, MK10 and GRAM15, which have been designed to discover segmental (syntagmatic) and disjunctive (paradigmatic) linguistic structures respectively. GRAM15 operates on the "dictionary" or grammar derived by MK10 from a sample of text and, together, these two programs have proved capable of discovering the grammar used to construct texts like that shown in Fig. 1 [Wolff, 1978a and b]. As a model of language acquisition processes in children MK10/GRAM15 has a number of weaknesses. SNPR14 has been designed to remedy two of them and to provide an answer to a theoretical problem which has not previously received serious attention in this project or, apparently, elsewhere.

One weakness in the earlier model is the rigid separation of processes for discovering segmental structure from processes for abstracting disjunctive relations. It seems likely that in children these two kinds of process are integrated. MK10/GRAM15 is also unsatisfactory because it can only discover grammars successfully if MK10 is stopped exactly at the spot where all "sentence" types in the text have been isolated and no larger structures have been formed. A realistic model of acquisition processes should not depend on manual intervention of this sort.

Apart from providing remedies for these two weaknesses SNPR14 differs from the earlier model in that it can "generalise" -- it can form grammatical rules (including recursive rules) with a generative range greater than the set of character strings from which they were derived. This is a necessary feature of any model which is to account for our ability to produce and comprehend an infinite variety of grammatically acceptable sentences when we hear only a finite sample of sentences during childhood. The theoretical problem which arises when generalisations are introduced into the system is to find well-motivated principles for distinguishing "correct" generalisations which must be retained in the developing grammar from "incorrect" generalisations which must be eliminated. [Chomsky, 1957, 1965]. This is part of the general problem of deciding why it is that among the many grammars which can generate any given corpus it is usual to find one or more which are much more obviously appropriate than others. This is not merely a

problem of theoretical linguistics: a prominent feature of young children's speech is the occurrence of "overgeneralisations" like hitted and mouses which drop out of use in the course of language development. How it is that children realise that these are wrong while other generalisations which have an equal (zero) frequency of occurrence are right is the problem to be solved. Program SNPR14 and the theory on which it is based provide a tentative answer.

### COGNITIVE ECONOMY

A basic supposition of this project has been that some aspects at least of linguistic and cognitive development are manifestations of data-compression principles which have the effect of minimising information storage requirements in the brain or information transmission requirements or both. This supposition derives in part from the notion that the human nervous system is the product of evolutionary processes of natural selection which are likely to have favoured principles of efficiency in information handling [Von Bekesy, 1967; Barlow, 1969].

Efficiency is a broad term embracing information storage and transmission costs, and also such concepts as reliability, accuracy and speed of operations. It would be wrong to suppose that data compression is universal and that redundancy has no place in brain function -- several trade-offs operate under the rubric of efficiency [Wolff, 1978a]. This said, it is still of value to explore the possible role of data compression principles in language acquisition. Three such principles have been considered previously [Wolff, 1978a] and will be reconsidered here together with two others.

#### Data Compression Principles

1. A body of data like ABCDPQABCDABCDPQABCDPQRPQR may be recoded as (x)(y)(x)(x)(y)(x)(y)(y) where x ----> ABCD and y ----> PQR.
2. Two or more sequences like ABCIJKDEF and ABCLMNDEF may be recoded as ABC(x)DEF where x ----> IJK|LMN.
3. When segmental and disjunctive groupings are being chosen in 1 and 2, frequent groupings are preferable to rare ones (principle 3a) and big ones are better than little ones (principle 3b). The product of frequency and size should be maximised.
4. If a pattern is repeated as a sequence of contiguous instances then the sequence may be reduced to one instance of the pattern, coded for repetition, perhaps also with a record of the number of instances. Recursion is a coding device of this sort.
5. A crude but effective means of economising on data storage or transmission is to discard data or simply not record it. Crude as it seemingly is this principle of economy is related directly to generalisation as will be seen.

Amongst the evidence for the psychological relevance of principles 1 and

3 is the fact that program MK10, which is based on those principles, successfully identifies word segments in natural language texts and mimics aspects of vocabulary growth in children [Wolff, 1977]. Principles 2 and 3 have been applied by Rosenfeld et al. [1968] and by Kiss [1973] who have demonstrated that word groups derived from shared contexts coincide quite closely with the part of speech categorizations traditionally recognised by linguists. Grammar discovery by MK10/GRAM15 depends on the application of principles 1, 2 and 3.

So far in this discussion we have not specified what body of data is being reduced by the application of these compression principles. The grammar being built by a child clearly takes up some storage space and there is presumably some advantage in minimising this space (the size, Sg, of the grammar). However, the grammar can itself serve as an encoding device for cognitive data and we may suppose that there is also an advantage in creating a grammar which encodes such data as economically as possible. The effectiveness of a grammar or part of a grammar for compressing data (termed its "compression capacity" or CC), may be defined as  $(V-v)/V$  where  $v$  is the volume, in bits, of a body of data after encoding by the grammar and  $V$  is the volume of the data in unprocessed form. As a general rule there seems to be a trade-off between Sg and CC: small grammars tend to be inefficient coding devices and vice versa. Given this trade-off it is not obvious at first sight what relative weights should be attached to CC and Sg; a priori we do not know the relative importance, biologically, of these two measures. However this problem may be shelved if we suppose that children start with a small set of innate perceptual analysers which is equivalent to a primitive grammar and then add to this grammar in such a way that for each increase in Sg there is a corresponding increase in CC. We may suppose that grammar construction stops when the balance between Sg and CC is optimal and we may further suppose that this optimum varies from one person to another.

Whatever the balance may be for any individual, it seems clear that there is always an advantage in trying to maximise the ratio of CC to Sg for any given Sg. If a grammar is built in this way it seems that the gains in CC for unit increase in Sg will tend to be greatest in the early stages and decrease progressively as the grammar becomes larger. Additions to a grammar should tend to become progressively less "profitable" in terms of increases in CC and this may explain why the pace of language development is apparently greatest in the early years and decreases throughout childhood and beyond. Program SNPR14 will now be described in general terms to try to show how this kind of effect may be achieved using the five data compression principles set out above. This description is also intended to show how generalisation, overgeneralisation and corrections of overgeneralisation may be fitted into this scheme.

#### DESCRIPTION OF PROGRAM SNPR14

The general format of SNPR14 is like that of MK10 [Wolff, 1975, 1977] but it contains additional features designed to remedy the shortcomings of MK10/GRAM15 which were described above. Like MK10/GRAM15, SNPR14 starts with minimal (M) elements and creates elements of two main types, SYN elements which encode sequential or syntagmatic relations in the

text and PAR elements which encode disjunctive or paradigmatic relations. The term SYN element covers simple (S) elements and complex (C) elements. Each of the former is a string of two or more M elements and each of the latter is an element which contains at least one PAR element or at least one C element amongst its constituents. Since PAR elements may themselves contain M, S or C elements it is clear that C elements may be arbitrarily complex.

Program SNPR14 is, at the outset, provided with a primitive grammar of M elements which has the form # ---> A|B|C|...|Z|##. This grammar allows it to encode any (alphabetic) text by repeatedly choosing one character from the set of characters specified in the single recursive disjunctive re-write rule. This grammar illustrates the fifth compression principle because it can be regarded as a very compact (if crude) statement of the structure of the input corpus. This paucity of information about the structure of the text leads the grammar to generalise to an infinite variety of other alphabetic texts. The compactness of the grammar contrasts with its inability to introduce any compression into its encoding of the sample text -- its CC is zero. (This primitive grammar may be compared with another primitive grammar with one re-write rule of the form: # ---> the complete sample of text. Unlike the first example, this primitive grammar is not at all compact and does not generalise, but it can encode the sample text using just one bit of information. A "realistic" grammar for a text sample is some kind of compromise between these two extremes.)

Like MK10, program SNPR14 scans its text sample repeatedly, keeping a count of the frequencies of contiguous pairs of elements and, at the end of each scan, picks out the most frequently occurring pair and adds it to its "dictionary" of elements -- its grammar. In subsequent scans this pair behaves as an indivisible unit, segment or element.

On every scan a parsing system assigns segments to the text in accordance with the current state of the grammar. By always selecting the largest element that matches any portion of text it tends to maximise the average size of segments in any parsing in accordance with principle 3b.

The program thus "builds" elements and adds them to its grammar in such a way that frequent elements are selected before rare ones and, for any given frequency, elements are built to be as large as possible (principles 3a and 3b). The effect of this is that gains in CC for unit increase in Sg tend at all times to be maximised and it seems also to mean that elements are, in general, added to the grammar in descending order of CC.

SNPR14 differs from MK10/GRAM15 in that a "folding" process like GRAM15 operates after every scan so that the two processes are interwoven and interdependent. This is the integration which was missing from MK10/GRAM15. The process of finding and incorporating PAR elements is essentially the same as in GRAM15 but with one important point of difference to be described below. As in GRAM15, the "contexts" of any element are derived from those other SYN elements which contain it: ABC\*DEF is a context for LMN derived from ABCLMNDEF. A systematic search is made to find those two elements (if any) which have the highest frequency of occurrence in shared contexts. This pair is formed



into a PAR element and the reference number of the PAR element is then used to modify the elements from which the shared contexts were derived: ABCIJKDEF and ABCLMNDEF both become ABC(x)DEF (where  $x \rightarrow IJK|LMN$ ) and one of the two instances is deleted. Again, as in GRAM15, this process is repeated until PAR elements have been built to be as large as possible and no more PAR elements can be found. The overall effect of this operation (which is an application of principles 2 and 3), is to reduce Sg without materially affecting CC.

The important point of difference between this process and that in GRAM15 is that substitutions are made not only in those SYN elements from which the shared contexts of each PAR element were derived but also in all other SYN elements which contain one or other of the constituents of that PAR element. An element like QRSIMNTUV would become QRS(x)TUV even though x (with the structure IJK|LMN) was derived from ABCIJKDEF and ABCLMNDEF. This extension of the range of application of PAR elements is one of the two ways in which SNPR14 forms generalisations. It is an example of a principle proposed by Cook & Rosenfeld [1976]: whenever one production in a grammar "covers" another production (i.e., when the first production can generate the same range of terminal strings as the second production, and more) then the first production may replace the second production in the grammar. It is also an example of principle 5: we have discarded some information from the grammar and have at the same time predicted that the string QRSIJKTUV is part of the language. This prediction may or may not be true (see later). If it turns out to be true then there will be a gain in CC despite the reduction of Sg: the ratio of CC to Sg will be improved.

The generalisation mechanism which has been described has the potential for forming recursive structures without ad hoc provision because any element in which a PAR element is incorporated may itself be a constituent of that PAR element, either immediately or at depth. Consider, for example, a grammar containing elements A, B, C, BB, ABC and ABBC amongst others. A PAR element (x) with the structure B|BB may be formed from ABC and ABBC and then both of these elements will become A(x)C (and one of them will be deleted). BB will be modified to become (x)(x) so that structure x will therefore be B|(x)(x) which is recursive. Such strings as ABBBC, etc. would then fall within the generative range of the grammar. Recursion is an example of the fourth compression principle.

Program SNPR14 as it has been described so far is a system which forms S, C and PAR elements from the earliest stages of processing. This means that the parsing system has to be able to recognise elements of all types as indeed it can. It also means that when new SYN elements are formed by concatenation at the end of each scan, each of the two elements which are joined together may be of any type -- M, S, C or PAR elements. This feature of SNPR14 is the second of the two ways in which the program may form generalisations. A C element like (m)X(n)Y (where  $m \rightarrow A|B$  and  $n \rightarrow P|Q$ ) may be created by concatenation of (m)X and (n)Y when the text contains only AXPY, BXPY and AXQY amongst the four possible terminal strings of this element. The string BXQY is a generalisation predicted by element (m)X(n)Y. As with the previously described mode of generalisation, this operation can have the effect of improving the ratio of CC to Sg.

### Correction of Overgeneralisations

The two generalisation mechanisms which have been described produce an increase in the range of terminal strings which SYN elements can generate beyond the range of strings from which those elements were initially derived. Basic information theory indicates that the specification of one item out of a small set requires less information than specifying one item out of a larger set. So if the generative range of a grammar extends too far beyond the corpus from which it is derived we may expect losses in CC. There may, therefore, be some advantage in putting a limit on the generalisation process so that generative range does not become too big. In the following paragraphs a simple "rebuilding" mechanism is described which has the effect of "correcting" certain "overgeneralisations" while leaving other generalisations untouched. It is proposed as a tentative explanation of how a child eliminates utterances like hitted and mouses but retains many other generalisations as a permanent feature of his linguistic competence inspite of their zero frequency in the speech which he has heard. It seems likely that the overall effect of these mechanisms for forming generalisations and correcting overgeneralisations is to improve the ratio of CC to Sg.

Rebuilding mechanism. For a C element to be identified during a scan of the text, each constituent PAR element, at any level, must be identified by the occurrence in the text of one of the constituents of that PAR element. For example, the C element (m)X(n)Y (where m ---> A|B and n ---> P|Q) can be recognised if the string AXPY occurs in the text and in that case m is realised as A and n is realised as P. Part of the rebuilding mechanism is a system which keeps track of these realisations for all PAR elements contained within all C elements identified during each scan of the text. At the end of each scan a check is made, for each C element, to see whether any of the constituents of any of its constituent PAR elements have failed to occur in the text. When that happens the PAR elements which have not been fully realised within a given C element are "rebuilt" by removing the non-occurring constituent or constituents. Corresponding adjustments are then made in the structure of the C element (and any "lower level" C elements which may be implicated). These adjustments are made in such a way that they effect only the structure of the given C element(s). Consider, for example, the strings AXPY, BXPY, AXQY and BXQY which would identify an element (m)X(t)Y (where m ---> A|B and t ---> P|Q|R). If these were the only instances of the element in a text sample then the PAR element t would be rebuilt as an element with the structure P|Q which would replace t in the C element. If the PAR element P|Q already exists in the grammar as, say, n then the C element would become (m)X(n)Y. If P|Q does not already exist then it would be given its own number, say p, and the C element would become (m)X(p)Y.

The same point may be illustrated with a more real-life example using phonetic symbols (in which certain complexities of phonological and semantic conditioning have been glossed.) A structure like /(h)ez/ (where h ---> boks|matf|kis|maus|...) would correctly generate words like /boksez/, /matfez/ and /kizez/ but would also produce the childish form /mauzez/. If h is reduced to i (where i ---> boks|matf|kis|...) then the new structure, /(i)ez/, will not overgeneralise in this way. The adult form /mais/ may be built up independently and then at some

stage, presumably, it would be incorporated in a PAR element like `/(i)ez|mais|.../`.

The attraction of this kind of mechanism is that it can eliminate some generalisations but leaves others permanently in the grammar. If the C element `(m)X(t)Y` (where `m ---> A|B` and `t ---> P|Q|R`) is identified by the three strings `AXPY`, `BXPY` and `AXQY`, and if the string `BXQY` fails to occur in the text then `t` will be reduced to `P|Q` as before and `BXQY` will remain as a permanent generalisation because the three strings contain all of `A`, `B`, `P` and `Q` in the context of the C element.

#### Summary of SNPR14

This program may be seen as a set of interwoven processes designed to build a grammar in such a way that CC is maximised for any given Sg. The "building" process (which resembles MK10) adds elements to the grammar in approximately descending order of CC and the storage requirements of each element tend to increase as the program proceeds so that gains in CC for unit increase in Sg become progressively smaller. The "folding" process (like GRAM15) apparently has the effect of reducing Sg more than enough to compensate for any resulting increase in CC and thus helps to raise the CC/Sg ratio. The two generalisation processes seem to have the effect of producing further gains in the CC/Sg ratio. The rebuilding process may produce some increase in Sg (if new PAR elements are produced) but we may suppose that these costs are outweighed by the benefits of reducing excessive generative range in the grammar. Quantified tests of these proposals have not yet been attempted.

#### RESULTS

A text has been prepared like that shown in Fig. 1 but without any instances of the sentences `JOHNLOVESMARY` and `WEWALKFAST`. In the early scans of this text the program forms S elements only. Words like `SUSAN`, `HATED`, and `DAVID` are built up as `((((SU)S)A)N)`, `(H(((AT)E)D))` and `(D(((AV)I)D))`. At the stage when the grammar contains `SUSAN` and `UN` (part of `RUN`), the first PAR element (reference number 42) is formed with the structure `U|SUSAN`. (Both `U` and `SUSAN` share the context `*N`.) `SUSAN` and `UN` are both converted into `(42)N` and one of them is deleted. On the next scan, element `(42)N` is built into `R(42)N`, a structure which can generate `RUN` and the overgeneralisation, `RSUSAN`. On the following scan the monitoring and rebuilding process converts `R(42)N` to `RUN` and it also restores element `(42)N` to `SUSAN`. In this way the overgeneralisation is corrected. Other examples of (over)generalisations occur as the program proceeds which arise both from building, as in the above example, and also from substitution of PAR elements in elements other than those from which they were derived. All overgeneralisations are rapidly corrected.

The first "correct" PAR element to be formed is `JOHN|DAVID` (ref. 69), derived from the S elements `DAVIDHATED` and `JOHNHATED`. The two S elements are each converted into a C element, `(69)HATED`, and one of them is deleted. The program proceeds quite soon to create a C element with the structure `(69)(74)(72)` (where `69 ---> JOHN|DAVID`; `74 ---> HATED|LOVES` and `72 ---> SUSAN|MARY`). This is identical in form to one

of the two sentence patterns in Fig. 1. Soon after that, the program creates another C element (ref. 76) with the structure (77)(75)(70) (where 77 ----> WE|YOU; 75 ----> RUN|WALK and 70 ----> SLOWLY|FAST). This is the same as the other sentence pattern in Fig. 1. Notice that these two patterns have been built up despite the omission from the text of JOHNLOVESMARY and WEWALKFAST; the rebuilding mechanism cannot eliminate these generalisations from the grammar.

These two sentence patterns are built up without the need to stop the program manually at any particular point. But SNPR14 has, at present, no natural stopping point and, if it is allowed to run on, it will incorporate the sentence patterns within larger structures: their internal structure is preserved within these containing structures. At any stage after the two sentence patterns have been created, a parsing of the text by the program clearly reflects the structure which the original grammar would assign to the text and in this sense it can be said that SNPR14 successfully retrieves the original grammar. A fairly large number of elements which were formed in the course of building up the sentence patterns cease to have any function either as elements identified at the top level, or as constituents of those elements. Such non-functional elements are "garbage" which may be discarded.

SNPR14 has been run on texts other than that shown in Fig. 1. including a text whose grammar has an hierarchical organisation and one whose grammar shows recursion. The program has been less successful in these cases apparently because of a general weakness in the program and not because of the hierarchy and recursion features per se: for reasons which are not yet fully understood, the program sometimes fails to form key constituents and then it is unable to build the larger structures which contain them.

#### Concluding Remarks

Program SNPR14 is a partial model of first language acquisition which seems to offer some insight into how children may discover segmental and disjunctive groupings in language and how they may generalise beyond the language which they hear, correcting any overgeneralisations which occur. The program comes close to meeting the criteria of success which were set. It seems to offer an explanation of certain phenomena in child language other than those to which it was originally addressed, but space does not permit a discussion of these here. The immediate task for the future is to improve its performance sufficiently for meaningful results to be obtained with natural language so that comparisons between the model and children may be made in more detail.

#### REFERENCES

- ANDERSON, J.R. [1977]. Induction of augmented transition networks. Cognitive Science 1, 125-157.
- BARLOW, H.B. [1969]. Trigger features, adaptation and economy of impulses. In K.N. Leibovic (ed.) Information Processing in the Nervous System. New York: Springer-Verlag.
- BIERMANN, A.W. & FELDMAN, J.A. [1972]. A survey of results in grammatical inference. In M.S. Watanabe (ed.) Frontiers of Pattern Recognition. New York: Academic Press, pp. 31-54.

- CHOMSKY, N. [1957]. Syntactic Structures. The Hague: Mouton.
- CHOMSKY, N. [1965]. Aspects of the Theory of Syntax. Cambridge, Mass.: M.I.T. Press.
- COOK, C.M. & ROSENFELD, A. [1976]. Some experiments in grammatical inference. In J.C. Simon (ed.) Computer Oriented Learning Processes. Leyden: Noordhoff, pp. 157-174.
- COULSON, D. & KAYSER, D. [1978]. Learning criterion and inductive behaviour. Pattern Recognition 10, 19-25.
- FU, K.S. & BOOTH, T.L. [1975]. Grammatical inference: Introduction and survey 1 and 2. IEEE Transactions on Systems, Man and Cybernetics. SMC5(1), 95-111; SMC5(4), 409-423.
- GAMMON, E. [1969]. Quantitative approximations to the word. Tijdschrift van het Instituut voor Toegepaste Linguïstiek, Leuven 5, 43-61.
- HAMBURGER, H. & WEXLER, K. [1975]. A mathematical theory of learning transformational grammar. J. Mathematical Psychology 12, 137-177.
- HARRIS, L.R. [1977]. A system for primitive natural language acquisition. International J. of Man-machine Studies 9, 153-206.
- HARRIS, Z.S. [1955]. From phoneme to morpheme. Language 31, 190-222.
- KELLEY, K.L. [1967]. Early syntactic acquisition. Rand Corporation Report P-3719.
- KISS, G.R. [1973]. Grammatical word classes: a learning process and its simulation. Psychology of Learning and Motivation 7, 1-41.
- KNOBE, B. & KNOBE, K. [1976]. A method for inferring context-free grammars. Information and Control 31, 129-146.
- OLIVIER, D.C. [1968]. Stochastic grammars and language acquisition mechanisms. Unpublished Doctoral thesis, Harvard University.
- PINKER, S. [1979]. Formal models of language learning. Cognition 7, 217-283.
- POWER, R.J.D. & LONGUET-HIGGINS, H.C. [1978]. Learning to count: a computational model of language acquisition. Proceedings of the Royal Society (London) B 200, 391-417.
- ROSENFELD, A., HUANG, H.K. & SCHNEIDER, V.B. [1969]. An application of cluster detection of text and picture processing. IEEE Transactions on Information Theory IT-15(6), 672-681.
- SALVETER, S.C. [1979]. Inferring conceptual graphs. Cognitive Science 3, 141-166.
- STOLZ, W. [1965]. A probabilistic procedure for grouping words into phrases. Language and Speech 8, 219-235.
- VON BÉKÉSY, G. [1967]. Sensory Inhibition. Princeton, N.J.: Princeton University Press.
- WOLFF, J.G. [1975]. An algorithm for the segmentation of an artificial language analogue. British J. Psychology 66, 79-90.
- WOLFF, J.G. [1976]. Frequency, conceptual structure and pattern recognition. British J. Psychology 67, 377-390.
- WOLFF, J.G. [1977]. The discovery of segments in natural language. British J. Psychology 68, 97-106.
- WOLFF, J.G. [1978a]. Grammar discovery as data compression. Proceedings of the AISB/GI Conference on Artificial Intelligence, Hamburg. pp. 375-379.
- WOLFF, J.G. [1978b]. The discovery of syntagmatic and paradigmatic classes. Bulletin of the Association for Literary and Linguistic Computing 6(2), 141-158.

# AVOIDING EQUIVALENCE EXPLOSIONS IN THEOREM PROVING - PROPOSITIONAL LOGIC

Graham Wrightson  
Institut für Informatik I, Universität Karlsruhe  
75 Karlsruhe 1, Federal Republic of Germany

## 1. Introduction

At the Fourth Workshop on Automated Deduction in Austin, Texas 1979, Peter Andrews presented the following valid formula in first-order logic, which, because of the seven equivalence connectives, caused difficulties for theorem provers while trying to show the non-satisfiability of the negated formula:  $(\exists x \forall y (Px \leftrightarrow Py) \leftrightarrow (\exists x Qx \leftrightarrow \forall y Py)) \leftrightarrow (\exists x \forall y (Qx \leftrightarrow Qy) \leftrightarrow (\exists x Px \leftrightarrow \forall y Qy))$ .

In the meantime at least two implemented theorem provers have been able to handle this formula successfully: Dennis de Champeaux's implementation at the University of Amsterdam and Jörg Siekmann's at the University of Karlsruhe [3]. In the latter case, based upon connection graphs, splitting rules were used which generated two connection graphs, each consisting of about 70 clauses and 9000 links. Although both empty clauses were quickly derived much time and space had to be placed into setting up the initial graphs.

In [8] the semantic tableau method of Smullyan [7] was used to present a modified Kripke [5] proof procedure for higher-order modal logic. An implementation of this procedure [4] for first-order logic suffered from a large demand for memory space when equivalence connectives appeared in the formulas to be proved. One aspect became particularly apparent in this procedure however, and that was the obvious redundancy of information produced in the finished tableau whenever there were initial equivalence connectives in the given formula.

The path-testing methods of Bibel [2] have the advantage over resolution and semantic tableau methods of not requiring very much more memory space once the given formula has been transformed into the non-normal form matrix - with one exception however - provided the given formula contains no equivalence connectives. So in order to solve this problem we have modified the semantic tableau and path-testing methods, as will be shown in sections 4 and 5.

## 2. Notation and Definitions

In the sequel the expression consistency tree (or tree for short) will often be used synonymously for semantic tableau. The reader not familiar with this theorem proving approach or propositional logic is referred to Leblanc and Wisdom [6] or Smullyan [7].

The logic used in this paper is propositional logic. An atom (atomic formula) is a formula containing no logical connectives. A literal is an atom or the negation thereof. The formula  $\sim(A \leftrightarrow B)$  will sometimes be written as  $(A \nleftrightarrow B)$  and  $\nleftrightarrow$  referred to as a non-equivalence connective. An equivalence operand is a formula of the form  $(A \circ B)$ , where  $\circ \in \{\leftrightarrow, \nleftrightarrow\}$  and A, B are formulas. Given a formula  $(A \circ B)$ , where  $\circ \in \{\vee, \wedge, \rightarrow, \leftrightarrow, \nleftrightarrow\}$  and A, B are formulas then A is called the first operand of  $(A \circ B)$  and B is called the second operand of  $(A \circ B)$ .

Given a formula  $\Phi$  then the set Operands defined iteratively by: Operands:={ $\Phi$ }; while one of the elements  $\Phi'$  of Operands is an equivalence operand then Operands:=(Operands U {first operand of  $\Phi'$ , second operand of  $\Phi'$ }) $\Phi$ , is called the set of equivalence-free operands of  $\Phi$  provided card (Operands)>1.

An element of the set of equivalence-free operands of  $\Phi$  is called an equivalence-free operand of  $\Phi$ . A sign s for an equivalence-free operand of an equivalence operand  $\Phi$  is an element of {0,1}.

If A (or  $\sim A$ ) is an equivalence-free operand of an equivalence operand  $\Phi$  then signed A is defined as

$\sim A$  if the sign for A is 0;      A if the sign for A is 0;  
A if the sign for A is 1;       $\sim A$  if the sign for A is 1.

A branch is a sequence of formulas in a tree, such that the origin (the formula at the top (root) of the tree) is in every branch which is in the tree and that every formula which is below the origin is a successor of some previous formula. A branch is closed if it contains both a formula of the form A and a formula of the form  $\sim A$ . A tree is closed if every branch in that tree is closed. A branch (tree) is open if it is not closed. A formula is checked if one of the rules for truth-functional connectives has been applied to it. Otherwise it is unchecked.

A finished equivalence tree of an equivalence operand  $\Phi$  is a tree such that  $\Phi$  is at the origin of the tree and that at every other node in the tree there is either an equivalence operand, which must be a subformula of  $\Phi$  and checked, or an unchecked equivalence-free operand of  $\Phi$ , which is either negated or not.

The signs for the p equivalence-free operands in the branch of a finished equivalence tree is an ordered p-tuple of signs, written as the word representation  $d_1 d_2 \dots d_i \dots d_p$  with  $d_i \in \{0,1\}$ .

$S_q^p$  is the set of  $\binom{p}{q}$  q-combinations of the p signs for the p equivalence-free operands such that q of the equivalence-free operands are given the sign 1.

In section 5 we introduce the path-testing approach in which the following definitions as presented in Bibel [1] are used. It should be noted that we are using matrices only in the negative sense of Bibel [1].

Matrices are defined inductively by: (i) any literal is a matrix; (ii) if A is an equivalence operand then A is a matrix; (iii) if  $M_1, M_2, \dots, M_n$  are matrices then the set  $\{M_1, M_2, \dots, M_n\}$  is a matrix.

Given a matrix  $M=\{C_1, C_2, \dots, C_n\}$  then the  $C_i, i \in \{1, \dots, n\}$  will also be called the clauses of M.

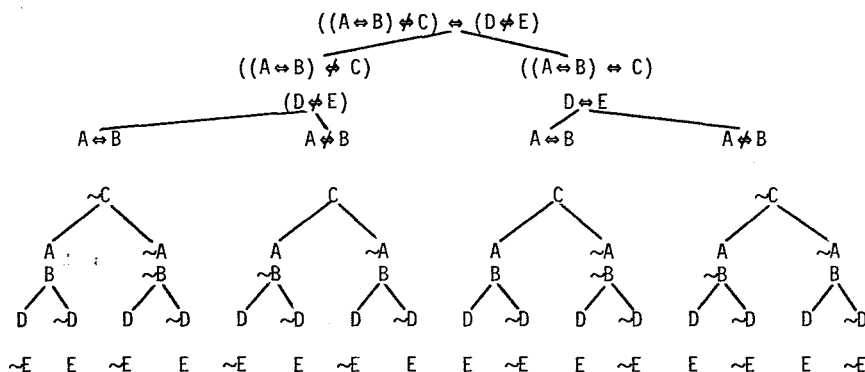
If  $L_1, L_2$  are matrices then  $L_1 \leftrightarrow L_2$  and  $L_1 \nleftrightarrow L_2$  are equivalence operands.

The formula represented by a matrix is defined inductively as follows:

(i) if the matrix is a literal A then the formula represented by the matrix is p, if A is p;  $\sim p$ , if A is  $\bar{p}$ ; (ii) if the matrix is an equivalence operand A then the formula represented by the matrix is the equivalence operand A; (iii) if  $F_1, \dots, F_n$  are the formulas represented by the matrices of a clause  $C=\{M_1, \dots, M_n\}$  of a matrix then the formula  $(F_1 \vee \dots \vee F_n)$  is represented by C; (iv) if  $F_1, \dots, F_n$  are the formulas represented by the clauses of a matrix  $M=\{C_1, \dots, C_n\}$  then the formula  $(F_1 \wedge \dots \wedge F_n)$  is represented by M.

### 3. Some Combinatorial Results

In Wrightson [9] it was shown that there are certain regularities in the combinations of the word representations, which are the signs for the equivalence-free operands in the branch of a finished equivalence tree. These regularities can be illustrated by the following tree.



This example illustrates a finished equivalence tree where A, B, C, D, E are metasymbols. By interpreting the negation in front of a metasymbol as a 0 and the absence of a negation as a 1 then the word representation for the last five metasymbols in the order A, B, C, D, E in each branch is obtained. The ordered 5-tuple for the branch on the far left is 11010, the second branch from the left is 11001, etc. The following is an enumeration of the signs for the metasymbols.

11010, 11001, 00010, 00001, 10110, 10101, 01110, 01101,  
11111, 11100, 00111, 00100, 10011, 10000, 01000, 01011

What is interesting about this list is that all ordered 5-tuples occur of five ones distributed on five places, three ones distributed on five places and one one distributed on five places i.e. the combinations given by

$$S_5^5 \cup S_3^5 \cup S_1^5.$$

For the general case these regularities can be captured in the theorem and corollary below which were proved in Wrightson [9].

**Theorem:** Given the  $n$  equivalence-free operands of an equivalence operand  $\Phi$ . If the number of non-equivalences found during the construction of the set Operands is even, resp. odd, then the set of word representations of signs for the equivalence-free operands in the branches of the finished equivalence tree of  $\Phi$  is given by

$$1. \bigcup_{i=0}^{n/2} S_{n-2i}^n \quad \text{for } n \geq 2 \text{ and even,} \quad 2. \bigcup_{i=0}^{(n-1)/2} S_{n-2i}^n \quad \text{for } n \geq 3 \text{ and odd,}$$

respectively

$$3. \bigcup_{i=1}^{n/2} S_{n-(2i-1)}^n \quad \text{for } n \geq 2 \text{ and even,} \quad 4. \bigcup_{i=1}^{(n+1)/2} S_{n-(2i-1)}^n \quad \text{for } n \geq 3 \text{ and odd.}$$



Corollary: The number of combinations in each of the sets 1. to 4. of the Theorem is given by

$$\begin{array}{ll}
 1. \sum_{i=0}^{n/2} \binom{n}{n-2i} = 2^{n-1} & 2. \sum_{i=0}^{(n-1)/2} \binom{n}{n-2i} = 2^{n-1} \\
 3. \sum_{i=1}^{n/2} \binom{n}{n-(2i-1)} = 2^{n-1} & 4. \sum_{i=1}^{(n+1)/2} \binom{n}{n-(2i-1)} = 2^{n-1}
 \end{array}$$

#### 4. A Modified Semantic Tableau Proof Procedure

The following proof procedure PROOF is based upon that for semantic tableaux as developed in Leblanc and Wisdom [6] and incorporates the ideas shown in the previous section, i.e. the two truth-functional rules for equivalence operands are excluded. Instead, all branches which would otherwise be generated and stored by the use of these two rules, are now generated but not stored. All that is stored is the set of Operands and the information needed for computing the combinations of signs for each branch.

As is also the case for the next section, PROOF and the various procedures called by it are primarily intended to reflect the logical structure of the approach and aspects of programming techniques are only secondary. At the top level, once the negated formula  $\Phi$  has been entered to the procedure, PROOF calls the procedure PROVE.

PROOF ( $\Phi$ )

if PROVE( $\Phi$ ) then return "negation of  $\Phi$  is valid"  
 else return "negation of  $\Phi$  is not valid"

END-OF-PROOF

PROVE ( $\Phi$ )

$\Phi \rightarrow$  TREE; APPLY-RULES(TREE); return true

END-OF-PROVE

PROVE calls APPLY-RULES after first having placed  $\Phi$  at the origin of the consistency TREE. As long as any of the truth-functional RULES (as in [6] but without the two mentioned above) are APPLICABLE-TO the TREE then these are applied by APPLY-RULES. Those BRANCHes which do not contain an unchecked equivalence operand are now TESTed for CLOSURE. If one is not closed then the value 'false' is returned and the whole PROOF procedure terminated. Otherwise the mechanism for handling equivalence operands is initiated. This first CHECKs the equivalence operand in question  $\Phi$ , and then calls a procedure to DETERMINE the COMBINATIONS, the OPERANDS, the NUMBER of OPERANDS, the number of COMBINATIONS and the PARAMETERS for the UNION as given in the theorem, section 3. The OPERANDS are then APPENDED to the BOTTOM OF the BRANCH CONTAINING  $\Phi$ . For each combination the SIGNS are GENERATED on the OPERANDS in the TREE' and this results in a new TREE' which, if it is not closed, causes the return of false.

APPLY-RULES(TREE)

while RULE-APPLICABLE-TO(TREE) do APPLY-RULE(TREE)  $\rightarrow$  TREE;  
 while TEST-BRANCH-APPLICABLE-TO(TREE)  
 do [TEST-BRANCH(TREE)  $\rightarrow$  CLOSURE; if CLOSURE equal notclosed then return false]

```

while UNCHECKED-EQUIVALENCE-OPERAND-IN(TREE)
  do [SEEK-UNCHECKED-EQUIVALENCE-OPERAND-IN(TREE) →
      (Φ', BRANCH-CONTAINING-Φ'); CHECK (Φ');
      DETERMINE-COMBINATIONS(Φ') → (OPERANDS, NUM-OPERANDS, NUM-COMBINATIONS,
                                     UNION-PARAMETERS);
      AT-BOTTOM-OF-BRANCH-CONTAINING-Φ'-APPEND(BRANCH-CONTAINING-Φ',
                                                OPERANDS) → TREE';
      while NUM-COMBINATIONS not equal 0
        do [GENERATE-SIGNS-ON-OPERANDS-IN-TREE'(TREE', UNION-PARAMETERS,
                                                NUM-OPERANDS) → TREE';
            if not PROVE(TREE') then return false;
            NUM-COMBINATIONS -1 → NUM-COMBINATIONS];] return true;
END-OF-APPLY-RULES

```

DETERMINE-COMBINATIONS calls DETERMINE-OPERANDS which computes the OPERANDS, the number of NON-EQUIVALENCE connectives and the number of OPERANDS according to the definition of Operands in section 2. Then the PARAMETERS for the UNION of the combinations and the number of COMBINATIONS are produced as given in the theorem and corollary.

```

DETERMINE-COMBINATIONS(Φ)
  DETERMINE OPERANDS(Φ) → OPERANDS, NUM-NON-EQUIV, NUM-OPERANDS;
  if EVEN(NUM-OPERANDS) and EVEN(NUM-NON-EQUIV)
    then (N, N-2I, 0, N/2) → UNION-PARAMETERS;
  if not EVEN(NUM-OPERANDS) and EVEN(NUM-NON-EQUIV)
    then (N, N-2I, 0, (N-1)/2) → UNION-PARAMETERS;
  if EVEN(NUM-OPERANDS) and not EVEN(NUM-NON-EQUIV)
    then (N, N-(2I-1), 1, N/2) → UNION-PARAMETERS;
  if not EVEN(NUM-OPERANDS) and not EVEN(NUM-NON-EQUIV)
    then (N, N-(2I-1), 1, (N+1)/2) → UNION-PARAMETERS;
  SUM(NUM-OPERANDS) → NUM-COMBINATIONS
END-OF-DETERMINE-COMBINATIONS

```

## 5. A Modified Matrix Path-Testing Procedure

In this section a matrix path-testing approach similar to that in [2] is presented. The differences, as can be expected, are caused by the treatment of the equivalence operands, particularly with regard to the testing of 'pure' literals. It became necessary to introduce recursive procedure calls at certain places because the systematic approach of algorithm 4.1 A in [2] could not always be maintained at those places in which equivalence operands occurred.

As in section 4, the formula to be tested for validity is first negated and the result  $\Phi$  is then entered to PROOF. PROVE, now a recursive procedure, is then called. PUSH, which acts upon the stack WAIT, is the same as that in Bibel's original algorithm with one exception, namely, whenever an equivalence operand has to be PUSHed. In this case the equivalence operand, ACT and M are placed on the stack along with information calculated by DETERMINE-COMBINATIONS (as in last section) which is called by PUSH. POP also has the extra job, whenever it has to POP an equivalence operand, of initializing the generation of the next combination of signs (similar to GENERATE-SIGNS-ON-OPERANDS-IN-TREE, in previous section) and returning the signed operands as a clause C. Of course, all the other details of reducing NUM-COMBINATIONS by 1 and noting the last combination of signs generated for the next POP call, must also be handled by POP.

```

PROVE( $\Phi$ )
1   $\Phi \rightarrow M$ ;
2   $ACT \rightarrow \emptyset$ ; while non-empty(WAIT) do POP(WAIT); PUSH(WAIT, "1m");
3  choose a clause C from the matrix M;  $M \leftarrow M \setminus C$ ;
4  if C is an equivalence operand then [PUSH(WAIT, (C, ACT, M));
   POP(WAIT)  $\rightarrow$  (C, ACT, M); if PROVE(C, ACT, M) then goto 20 else return false]
5  choose a matrix M' from C; if  $C \setminus M' \neq \emptyset$  then PUSH(WAIT, (C \setminus M', ACT, M));
6  if M' is not a literal and not an equivalence operand
   then [PUSH(WAIT, "1m");  $M \leftarrow M' \cup M$ ; goto 20]
7  if M' is not a literal but is an equivalence operand
   then [PUSH(WAIT, "1m"); PUSH(WAIT, (M', ACT, M)); POP(WAIT)  $\rightarrow$  (C, ACT, M);
   if PROVE(C, ACT, M) then goto 3 else return false]
8   $L \leftarrow M'$ ;  $ACT \leftarrow ACT \cup L$ ;
9  if  $M = \emptyset$  then return false;
10 if there is no clause  $C \in M$  such that either L or  $\bar{L}$  occurs in C
   then [while POP(WAIT)  $\neq$  "1m" do POP(WAIT); POP(WAIT); goto 3]
11 if there is no equivalence clause  $C \in M$  such that  $\bar{K}$  occurs in C for some  $K \in ACT$ 
   then [while POP(WAIT)  $\neq$  "1m" do POP(WAIT);  $ACT \leftarrow \emptyset$ ; goto 3]
12 else [choose  $K \in ACT$  such that  $\bar{K}$  occurs in M;  $L \leftarrow K$ ];
13 choose a clause  $C \in M$  such that if neither  $\Leftrightarrow$  nor  $\nexists$  occurs in C
   then  $\bar{L}$  occurs in C else  $\bar{L}$  or L occurs in an equivalence operand in C;  $M \leftarrow M \setminus C$ ;
14 (same as line 4)
15 choose a matrix  $M' \in C$  such that if neither  $\Leftrightarrow$  nor  $\nexists$  occurs in M'
   then  $\bar{L}$  occurs in M' else  $\bar{L}$  or L occurs in an equivalence operand in M';  $C \leftarrow C \setminus M'$ ;
16 if M' is an equivalence operand then [PUSH(WAIT, (M', ACT, M));
   POP(WAIT)  $\rightarrow$  (C, ACT, M); if PROVE(C, ACT, M) then goto 20 else return false]
17 for each  $C' \in M$  such that  $C'$  consists of a single literal  $K \neq L$  do
   [check each entry on WAIT (except those whose C-parts are equivalence
   operands) from the top until the first occurrence of "1m" whether for
   its C-part, C", we have  $\bar{K} \in C$ "; if this is true then cancel this occurrence
   of  $\bar{K}$ ; if even  $C' = \{\bar{K}\}$  then remove the whole entry from WAIT];
18 among the matrices of  $\bar{C}$  delete those which are literals K such that  $\bar{K} \in ACT$ ;
19 if  $C \neq \emptyset$  then PUSH(WAIT, (C, ACT, M)); if  $M' \neq \bar{L}$  then [PUSH(WAIT, "1m"; goto 23]
20 while POP(WAIT) = "1m" do POP(WAIT); if EMPTY(WAIT) then return true;
21 if NEXT(WAIT) is an equivalence operand then [POP(WAIT)  $\rightarrow$  (C, ACT, M);
   if PROVE(C, ACT, M) then goto 20 else return false]
22 else [POP(WAIT)  $\rightarrow$  (C, ACT, M); goto 7]
23 choose a clause  $C \in M$  such that if neither  $\Leftrightarrow$  nor  $\nexists$  occurs in C
   then L occurs in C else  $\bar{L}$  or L occurs in an equivalence operand in C;
    $M \leftarrow (M \setminus C) \cup M$ ;
24 (same as line 4)
25 choose a matrix  $M' \in C$  such that if neither  $\Leftrightarrow$  nor  $\nexists$  occurs in M'
   then  $\bar{L}$  occurs in M' else  $\bar{L}$  or L occurs in an equivalence operand in M';
    $C \leftarrow C \setminus M'$ ;
26 (same as line 16)
27 goto 19
END-OF-PROVE

```

The reader is referred to Bibel's paper in these proceedings for an explanation of the notation in the above algorithm.

## 6. Conclusion and Prospects

It can easily be seen that the above approach reduces the storage demands from  $O(2^n)$  to  $O(n)$  except for a small constant overhead. In principle the approach can be transferred to first-order logic [10] although the saving will not always be as great. An implementation of procedures similar to those in sections 4 and 5 but for first-order logic are currently being carried out by the authors of [4].

## Acknowledgements

Thanks are due to Wolfgang Bibel, Karl Dürre and Frank-Peter Schmidt-Lademann for numerous discussions, as well as to Prof. Alfred Schmitt under whose auspices this research was carried out.

## Literature

- [1] Bibel, W.: Tautology Testing with a Generalized Matrix Reduction Method. Theoretical Computer Science 8, (1979).
- [2] Bibel, W.: A Comparative Study of Several Proof Procedures. Internal Report, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, 1979. (also in proceedings of this conference)
- [3] Eisinger, N.; Siekmann, J.; Unvericht, E.: The Markgraf Karl Refutation Procedure. Interner Bericht, Institut für Informatik I, Universität Karlsruhe, 1979. (also in proceedings of this conference)
- [4] Fortenbacher, A.; Gutmann, M.; Schreck, M.; Wrightson, G.: Implementation and Results of a Proof Procedure for First-Order Modal Logic. Fakultät für Informatik, Universität Karlsruhe, 1980 (forthcoming).
- [5] Kripke, S.A.: Semantical Analysis of Modal Logic. Normal Modal Propositional Calculi. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 9, (1963).
- [6] Leblanc, H.; Wisdom, W.A.: Deductive Logic. Allyn and Bacon, 1972.
- [7] Smullyan, R.M.: First-Order Logic. Springer, 1968.
- [8] Wrightson, G.: A Proof Procedure for Higher-Order Modal Logic. Proceedings Fourth Workshop on Automated Deduction, Austin, Texas, 1979.
- [9] Wrightson, G.: On the Treatment of Equivalence Connectives in Automated Theorem Proving. Interner Bericht 18/79, Fakultät für Informatik, Universität Karlsruhe, 1979.
- [10] Wrightson, G.: Avoiding Equivalence Explosions in Automated Theorem Proving-First-Order Logic (forthcoming).

